VMS

VMS DECwindows Xlib Routines Reference Manual: Part II

Order Number: AA–MG27A–TE

VMS DECwindows Xlib Routines Reference Manual

Order Number: Part II: AA–MG27A–TE

December 1988

This manual describes the VMS DECwindows Xlib programming routines.

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

digital equipment corporation
maynard, massachusetts

December 1988

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDA
DDIF
DEC
DECnet
DECUS
DECwindows
DIGITAL
LN03
MASSBUS
PrintServer 40
Q-bus
ReGIS
ULTRIX
UNIBUS
VAX
VAXcluster
VAX RMS
VAXstation
VMS
VT
XUI
digital™

The following are third-party trademarks:

PostScript is a registered trademark of Adobe Systems, Inc.

X Window System, Version 10 and its derivations (X, X10, X Version 10, X Window System) are trademarks of the Massachusetts Institute of Technology.

X Window System, Version 11 and its derivations (X, X11, X Version 11, X Window System) are trademarks of the Massachusetts Institute of Technology.

ZK4732

8 Property Routines

Properties are used to store information about a window. This information can then be shared among programs and with the window manager. Properties are not interpreted by the server; they are interpreted by the program sending data to or receiving data from them.

Xlib contains routines that let you perform the following property operations:

- Set or retrieve values in specified properties
- Change a property for a specified window
- Delete the association between a property and its identifier
- Return description information for a property
- Return the atom name for an atom identifier
- Return the atom identifier for an atom name
- List properties associated with a specified window
- Shift the positions of properties within a property array

For concepts related to property routines and information on how to use property routines, see the *VMS DECwindows Xlib Programming Volume.*

The routines described in this chapter are listed in Table 8–1.

Table 8–1 Property Routines

Routine Name	Description
CHANGE PROPERTY	Changes one property for a specified window
CLEAR ICON WINDOW	Dissociates the icon window from the regular window
CONVERT SELECTION	Request to send a window selection event notification
DELETE CONTEXT	Deletes an entry for a specified window and context type
DELETE PROPERTY	Deletes the association between a property and a specified window
FETCH BUFFER	Returns the data stored in the specified cut buffer
FETCH BYTES	Returns the data stored in cut buffer zero
FETCH NAME	Provides the name for the specified window (if one exists in the WM_NAME property)

(continued on next page)

Table 8–1 (Cont.) Property Routines

Routine Name	Description
FIND CONTEXT	Obtains the data associated with a specified window and context type
GET ATOM NAME	Returns the atom name for the specified atom identifier
GET CLASS HINT	Obtains the class of a specified window.
GET ICON NAME	Obtains the name that a window wants displayed in its icon
GET ICON SIZES	Obtains the recommended icon sizes from a window manager
GET ICON WINDOW	Returns the identifier for the icon window associated with the specified window
GET NORMAL HINTS	Obtains recommended values for the size and location of a regular window
GET RESIZE HINT	Obtains for a window manager program the recommended window size and assigns it to an application program
GET SELECTION OWNER	Returns the owner of the specified window selection
GET SIZE HINTS	Obtains window size hints for any property using the WM_SIZE_HINTS format
GET TRANSIENT FOR HINT	Obtains the WM_TRANSIENT_FOR property of a specified window.
GET WINDOW PROPERTY	Returns type, format, and description information for one property associated with a window
GET WM HINTS	Obtains the window manager hints contained in the window manager hints property
GET ZOOM HINTS	Obtains recommended values for the size and location of a window in its zoomed state
INTERN ATOM	Returns the atom identifier for the specified atom name
LIST PROPERTIES	Returns a list of all properties associated with a window
ROTATE WINDOW PROPERTIES	Shifts the positions of the properties within the property array
SAVE CONTEXT	Saves the data associated with a specified window and context type
SET CLASS HINT	Sets the class of a specified window.
SET COMMAND	Sets the command used to invoke an application program
SET ICON NAME	Specifies a name to be displayed when the icon for a window is displayed

(continued on next page)

Table 8–1 (Cont.) Property Routines

Routine Name	Description
SET ICON SIZES	Sets the recommended sizes for the icon for a window
SET ICON WINDOW	Sets and displays an icon for the specified window
SET NORMAL HINTS	Sets recommended values for the size and location of a regular window
SET RESIZE HINT	The recommended window size for use by window manager programs
SET SELECTION OWNER	Sets the owner for the specified window selection
SET SIZE HINTS	Specifies window size hints for any property using the WM_SIZE_HINTS format
SET STANDARD PROPERTIES	Sets the window name, icon name and pixmap to be displayed when window is iconified, command name and arguments, and window sizing hints for the specified window
SET TRANSIENT FOR HINT	Sets the WM_TRANSIENT_FOR property of a specified window.
SET WM HINTS	Sets the values for the window manager hints
SET ZOOM HINTS	Sets recommended size and location for a window in the zoomed state
STORE NAME	Assigns a name to a window
UNIQUE CONTEXT	Creates a unique context type

8.1 Size Hints Data Structure

The size hints data structure is used to specify window hints for regular windows and zoom windows. The window manager may not adhere to these recommendations.

The data structure for the VAX binding is shown in Figure 8–1, and information about members in the data structure is described in Table 8–2.

Figure 8–1 Size Hints Data Structure (VAX Binding)

x$l_szhn_flags	0
x$l_szhn_x	4
x$l_szhn_y	8
x$l_szhn_width	12

(continued on next page)

Figure 8–1 (Cont.) Size Hints Data Structure (VAX Binding)

x$l_szhn_height	16
x$l_szhn_min_width	20
x$l_szhn_min_height	24
x$l_szhn_max_width	28
x$l_szhn_max_height	32
x$l_szhn_width_inc	36
x$l_szhn_height_inc	40
x$l_szhn_mnas_x	44
x$l_szhn_mnas_y	48
x$l_szhn_mxas_x	52
x$l_szhn_mxas_y	56

Table 8–2 Members of the Size Hints Data Structure (VAX Binding)

Member Name	Contents
X$L_SZHN_FLAGS	Defines which members the client is assigning values to
X$L_SZHN_X	The x-coordinate that defines window position
X$L_SZHN_Y	The y-coordinate that defines window position
X$L_SZHN_WIDTH	Defines the width of the window
X$L_SZHN_HEIGHT	Defines the height of the window
X$L_SZHN_MIN_WIDTH	Specifies the minimum useful width of the window
X$L_SZHN_MIN_HEIGHT	The minimum useful height of the window
X$L_SZHN_MAX_WIDTH	Specifies the maximum useful width of the window
X$L_SZHN_MAX_HEIGHT	The maximum useful height of the window
X$L_SZHN_WIDTH_INC	Defines the increments by which the width of the window prefers to be resized
X$L_SZHN_HEIGHT_INC	Defines the increments by which the height of the window prefers to be resized
X$L_SZHN_MNAS_X	With the X$L_SZHN_MNAS_Y member, specifies the minimum aspect ratio of the window

(continued on next page)

Table 8–2 (Cont.) Members of the Size Hints Data Structure (VAX Binding)

Member Name	Contents
X$L_SZHN_MNAS_Y	With the X$L_SZHN_MNAS_X member, specifies the minimum aspect ratio of the window.
X$L_SZHN_MXAS_X	With the X$L_SZHN_MXAS_Y member, specifies the maximum aspect ratio of the window
X$L_SZHN_MXAS_Y	With the X$L_SZHN_MXAS_X member, specifies the maximum aspect ratio of the window. Setting the minimum and maximum values indicates the preferred range of the size of a window. An aspect ratio is expressed in terms of a ratio between x and y. For example, if the minimum value of x is 1 and y is 2, and the maximum value of x is 2 and y is 5, then the minimum window is 1/2, and the aspect ratio maximum is 2/5. Therefore, in this case, a window could have a width of 300 pixels and a height of 600 pixels minimally, and maximally a width of 600 pixels and a height of 1500 pixels.

The data structure for the MIT C binding is shown in Figure 8–2, and information about members in the data structure is described in Table 8–3.

Figure 8–2 Size Hints Data Structure (MIT C Binding)

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

Table 8–3 Members of the Size Hints Data Structure (MIT C Binding)

Member Name	Contents
flags	Defines which members the client is assigning values to.
x	The x-coordinate that defines window position.
y	The y-coordinate that defines window position.
width	Defines the width of the window.

(continued on next page)

Table 8–3 (Cont.) Members of the Size Hints Data Structure (MIT C Binding)

Member Name	Contents
height	Defines the height of the window.
min_width	The minimum useful width of the window.
min_height	The minimum useful height of the window.
max_width	The maximum useful width of the window.
max_height	The maximum useful height of the window.
width_inc	Defines the increments by which the width of the window prefers to be resized.
height_inc	Defines the increments by which the height of the window prefers to be resized.
min_aspect_x	With the min_aspect_y member, specifies the minimum aspect ratio of the window.
min_aspect_y	With the min_aspect_x member, specifies the minimum aspect ratio of the window.
max_aspect_x	With the max_aspect_y member, specifies the maximum aspect ratio of the window.
max_aspect_y	With the max_aspect_x member, specifies the maximum aspect ratio of the window. Setting the minimum and maximum values indicates the preferred range of the size of a window. An aspect ratio is expressed in terms of a ratio between x and y. For example, if the minimum value of x is 1 and y is 2, and the maximum value of x is 2 and y is 5, then the minimum window is 1/2, and the aspect ratio maximum is 2/5. Therefore, in this case, a window could have a width of 300 pixels and a height of 600 pixels minimally, and maximally a width of 600 pixels and a height of 1500 pixels.

8.2 Icon Size Data Structure

Usually window manager programs use this data structure to set the WM_ICON_SIZE property. Application programs can then read the values specified in this property to size icon windows in cooperation with the window manager.

The data structure for the VAX binding is shown in Figure 8–3, and information about members in the data structure is described in Table 8–4.

Figure 8–3 Icon Size Data Structure (VAX Binding)

x$l_icsz_min_width	0
x$l_icsz_min_height	4
x$l_icsz_max_width	8
x$l_icsz_max_height	12
x$l_icsz_width_inc	16
x$l_icsz_height_inc	20

Table 8–4 Members of the Icon Size Data Structure (VAX Binding)

Member Name	Contents
X$L_ICSZ_MIN_WIDTH, X$L_ICSZ_MIN_HEIGHT	The minimum size for an icon window.
X$L_ICSZ_MAX_WIDTH, X$L_ICSZ_MAX_HEIGHT	The maximum size for an icon window. When an increment is added to the base width and height, the new base width and height cannot exceed these maximum values.
X$L_ICSZ_WIDTH_INC, X$L_ICSZ_HEIGHT_INC	Specifies increments that can be added to a base width and height. Any multiple of the increments can be used as long as the total width and height do not exceed the maximum values set in X$L_ICSZ_MAX_WIDTH and X$L_ICSZ_MAX_HEIGHT. If zero is specified for an increment, then no increments should be used.

The data structure for the MIT C binding is shown in Figure 8–4, and information about members in the data structure is described in Table 8–5.

Figure 8–4 Icon Size Data Structure (MIT C Binding)

```
typedef struct {
        int min_width,min_height;
        int max_width,min_height;
        int width_inc,height_inc;
}XIconSize;
```

Table 8–5 Members of the Icon Size Data Structure (MIT C Binding)

Member Name	Contents
min_width, min_height	The minimum size for an icon window.
max_width, max_height	The maximum size for an icon window. When an increment is added to the base width and height, the new base width and height cannot exceed these maximum values.
width_inc, height_inc	Specifies increments that can be added to a base width and height. Any multiple of the increments can be used as long as the total width and height do not exceed the maximum values set in max_width and max_height. If zero is specified for an increment, then no increments should be used.

8.3 Window Manager Hints Data Structure

The window manager hints (WM hints) data structure allows you to recommend five window hints to the window manager. It is not guaranteed that the window manager will use the values you set. The use of these recommendations is dependent on an individual window manager program.

The data structure for the VAX binding is shown in Figure 8–5, and information about members in the data structure is described in Table 8–6.

Figure 8–5 WM Hints Data Structure (VAX Binding)

x$l_hint_flags	0
x$l_hint_input	4
x$l_hint_initial_state	8
x$l_hint_icon_pixmap	12
x$l_hint_icon_window	16
x$l_hint_icon_x	20
x$l_hint_icon_y	24

(continued on next page)

Figure 8–5 (Cont.) WM Hints Data Structure (VAX Binding)

x$l_hint_icon_mask	28
x$l_hint_window_group	32

Table 8–6 Members of the WM Hints Data Structure (VAX Binding)

<table>
<tr><th>Member Name</th><th>Contents</th></tr>
<tr><td>X$L_HINT_FLAGS</td><td>The members of the structure that are defined.</td></tr>
<tr><td>X$L_HINT_INPUT</td><td>Indicates whether or not the application relies on the window manager to get keyboard input.</td></tr>
<tr><td>X$L_HINT_INITIAL_STATE</td><td>Defines how the window should appear in its initial configuration. Possible initial states are as follows:
<table>
<tr><th>Constant</th><th>Description</th></tr>
<tr><td>x$c_dont_care_state</td><td>Client is not interested in the initial state</td></tr>
<tr><td>x$c_normal_state</td><td>Initial state used most often</td></tr>
<tr><td>x$c_zoom_state</td><td>Window starts zoomed</td></tr>
<tr><td>x$c_iconic_state</td><td>Window starts as an icon</td></tr>
<tr><td>x$c_inactive_state</td><td>Window is seldom used</td></tr>
</table></td></tr>
<tr><td>X$L_HINT_ICON_PIXMAP</td><td>Identifies the pixmap used to create the window icon.</td></tr>
<tr><td>X$L_HINT_ICON_WINDOW</td><td>The window to be used as an icon.</td></tr>
<tr><td>X$L_HINT_ICON_X</td><td>The initial x-coordinate of the icon position.</td></tr>
<tr><td>X$L_HINT_ICON_Y</td><td>The initial y-coordinate of the icon postion.</td></tr>
<tr><td>X$L_HINT_ICON_MASK</td><td>The pixmap that filters which pixels of the icon pixmap should be drawn.</td></tr>
<tr><td>X$L_HINT_WINDOW_GROUP</td><td>The window identifier of the main application window when a group of windows changes state as a unit.</td></tr>
</table>

The data structure for the VAX binding is shown in Figure 8–6, and information about members in the data structure is described in Table 8–7.

Figure 8–6 WM Hints Data Structure (MIT C Binding)

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

Table 8–7 Members of the WM Hints Data Structure (MIT C Binding)

<table>
<tr><th>Member Name</th><th>Contents</th></tr>
<tr><td>flags</td><td>The members of the structure that are defined.</td></tr>
<tr><td>input</td><td>Indicates whether or not the application relies on the window manager to get keyboard input.</td></tr>
<tr><td>initial_state</td><td>Defines how the window should appear in its initial configuration. Possible initial states are as follows:
<table>
<tr><th>Constant Name</th><th>Description</th></tr>
<tr><td>DontCareState</td><td>Client is not interested in the initial state</td></tr>
<tr><td>NormalState</td><td>Initial state used most often</td></tr>
<tr><td>ZoomState</td><td>Window starts zoomed</td></tr>
<tr><td>IconicState</td><td>Window starts as an icon</td></tr>
<tr><td>InactiveState</td><td>Window is seldom used</td></tr>
</table>
</td></tr>
<tr><td>icon_pixmap</td><td>Identifies the pixmap used to create the window icon.</td></tr>
<tr><td>icon_window</td><td>The window to be used as an icon.</td></tr>
<tr><td>icon_x</td><td>The initial x-coordinate of the icon position.</td></tr>
<tr><td>icon_y</td><td>The initial y-coordinate of the icon position.</td></tr>
<tr><td>icon_mask</td><td>The pixmap that filters which pixels of the icon pixmap should be drawn.</td></tr>
<tr><td>window_group</td><td>The window identifier of the main application window when a group of windows changes state as a unit.</td></tr>
</table>

8.4 Property Routines

The following pages describe the Xlib property routines.

CHANGE PROPERTY

Changes a property of a specified window.

VAX FORMAT

X$CHANGE_PROPERTY
(display, window_id, property_id, type_id, format, change_mode, prop_data, num_elements)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
property_id	identifier	uns longword	read	reference
type_id	identifier	uns longword	read	reference
format	longword	longword	read	reference
change_mode	longword	uns longword	read	reference
prop_data	array	byte	read	reference
num_elements	longword	longword	read	reference

MIT C FORMAT

XChangeProperty
(display, window_id, property_id, type_id, format, change_mode, prop_data, num_elements)

argument information

```
XChangeProperty(display, window_id, property_id, type_id, format,
change_mode, prop_data, num_elements)
      Display *display;
      Window window_id;
      Atom property_id, type_id;
      int format;
      int change_mode;
      unsigned char *prop_data;
      int num_elements;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window.

property_id

The identifier of the atom that specifies the property to be changed.

type_id

The identifier of the atom that specifies the type of property to be changed.

format

The format of the property, which can be an 8-bit, 16-bit, or 32-bit format. If the format is 16-bit or 32-bit, the pointer to the property data, specified in **prop_data**, must point to character data.

change_mode

The type of property change to be completed by the routine. The predefined values for **change_mode** are as follows:

VAX	C	Description
X$C_PROP_MODE_ REPLACE	PropModeReplace	Deletes the previous property value and replaces it with the new property value.
X$C_PROP_MODE_ PREPEND	PropModePrepend	Places the new property data at the beginning of the existing data, as long as the property type and format specified in **type_id** and **format** match the type and format of the specified property. If property is undefined, it is treated as defined with the correct type and format with zero-length data.
X$C_PROP_MODE_ APPEND	PropModeAppend	Places the new property data at the end of the existing data, as long as the property type and format specified in **type_id** and **format** match the type and format of the specified property. If property is undefined, it is treated as defined with the correct type and format with zero-length data.
None	No value specified	Assumes that the values for the **type** and **format** specified match the values for the specified property, and that the new value is zero.

Other values specified in this argument are not valid.

prop_data

Pointer to the new property data. If the **format** argument specifies a 16-bit or 32-bit format, the data pointed to must be character data.

num_elements

The number of properties in the specified format for the window.

DESCRIPTION

CHANGE PROPERTY changes the data for a specified property for the specified window. The identifier of the window was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW. When the property is changed, a Property Notify event is generated.

You specify the property to be changed by specifying the identifier of the atom associated with the property. The atom identifier was originally returned by INTERN ATOM. The property remains associated with its atom identifier even after the user who defined it disconnects from the server. The property is disassociated from the identifier only after the application program performs these actions:

- Deletes the property with the DELETE PROPERTY routine
- Destroys the window associated with the property
- Closes the last connection to the server

The property type, specified in **type_id**, is not interpreted by this routine. CHANGE PROPERTY passes the property type identifier to the application program for later use with GET WINDOW PROPERTY.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

CONVERT SELECTION

Requests that the specified selection be converted to the specified target type.

VAX FORMAT

X$CONVERT_SELECTION
(display, selection_id, target_id, property_id, requestor_id, time)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
selection_id	identifier	uns longword	read	reference
target_id	identifier	uns longword	read	reference
property_id	identifier	uns longword	read	reference
requestor_id	identifier	uns longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

XConvertSelection
(display, selection_id, target_id, property_id, requestor_id, time)

argument information

```
XConvertSelection(display, selection_id, target_id, property_id,
requestor_id, time)
      Display *display;
      Atom selection_id, target_id;
      Atom property_id;
      Window requestor_id;
      Time time;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

selection_id
The identifier of the selection atom.

target_id
The identifier of the target atom.

property_id
The identifier of the property. It can be passed as None.

requestor_id

The identifier of the window that receives selection notification, if the selection has no owner.

time

The time when the conversion should take place. Either a timestamp, in milliseconds, or the predefined value Current Time can be specified.

DESCRIPTION

CONVERT SELECTION requests that the specified selection be converted to the specified target type. If the specified selection is owned by a window, the server sends a selection request event to the owner. If no owner for the specified selection exists, the server generates a Selection Notify event to the requestor with property None.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

DELETE CONTEXT

Deletes an entry for a specified window and context type.

VAX FORMAT

status_return = **X$DELETE_CONTEXT**
(display, window_id, context_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
context_id	identifier	uns longword	read	reference

MIT C FORMAT

int = **XDeleteContext**
(display, window_id, context_id)

argument information

```
int XDeleteContext(display, window_id, context)
      Display *display;
      Window window_id;
      XContext context;
```

RETURNS

status_return

Error code returned by the function. DELETE CONTEXT returns a nonzero error code if an error occurs, and zero if an error does not occur.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window with which the data is associated.

context_id

The identifier of the context type corresponding to the data structure.

DESCRIPTION

DELETE CONTEXT deletes an entry for a specified window and context type from the context manager data structure. The identifier of the window was originally returned by CREATE SIMPLE WINDOW.

DELETE CONTEXT returns a nonzero error code if an error occurs, and zero if an error does not occur.

DELETE PROPERTY

Deletes the association between a property and a specified window.

VAX FORMAT

X$DELETE_PROPERTY
(display, window_id, property_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
property_id	identifier	uns longword	read	reference

MIT C FORMAT

XDeleteProperty
(display, window_id, property_id)

argument information

```
XDeleteProperty(display, window_id, property_id)
      Display *display;
      Window window_id;
      Atom property_id;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The identifier of the window with the property to be deleted.

property_id
The identifier of the atom that specifies the property to be deleted.

DESCRIPTION

DELETE PROPERTY deletes the association between a property and a specified window. The atom identifier used to reference the property can no longer be used. If the property to be deleted was defined on the specified window, a Property Notify event is generated.

DELETE PROPERTY deletes the property only if the property was defined on the specified window. Otherwise, a Bad Atom error is returned.

The identifier of the window was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW. The identifier of the atom was originally returned by INTERN ATOM.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

FETCH BUFFER

Returns the data stored in the specified cut buffer.

VAX FORMAT

status_return = **X$FETCH_BUFFER** *(display, num_bytes_return, num_buffer, buff_addr_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
num_bytes_return	longword	longword	write	reference
num_buffer	longword	longword	read	reference
buff_addr_return	longword	longword	write	value

MIT C FORMAT

char = **XFetchBuffer** *(display, num_bytes_return, num_buffer)*

argument information

```
char *XFetchBuffer(display, num_bytes_return, num_buffer)
      Display *display;
      int *num_bytes_return;
      int num_buffer;
```

RETURNS

status_return (VAX only)

A return value that specifies whether the routine completed successfully.

char (MIT C only)

A pointer to the data stored in the specified buffer.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

num_bytes_return

The number of bytes in the string in the buffer. If there is no data in the buffer, the value 0 is returned in the **num_bytes_return** argument.

num_buffer

The buffer in which the byte string is stored. Valid entries are 0 through 7.

buff_addr_return (VAX only)

The address of the stream of bytes.

DESCRIPTION

FETCH BUFFER returns a pointer to the string of bytes stored in the specified cut buffer and a value indicating the number of bytes in the string. If the buffer contains data, FETCH BUFFER returns the number of bytes in the **num_bytes_return** argument; otherwise, it returns a null value and sets **num_bytes_return** to zero.

X ERRORS

VAX	C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

FETCH BYTES

Returns the data stored in cut buffer zero.

VAX FORMAT

buff_addr_return = **X$FETCH_BYTES** *(display, num_bytes_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
buff_addr_return	longword	longword	write	value
display	identifier	uns longword	read	reference
num_bytes_return	longword	longword	write	reference

MIT C FORMAT

char = **XFetchBytes** *(display, num_bytes_return)*

argument information

```
char *XFetchBytes(display, num_bytes_return)
      Display *display;
      int *num_bytes_return;
```

RETURNS

buff_addr_return (VAX only)

The address of a stream of bytes stored in cut buffer zero.

char (MIT C only)

A pointer to the data stored in cut buffer zero.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

num_bytes_return

The number of bytes in the string in the buffer. FETCH BYTES returns the number of bytes in this argument. If there is no data in the buffer, the value 0 is returned in this argument.

DESCRIPTION

FETCH BYTES returns the number of bytes in the **num_bytes_return** argument, if the buffer contains data. Otherwise, the function returns NULL and sets **num_bytes_return** to 0.

The appropriate amount of storage is allocated and the pointer is returned; the client must free this storage with the FREE routine when finished with it. Note that the cut buffer's contents need not be text, so it may contain embedded null bytes and cannot end with a null byte.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

FETCH NAME

Provides the name for the specified window (if one exists in the WM_NAME property).

VAX FORMAT

status_return = **X$FETCH_NAME**
(display, window_id, window_name_return
[,name_len_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
window_name_return	char string	char string	write	descriptor
name_len_return	word	uns word	write	reference

MIT C FORMAT

status_return = **XFetchName**
(display, window_id, window_name_return)

argument information

```
int XFetchName(display, window_id, window_name_return)
      Display *display;
      Window window_id;
      char **window_name_return;
```

RETURNS

status_return

Return value that specifies whether the routine completed successfully. FETCH NAME returns a nonzero error code if an error occurs, and zero in the MIT C binding or X$C_SUCCESS in the VAX binding if an error does not occur or if no name exists for the window.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to determine the name for.

window_name_return

The name of the window returned by the routine. If a name has not been assigned to the window, a null value is returned.

name_len_return (VAX only)

The length of **window_name_return**. This argument is optional.

DESCRIPTION

FETCH NAME returns the name of the specified window. The window name is stored in the window manager name property. The identifier of the window was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW.

If the window does not have a name, the routine returns a null value to the **window_name_return** argument and zero for failure (in the MIT C binding) to **status_return**.

After a program has finished using the name, the program must free the string storing the window name.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

FIND CONTEXT

Obtains the data associated with a specified window and context type.

VAX FORMAT

status_return = **X$FIND_CONTEXT**
(display, window_id, context_id [,window_data_return] [,buff_len] [,buff_ptr_return] [,len_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
context_id	identifier	uns longword	read	reference
window_data_return	longword	uns longword	read	reference
buff_len	longword	longword	read	reference
buff_ptr_return	array	byte	write	reference
len_return	longword	uns longword	write	reference

MIT C FORMAT

status_return = **XFindContext**
(display, window_id, context_id, window_data_return)

argument information

```
int XFindContext(display, window_id, context_id,
                 window_data_return)
      Display *display;
      Window window_id;
      XContext context;
      caddr_t *window_data_return;
```

RETURNS

status_return
Error code returned by the function. FIND CONTEXT returns a nonzero error code if an error occurs, and zero if an error does not occur.

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The identifier of the window with which the data is associated.

context_id

The identifier of the context type to which the data corresponds.

window_data_return

The data associated with the specified window and type. FIND CONTEXT returns the associated data to this argument. This argument is optional in the VMS binding.

buff_len (VAX only)

The length of the supplied buffer. This argument is optional.

buff_ptr_return (VAX only)

The buffer into which data is written. This argument is optional.

len_return (VAX only)

The length of the data written into the buffer. This argument is optional.

DESCRIPTION

FIND CONTEXT obtains the data associated with a specified window and context type. The identifier of the window was originally returned by CREATE SIMPLE WINDOW. The identifier of the context type was originally returned by UNIQUE CONTEXT.

FIND CONTEXT returns the associated data to the **window_data_return** argument. FIND CONTEXT also returns a nonzero error code to **status_return** if an error occurs, and zero if no errors occur.

GET ATOM NAME

Returns the atom name associated with the specified atom identifier.

VAX FORMAT

status_return = **X$GET_ATOM_NAME** *(display, atom_id, name_return [,name_len_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
atom_id	identifier	uns longword	read	reference
name_return	char string	char string	write	descriptor
name_len_return	word	uns word	write	reference

MIT C FORMAT

name_return = **XGetAtomName** *(display, atom_id)*

argument information

```
char *XGetAtomName(display, atom_id)
      Display *display;
      Atom atom_id;
```

RETURNS

status_return (VAX only)

Return value that specifies whether the routine completed successfully. The routine returns one of the following condition values:

Value	Description
SS$_NORMAL	Routine completed successfully.
X$_ERRORREPLY	Error received from the server.

name_return (C only)

The name associated with the atom specified in **atom_id**. The **name_return** argument is a pointer to a null-terminated character string.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

atom_id

The identifier of the atom to return the name for. The atom identifier was originally returned by INTERN ATOM.

name_return (VAX only)

The name of the atom associated with the identifier specified in **atom_id**. The atom name is returned by the routine. The **name_return** argument is the address of a character string descriptor that points to the string.

name_len_return (VAX only)

The length of the atom name. The length is returned by the routine. This argument is optional.

DESCRIPTION

GET ATOM NAME returns the atom name associated with the specified atom identifier. The atom name remains associated with the identifier until the last user disconnects from the server.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.

GET CLASS HINT

Obtains the class of a specified window.

VAX FORMAT

status_return = **X$GET_CLASS_HINT** *(display, window_id, class_hints_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
class_hints_return	record	x$class_hint	write	reference

MIT C FORMAT

status_return = **XGetClassHint** *(display, window_id, class_hints_return)*

argument information

```
Status XGetClassHint(display, window_id, class_hints_return)
      Display *display;
      Window window_id;
      XClassHint *hints_return;
```

RETURNS

status_return

Return value that specifies whether the routine completed successfully. GET CLASS HINTS returns zero if no class has been defined on the window, and returns nonzero otherwise.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window for which you want to obtain the class.

class_hints_return

The class hints data structure, which specifies the class of the window.

DESCRIPTION

GET CLASS HINT obtains the class of a specified window. This information is stored in the predefined property WM_CLASS. In addition, this routine references a class hints data structure, which contains an application name and an application class. Note that this name may differ from the name set as WM_NAME.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET ICON NAME

Obtains the name to be displayed for a window in its icon.

VAX FORMAT

status_return = **X$GET_ICON_NAME** *(display, window_id, icon_name_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
icon_name_return	char string	char string	write	descriptor

MIT C FORMAT

status_return = **XGetIconName** *(display, window_id, icon_name_return)*

argument information

```
int XGetIconName(display, window_id, icon_name_return)
      Display *display;
      Window window_id;
      char **icon_name_return;
```

RETURNS

status_return

Return value that specifies whether the routine completed successfully. GET ICON NAME returns zero if a window manager has not set an icon name, and a nonzero value if it has.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to get the icon name for.

icon_name_return

The name to be displayed when the icon representation of the window is displayed. The name is stored in WM_ICON_NAME and is returned by the routine. If no icon name has been specified, a null value is returned.

DESCRIPTION

GET ICON NAME returns the name to be displayed for a window in its icon. The icon name is stored in the predefined property WM_ICON_NAME.

When an application program no longer needs to use the icon name, the program should free the icon name string.

To specify the name for the icon, use SET ICON NAME.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET ICON SIZES

Obtains the recommended icon sizes from a window manager.

VAX FORMAT

status_return = **X$GET_ICON_SIZES**
(display, window_id [,size_list_return] [,count_return] [,list_size] [,list_buff_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
size_list_return	address	uns longword	write	reference
count_return	longword	longword	write	reference
list_size	longword	longword	read	reference
list_buff_return	array	uns longword	write	reference

MIT C FORMAT

status_return = **XGetIconSizes**
(display, window_id, size_list_return, count_return)

argument information

```
Status XGetIconSizes(display, window_id, size_list_return,
                     count_return)
      Display *display;
      Window window_id;
      XIconSize **size_list_return;
      int *count_return;
```

RETURNS

status_return
Return value that states whether or not the routine completed successfully. GET ICON SIZES returns zero if no icon size property has been defined for the window, and returns a nonzero value otherwise.

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The identifier of the window to obtain the icon sizes for.

size_list_return

The virtual address of a pointer to an array of icon size data, returned by the routine and residing in space reserved by Xlib. The recommended icon size is defined by minimum, maximum, and incremental width and height specifications. If the incremental width and height specifications are zero, then a single size is recommended. If the incremental specifications are nonzero, then the minimum size plus an increment up to the maximum size is permitted.

For more information on the icon size data structure, see Section 8.2.

VAX only

This argument is optional.

count_return

The number of items in **size_list_return**.

VAX only

This argument is optional.

list_size (VAX only)

The size of the buffer in **list_buff_return**. This argument is optional.

list_buff_return (VAX only)

A pointer to a data buffer, residing in space you have reserved, where each entry is one icon size data element. The size of the buffer is specified by **list_size**. The icon size data is returned by the routine. This argument is optional.

DESCRIPTION

GET ICON SIZES obtains the sizes for the icon window representation of the regular window. Icon sizes are usually set by a window manager program and are stored in the WM_ICON_SIZE property. Programs can use the sizes to create an icon window that is compatible with the window manager.

The icon sizes include the following values:

- The minimum height and width
- The maximum height and width
- An increment to be added to the minimum height and width

To specify arguments that describe the icon size data returned by the routine, use **size_list_return** to access data owned by Xlib, or **list_size** and **list_buff_return** to obtain a private copy of the data. To free the storage returned by this routine, use FREE.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET NORMAL HINTS

Obtains recommended values for the size and location of a regular window.

VAX FORMAT

status_return = **X$GET_NORMAL_HINTS** *(display, window_id, hints_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
hints_return	record	x$size_hints	write	reference

MIT C FORMAT

status_return = **XGetNormalHints** *(display, window_id, hints_return)*

argument information

```
int XGetNormalHints(display, window_id, hints_return)
      Display *display;
      Window window_id;
      XSizeHints *hints_return;
```

RETURNS

status_return

Return value that specifies whether the routine completed successfully. GET NORMAL HINTS returns zero if no normal hints property has been defined on the window, and returns a nonzero value otherwise.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to obtain size and location values for.

hints_return

The size hints data structure containing the recommended values for the window.

For more information on the size hints data structure, see Section 8.1.

DESCRIPTION

GET NORMAL HINTS obtains recommended values for a regular (as opposed to icon or zoom) window size and location. This information is stored in the WM_NORMAL_HINTS predefined property. A window manager program can use this information to size and locate the window according to your specifications. However, a window manager may not use this information.

The following values are specified in the size hints data structure:

- Which values have been specified (the flags member)
- The x- and y-coordinates of the initial window location
- The desired width and height of the regular window
- The minimum width and height of the regular window
- The maximum width and height of the regular window
- An increment to be added to the minimum width and height
- The preferred aspect ratios

Use SET NORMAL HINTS to specify recommended values.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET SELECTION OWNER

Returns the owner of the specified window selection.

VAX FORMAT

owner_id_return = **X$GET_SELECTION_OWNER** *(display, selection_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
owner_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
selection_id	identifier	uns longword	read	reference

MIT C FORMAT

owner_id_return = **XGetSelectionOwner** *(display, selection_id)*

argument information

```
Window XGetSelectionOwner(display, selection_id)
      Display *display;
      Atom selection_id;
```

RETURNS

owner_id_return

The identifier of the window that owns the selection. If no selection is specified, the null value is returned.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

selection_id

The identifier of the selection.

DESCRIPTION

GET SELECTION OWNER returns the identifier of the window that currently owns the specified selection. If no window owns the selection, None is returned.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.

GET SIZE HINTS

Obtains window size hints for any property, using the WM_SIZE_HINTS format.

VAX FORMAT

status_return = **X$GET_SIZE_HINTS** *(display, window_id, hints_return, property_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
hints_return	record	x$size_hints	write	reference
property_id	identifier	uns longword	read	reference

MIT C FORMAT

status_return = **XGetSizeHints** *(display, window_id, hints_return, property_id)*

argument information

```
Status XGetSizeHints(display, window_id, hints_return,
                     property_id)
      Display *display;
      Window window_id;
      XSizeHints *hints_return;
      Atom property_id;
```

RETURNS

status_return

Return value that specifies whether the routine completed successfully. GET SIZE HINTS returns zero if no size hint property has been defined on the window, and returns nonzero otherwise.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to obtain size hints for.

hints_return

The size hints data structure containing the recommended values for the window.

For more information on the size hints data structure, see Section 8.1.

property_id

The identifier of the atom that specifies the size property. The size property contains the window size hints.

DESCRIPTION

GET SIZE HINTS obtains recommended values for a window's size and location. This information is stored in the predefined property format WM_SIZE_HINTS. This format is used with WM_NORMAL_HINTS and WM_ZOOM_HINTS to recommend sizing and location information for windows in their regular and zoom states. A window manager program can use this information to size and locate the window according to your specifications. However, a window manager may not use this information.

The following values are specified in the size hints data structure:

- Which values are specified (the flags member)
- The x- and y-coordinates of the initial window location
- The desired width and height of the window
- The minimum width and height of the window
- The maximum width and height of the window
- An increment to be added to the minimum width and height
- The preferred aspect ratios

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET TRANSIENT FOR HINT

Obtains the WM_TRANSIENT_FOR property of a specified window.

VAX FORMAT

status_return = **X$GET_TRANSIENT_FOR_HINT** *(display, window_id, prop_window_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
prop_window_return	identifier	uns longword	read	reference

MIT C FORMAT

status_return = **XGetTransientForHint** *(display, window_id, class_hints_return)*

argument information

```
Status XGetTransientForHint(display, window_id,
                             prop_window_return)
      Display *display;
      Window window_id;
      Window prop_window_return;
```

RETURNS

status_return
Return value that specifies whether the routine completed successfully. GET TRANSIENT FOR HINT returns zero if no transient-for property has been defined for the window, and returns nonzero otherwise.

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The window for which you want to obtain the transient-for property.

prop_window_return
The transient-for property of the window specified in the **window_id** argument.

DESCRIPTION

GET TRANSIENT FOR HINT obtains the transient-for property of a specified window. A transient window is a temporary window that acts on behalf of another window (for example, a popup dialog box that partially obscures the main application window). Setting the transient-for property on the popup window allows the window manager to automatically iconify the popup window when it iconifies the main application window.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET WINDOW PROPERTY

Returns type, format, and description information for one property associated with a window.

VAX FORMAT

status_return = **X$GET_WINDOW_PROPERTY** *(display, window_id, property_id, long_offset, long_len, delete, requested_type, actual_type_return, actual_format_return, num_items_return, bytes_after_return, [,property_data_return] [,property_data_len] [,property_data_buff_return] [,num_elements_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
property_id	identifier	uns longword	read	reference
long_offset	longword	longword	read	reference
long_len	longword	longword	read	reference
delete	longword	uns longword	read	reference
requested_type	identifier	uns longword	read	reference
actual_type_return	identifier	uns longword	write	reference
actual_format_return	longword	longword	write	reference
num_items_return	longword	longword	write	reference
bytes_after_return	longword	longword	write	reference
property_data_return	address	uns longword	write	reference
property_data_len	longword	longword	read	reference
property_data_buff_return	array	uns longword	write	reference
num_elements_return	longword	longword	write	reference

MIT C FORMAT

status_return = **XGetWindowProperty** *(display, window_id, property_id, long_offset, long_len, delete, requested_type, actual_type_return, actual_format_return, num_items_return, bytes_after_return, property_data_return)*

argument information

```
int XGetWindowProperty(display, window_id, property_id,
                      long_offset, long_len, delete,
                      requested_type, actual_type_return,
                      actual_format_return, num_items_return,
                      bytes_after_return, property_data_return)
      Display *display;
      Window window_id;
      Atom property_id;
      long long_offset, long_len;
      Bool delete;
      Atom requested_type;
      Atom *actual_type_return;
      int *actual_format_return;
      unsigned long *num_items_return;
      long *bytes_after_return;
      unsigned char **property_data_return;
```

RETURNS

status_return

Return value that specifies whether the routine completed successfully.

C only

This argument returns zero if the routine completes successfully, and nonzero if it does not complete successfully.

VAX only

This argument returns zero if the routine completes successfully, and one of the following values if it does not complete successfully.

Value	Description
X$_ERRORREPLY	Error received from the server.
X$_TRUNCATED	Buffer not big enough, therefore the results are truncated.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to obtain atom and property information for.

property_id

The identifier of the property to obtain information for. A property is identified by its associated atom identifier.

long_offset

The offset, in 32-bit units, where data will be retrieved.

long_len

The length, in 32-bit units, of the data to be retrieved.

delete

Whether to delete a property after it is returned. When **delete** is true and a property is returned, it is deleted from the window. If a property is deleted, a Property Notify event is generated for the window. When **delete** is false and a property is returned, it remains associated with the window.

requested_type

The identifier of the atom that specifies the type of property. If a particular type is specified, the property is returned in **actual_type_return** only if the type (as known to the server) matches the specified property type. If the predefined value of any property type is specified, the type is returned in **actual_type_return**, regardless of its type, and the format is returned in **actual_format_return**.

actual_type_return

The identifier of the atom, returned by the routine, that describes the actual type of property, as known to the server. If the property does not exist, the value zero is returned.

actual_format_return

The data type of the property returned by the routine, as known to the server. The data type can be 8 bit, 16 bit, or 32 bit. If the property does not exist, zero is returned.

property_data_return

The virtual address of a pointer to an array of property data returned by the routine and residing in space reserved by Xlib.

This argument is optional.

num_items_return

The number of 8-, 16-, or 32-bit units that were returned to **property_data_return**. If the property does not exist, or if the property type does not match the type specified in **requested_type**, no value is returned.

bytes_after_return

The number of bytes remaining to be read in the property if a partial read operation was performed. This value is returned by the routine. If the property does not exist, zero is returned.

property_data_len (VAX only)

The length of the data buffer specified in **property_data_buff_return**.

This argument is optional.

property_data_buff_return (VAX only)

A pointer to a data buffer, residing in space you have reserved, where each entry is one property element. The length of the buffer is specified by **property_data_len**. The property data is returned by the routine.

This argument is optional.

DESCRIPTION

GET WINDOW PROPERTY provides the type, format, and data for a property associated with the specified window. The identifier of the window was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW.

These five values are defined with GET WINDOW PROPERTY:

Value	Description
N	Actual length of the stored property in bytes (even if the format is 16 or 32).
I	4 times **long_offset**. The returned value starts at byte index I in the property (indexing from 0).
T	Actual length N minus byte index I.
L	MINIMUM(T, 4 times long_len). The returned value length, in bytes. If the value in **long_offset** results in a negative for L, an error occurs.
A = N— (I + L)	The value of **bytes_after_return**.

The returned value starts at byte index I in the property (indexing from 0) and its length in bytes is L. A Bad Value error results if **long_offset** is given such that L is negative. The value of **bytes_after_return** is A, giving the number of trailing unread bytes in the stored property.

To specify arguments that describe the property data returned by the routine, use **property_data_return** to access data owned by Xlib, or **property_data_len** and **property_data_buff_return** to obtain a private copy of the data. To free the storage returned by this routine, use FREE.

GET WINDOW PROPERTY sets the return arguments according to the following:

- If the specified property does not exist for the specified window, GET WINDOW PROPERTY returns None to **actual_type_return** and the value 0 to **actual_format_return** and **bytes_after_return**. The **num_items_return** argument is empty, and **delete** is ignored.
- If the specified property exists, but its type does not match the specified type, GET WINDOW PROPERTY returns the actual property type to **actual_type_return**, the actual property format (never zero) to **actual_format_return**, and the property length in bytes (even if **actual_format_return** is 16 or 32) to **bytes_after_return**. The **num_items_return** argument is empty, and **delete** is ignored.
- If the specified property exists and you assign AnyPropertyType to **req_type** or the specified property type matches the actual property type, GET WINDOW PROPERTY returns the actual property type to **actual_type_return** and the actual property format (never zero) to **actual_format_return**. It also returns a value to **bytes_after_return** and **num_items_return**, by defining the following values:

```
N=actual length of the stored property in bytes
       (even if the format is 16 or 32)
I=4*long_offset
T=N-I
L=MINIMUM(T,4*long_len)
A=N-(I+L)
```

The returned value starts at byte index I in the property (indexing from zero), and its length in bytes is L. A Bad Value error is returned if the value for **long_offset** causes L to be negative. The value of **bytes_after_return** is A, giving the number of trailing unread bytes in the stored property. If **delete** is true and **bytes_after_return** is zero, the function deletes the property from the window and generates a Property Notify event on the window.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET WM HINTS

Obtains the window manager hints contained in the window manager hints property.

VAX FORMAT

X$GET_WM_HINTS
(display, window_id, wmhints_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
wmhints_return	record	x$wm_hints	write	reference

MIT C FORMAT

wmhints_return = **XGetWMHints**
(display, window_id)

argument information

```
XWMHints *XGetWMHints(display, window_id)
      Display *display;
      Window window_id;
```

RETURNS

C only—wmhints_return

The window manager hints data structure returned by the routine. If no data structure has been set for the window, this argument returns a null value.

For more information on the window manager hints data structure, see Section 8.3.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to obtain window manager hints for.

wmhints_return (VAX only)

The window manager hints data structure returned by the routine.

For more information on the window manager hints data structure, see Section 8.3.

DESCRIPTION

GET WM HINTS obtains the window manager hints in the window manager hints property. The following window manager hints are returned:

- Whether the program accepts input
- How a program is started (as a regular window, zoomed, or as an icon)
- A pixmap used for the icon representation
- A window identifier of a window used as the icon
- The initial position of the icon, relative to the root window

Use the SET WM HINTS routine to set the window manager hints in the window manager hints property.

When finished with this function, an application must free the space used for this structure by calling FREE.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET ZOOM HINTS

Obtains recommended values for the size and location of a window in its zoomed state.

VAX FORMAT

status_return = **X$GET_ZOOM_HINTS** *(display, window_id, zhints_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
zhints_return	record	x$size_hints	write	reference

MIT C FORMAT

status_return = **XGetZoomHints** *(display, window_id, zhints_return)*

argument information

```
Status XGetZoomHints(display, window_id, zhints_return)
      Display *display;
      Window window_id;
      XSizeHints *zhints_return;
```

RETURNS

status_return

Return value that states whether or not the routine completed successfully. GET ZOOM HINTS returns zero if no zoom hints property has been defined for the window, and returns a nonzero value otherwise.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to obtain zoom size and location values for.

zhints_return

The size hints data structure containing the recommended values for the zoom window.

For more information on the size hints data structure, see Section 8.1.

DESCRIPTION

GET ZOOM HINTS obtains recommended values for a zoom window's size and location. Zoom hints are stored in the WM ZOOM HINTS predefined property. A window manager program can use this information to size and locate a zoom window according to your specifications. However, it is not guaranteed that a window manager will use this information.

The following values are specified in the size hints data structure:

- Which values have been specified (the flags member)
- The x- and y-coordinates of the initial window location
- The desired width and height of the window
- The minimum width and height of the window
- The maximum width and height of the window
- An increment to be added to the minimum width and height
- The mimimum and maximum aspect ratios

Use SET ZOOM HINTS to specify recommended values.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

INTERN ATOM

Returns the atom identifier for the specified atom name.

VAX FORMAT

atom_id_return = **X$INTERN_ATOM** *(display, atom_name, only_if_exists)*

argument information

Argument	Usage	Data Type	Access	Mechanism
atom_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
atom_name	char string	char string	read	descriptor
only_if_exists	Boolean	longword	read	reference

MIT C FORMAT

atom_id_return = **XInternAtom** *(display, atom_name, only_if_exists)*

argument information

```
Atom XInternAtom(display, atom_name, only_if_exists)
      Display *display;
      char *atom_name;
      Bool only_if_exists;
```

RETURNS

atom_id_return

The identifier for the specified atom name. When **only_if_exists** is true, None is returned in the **atom_id** argument if there is no associated atom identifier for the specified atom name.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

atom_name

The name of the atom associated with the atom identifier. The name specified in this argument must exactly match the name being maintained by the server. When supplying the name of the atom, you must use the correct uppercase and lowercase characters for each character in the string. For example, if you want to know the identifier of the atom name *String*, you must specify *String*. The names *string* or *STRING* would not return the correct identifier.

C only

The **atom_name** argument must be a null-terminated ASCII string.

only_if_exists

Boolean value that specifies whether to create an atom identifier for an atom name without an existing identifier. When **only_if_exists** is true, then None is returned in **atom_id_return** when no atom identifier exists for the named atom. When **only_if_exists** is false, then a new atom identifier is created for the specified atom name and returned in **atom_id**.

DESCRIPTION

INTERN ATOM returns the identifier of the atom specified in **atom_name**. If no identifier exists for the atom, you can specify whether INTERN ATOM should create an identifier for the atom or specify that none exists.

The atom identifier is used in other routines to refer to the atom. Any atom identifier and its associated name remain defined until the last user disconnects from the server.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

LIST PROPERTIES

Returns a list of all properties associated with a window.

VAX FORMAT

status_return = **X$LIST_PROPERTIES** *(display, window_id, num_prop_return [,properties_return] [,properties_size] [,properties_buff_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
num_prop_return	longword	longword	write	reference
properties_return	address	uns longword	write	reference
properties_size	longword	longword	read	reference
properties_buff_return	array	longword	write	reference

MIT C FORMAT

atom_list = **XListProperties** *(display, window_id, num_prop_return)*

argument information

```
Atom *XListProperties(display, window_id, num_prop_return)
      Display *display;
      Window window_id;
      int *num_prop_return;
```

RETURNS

status_return (VAX only)

Return value that specifies whether the routine completed successfully.

atom_list (MIT C only)

A pointer to an array of atom identifiers. Each element is an atom for the specified window. The length of the array is returned by the routine in **num_prop_return**.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window for which the property list will be returned.

num_prop_return

A pointer, returned by the routine, to the number of properties associated with the specified window.

properties_return (VAX only)

The virtual address of a pointer to an array of property data, returned by the routine and residing in space reserved by Xlib.

properties_size (VAX only)

The size of the **properties_buff_return** buffer that will receive the property list.

properties_buff_return (VAX only)

A pointer to a data buffer, residing in space you have reserved, where each entry is one property element. The length of the buffer is specified by **properties_size**. The property data is returned by the routine.

DESCRIPTION

LIST PROPERTIES returns all the properties associated with the specified window. The identifier of the window was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW.

To specify arguments that describe the property data returned by the routine, use **properties_return** to access data owned by Xlib, or **properties_size** and **properties_buff_return** to obtain a private copy of the data. To free the storage returned by this routine, use FREE.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

ROTATE WINDOW PROPERTIES

Shifts the positions of the properties within the property array.

VAX FORMAT

X$ROTATE_WINDOW_PROPERTIES *(display, window_id, properties, num_prop, num_positions)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
properties	array	uns longword	read	reference
num_prop	longword	longword	read	reference
num_positions	longword	longword	read	reference

MIT C FORMAT

XRotateWindowProperties *(display, window_id, properties, num_prop, num_positions)*

argument information

```
XRotateWindowProperties(display, window_id,
                        properties, num_prop, num_positions)
      Display *display;
      Window window_id;
      Atom properties[];
      int num_prop;
      int num_positions;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window with the properties to be rotated.

properties

A pointer to an array of properties in which each element is an atom identifier associated with a property. The length of the array is specified by **num_prop**.

num_prop

The number of properties in the properties array. This value specifies the length of the array in **properties**.

num_positions

The number of positions or property names to rotate.

DESCRIPTION

ROTATE WINDOW PROPERTIES shifts the positions of the properties within the property array. If the property names in the properties array are viewed as being numbered starting with zero and if there are **num_prop** property names in the list, then the value associated with property name I becomes the value associated with property name (I + **num_positions**) MOD N, for all I from zero to N – 1. The effect is to rotate the states by **num_positions** places around the virtual ring of property names (right for positive **num_positions**, left for negative **num_positions**).

A Property Notify event for each property in the order listed is generated. If a Bad Atom or Bad Match error is generated, no properties are changed.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_MATCH	BadMatch	Possible causes are as follows: • In a graphics request, the root and depth of the graphics context do not match those of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input-only window lacks this attribute.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SAVE CONTEXT

Saves the data associated with a specified window and context type.

VAX FORMAT

status_return = **X$SAVE_CONTEXT**
(display, window_id, context_id, window_data, len)

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
context_id	identifier	uns longword	read	reference
window_data	longword	uns longword	read	reference
len	longword	longword	read	reference

MIT C FORMAT

int = **XSaveContext**
(display, window_id, context_id, window_data)

argument information

```
int XSaveContext(display, window_id, context_id, window_data)
      Display *display;
      Window window_id;
      XContext context_id;
      caddr_t window_data;
```

RETURNS

status_return
The error code returned by the function. SAVE CONTEXT returns a nonzero error code if an error occurs, and zero if an error does not occur.

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The identifier of the window with which the data is associated.

context_id
The identifier of the context type to which the data corresponds.

window_data

The data associated with the specified window and context type.

len (VAX only)

Length of the data associated with the specified window and context type.

DESCRIPTION

SAVE CONTEXT saves the data value associated with a specified window and context type. The identifier of the window was originally returned by CREATE SIMPLE WINDOW. The identifier of the context type was originally returned by UNIQUE CONTEXT.

If an entry with the specified window and type already exists, SAVE CONTEXT overrides the existing entry with the new entry. However, to save time and space, it is recommended that you first delete the existing entry with DELETE CONTEXT.

SAVE CONTEXT returns a nonzero error code if an error occurs, and zero if no errors occur.

SET CLASS HINT

Sets the class of a specified window.

VAX FORMAT

X$SET_CLASS_HINT
(display, window_id, class_hints_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
class_hints_return	record	x$class_hint	write	reference

MIT C FORMAT

XSetClassHint
(display, window_id, class_hints_return)

argument information

```
XSetClassHint(display, window_id, class_hints_return)
      Display *display;
      Window window_id;
      XClassHint *hints_return;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window for which you want to set the class.

class_hints_return

The class hints data structure, which specifies the class of the window.

DESCRIPTION

SET CLASS HINT sets the class of a specified window. This information is stored in the predefined property WM_CLASS. In addition, this routine references a class hints data structure, which contains an application name and an application class. Note that this name may differ from the name set as WM_NAME.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET COMMAND

Sets the command used to invoke an application program.

VAX FORMAT

X$SET_COMMAND
(display, window_id, command, num_args)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
command	char string	char string	read	descriptor
num_args	longword	longword	read	reference

MIT C FORMAT

XSetCommand
(display, window_id, command, num_args)

argument information

```
XSetCommand(display, window_id, command, num_args)
      Display *display;
      Window window_id;
      char **command;
      int num_args;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to set the command property for.

command

A pointer to the command and arguments used to start the application, which are specified as an array of pointers to null-terminated strings.

num_args

The number of arguments in the command.

DESCRIPTION

SET COMMAND sets the command used to invoke an application program, as well as the arguments used to invoke the application.

You can also set this property with SET STANDARD PROPERTIES. However, when you use SET STANDARD PROPERTIES, you must set five other properties as well. It is recommended that simple programs in which only the minimum properties are to be set use SET STANDARD PROPERTIES, and that applications that are going to set additional properties not use it.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET ICON NAME

Specifies a name to be displayed when the icon for a window is displayed.

VAX FORMAT

X$SET_ICON_NAME
(display, window_id, icon_name)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
icon_name	char string	char string	read	descriptor

MIT C FORMAT

XSetIconName
(display, window_id, icon_name)

argument information

```
XSetIconName(display, window_id, icon_name)
      Display *display;
      Window window_id;
      char *icon_name;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The identifier of the window for which the icon name is to be specified.

icon_name
The name to be displayed on the icon when the icon for the specified window is displayed.

DESCRIPTION

SET ICON NAME specifies a name to be displayed in a window's icon. This routine sets the predefined property WM_ICON_NAME.

To obtain the name once it is specified, use GET ICON NAME.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET ICON SIZES

Sets the recommended sizes for the icon for a window.

VAX FORMAT

X$SET_ICON_SIZES
(display, window_id, size_list, count)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
size_list	array	uns longword	read	reference
count	longword	longword	read	reference

MIT C FORMAT

XSetIconSizes
(display, window_id, size_list, count)

argument information

```
XSetIconSizes(display, window_id, size_list, count)
      Display *display;
      Window window_id;
      XIconSize *size_list;
      int count;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window that the icon sizes are being set for.

size_list

A pointer to icon size information. The recommended size is defined by minimum, maximum, and incremental width and height specifications. If the incremental width and height specifications are zero, then a single size is recommended. If the incremental width and height specifications are nonzero, then the minimum size plus an increment up to the maximum size is permitted.

For more information on the icon size data structure, see Section 8.2.

count

The number of items in the icon size data structure specified in **size_list**.

DESCRIPTION

SET ICON SIZES sets the WM ICON SIZE property, which contains the sizes for the icon window representation of the regular window. Usually a window manager program uses this routine to specify the acceptable icon window sizes for other programs.

Other programs can use GET ICON SIZES to read the values set by the window manager.

SET ICON SIZES sets the following icon size attributes:

- The minimum height and width
- The maximum height and width
- An increment to be added to the minimum height and width

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET NORMAL HINTS

Sets recommended values for the size and location of a regular window.

VAX FORMAT

X$SET_NORMAL_HINTS *(display, window_id, hints)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
hints	record	x$size_hints	read	reference

MIT C FORMAT

XSetNormalHints *(display, window_id, hints)*

argument information

```
void XSetNormalHints(display, window_id, hints)
      Display *display;
      Window window_id;
      XSizeHints *hints;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the regular window to set size and location values for.

hints

The size hints data structure containing the recommended values for the window.

For more information on the size hints data structure, see Section 8.1.

DESCRIPTION

SET NORMAL HINTS specifies recommended values for a regular (as opposed to icon or zoom) window's size and location. A window manager program can use this information to size and locate the window according to your specifications. This information is stored in the WM NORMAL HINTS predefined property. However, a window manager may not use this information.

The following values are specified in the size hints data structure:

- Which values have been specified (the flags member)
- The x- and y-coordinates of the initial window location
- The desired width and height of the regular window
- The minimum width and height of the regular window
- The maximum width and height of the regular window
- An increment to be added to the minimum width and height
- The aspect ratios preferred

It is important to set the flags member within the size hints data structure to inform the window manager which specific members have been set. If the flags member is not set, a window manager may disregard a call to SET NORMAL HINTS.

Use GET NORMAL HINTS to obtain recommended values that have already been specified.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET SELECTION OWNER

Sets the owner for the window selection.

VAX FORMAT

X$SET_SELECTION_OWNER
(display, selection_id, owner_window_id, time)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
selection_id	identifier	uns longword	read	reference
owner_window_id	identifier	uns longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

XSetSelectionOwner
(display, selection_id, owner_window_id, time)

argument information

```
XSetSelectionOwner(display, selection_id, owner_window_id, time)
       Display *display;
       Atom selection_id;
       Window owner_window_id;
       Time time;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

selection_id

The identifier of the selection.

owner_window_id

The identifier of the window that owns the selection. If there is no owner, this value can be specified as None.

time

The time when the selection should take place. Either a timestamp, in milliseconds, or the predefined Current Time value can be specified.

DESCRIPTION

SET SELECTION OWNER specifies the owner for the selected atom. If the new owner is not the same as the current owner of the selection, and the current owner is a window, then the current owner receives a Selection Clear event. If a window is specified and that window is later destroyed, the owner of the selection automatically reverts to None. The selection atom is not interpreted by the server.

All selections are global to the server.

The time specified must be no earlier than the last-change time of the specified selection and no later than the current time, or the selection is not made.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET SIZE HINTS

Specifies window size hints for any property.

VAX FORMAT

X$SET_SIZE_HINTS
(display, window_id, hints_return, property)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
hints_return	record	x$size_hints	write	reference
property	identifier	uns longword	read	reference

MIT C FORMAT

XSetSizeHints
(display, window_id, hints_return, property)

argument information

```
XSetSizeHints(display, window_id, hints_return, property)
      Display *display;
      Window window_id;
      XSizeHints *hints_return;
      Atom property;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to specify size hints for.

hints_return

A pointer to the size hints data structure in which the recommended values for the window are specified.

For more information on the size hints data structure, see Section 8.1.

property

The identifier of the atom that specifies the size property. The size property contains the window size hints.

DESCRIPTION

SET SIZE HINTS specifies recommended values for a window's size and location. This information is stored in the predefined property WM SIZE HINTS. This format is used with the WM NORMAL HINTS and WM ZOOM HINTS properties to recommend sizing and location information for windows in their regular and zoom states. It can also be used with any other property that has the predefined property format of WM SIZE HINTS. A window manager program can use this information to size and locate the window according to your specifications. However, a window manager may not use this information.

The following values are specified in the size hints data structure:

- Which values have been specified (the flags member)
- The x- and y-coordinates of the initial window location
- The desired width and height of the window
- The minimum width and height of the window
- The maximum width and height of the window
- An increment to be added to the minimum width and height
- The preferred aspect ratios

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET STANDARD PROPERTIES

Sets the window name, icon name, icon pixmap, command line, amd window sizing for the specified window.

VAX FORMAT

X$SET_STANDARD_PROPERTIES *(display, window_id, window_name, icon_name, icon_pixmap, command, num_args, hints)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
window_name	char string	char string	read	descriptor
icon_name	char string	char string	read	descriptor
icon_pixmap	identifier	uns longword	read	reference
command	char string	char string	read	descriptor
num_args	longword	longword	read	reference
hints	record	x$size_hints	read	reference

MIT C FORMAT

XSetStandardProperties *(display, window_id, window_name, icon_name, icon_pixmap, command, num_args, hints)*

argument information

```
XSetStandardProperties(display, window_id, window_name,
                       icon_name, icon_pixmap, command,
                       num_args, hints)
      Display *display;
      Window window_id;
      char *window_name;
      char *icon_name;
      Pixmap icon_pixmap;
      char **command;
      int num_args;
      XSizeHints *hints;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to set the properties for.

window_name

The name of the window specified in **window_id**. This argument sets the WM_NAME property.

icon_name

The name to be displayed in the icon. This argument sets the WM_ICON_NAME property.

icon_pixmap

The identifier of the pixmap storing the icon to be associated with the window running the program. If no pixmap is used, specify None. This argument specifies the icon_pixmap member of the WM_HINTS property.

command

The name of the command and the list of arguments used to invoke the program. This argument sets the WM_COMMAND property.

num_args

The number of arguments in the command argument list.

hints

The size hints data structure lists the recommended window sizes for the program. This argument sets the WM_NORMAL_HINTS property.

For more information on the size hints data structure, see Section 8.1.

DESCRIPTION

SET STANDARD PROPERTIES sets five essential window properties for your program. The five properties are as follows:

- WM_NAME—The name of the window
- WM_ICON_NAME—The name to be displayed when the icon representation of the window is displayed
- WM_HINTS—The flags field and the icon pixmap field
- WM_COMMAND—The name of the command used to invoke your application program, along with the list of its arguments
- WM_NORMAL_HINTS—The recommended window sizes for the regular window running your program

No default values are assigned to these properties. If you do not set them, the window manager program determines the default values. However, even if you do set these properties, it does not guarantee that the window manager will follow them.

Use this routine when you want to set these five properties and no others. If you want to set more than these five properties or only some of these properties, use the individual routines and CHANGE PROPERTY.

The following routines can set some of these properties individually:

- SET COMMAND—Sets the WM_COMMAND property
- SET ICON NAME—Sets the WM_ICON_NAME property
- SET NORMAL HINTS—Sets the WM_NORMAL_HINTS property
- SET WMHINTS—Sets the complete WM_HINTS property
- STORE NAME—Sets the WM_NAME property

To set other properties, use CHANGE PROPERTY.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET TRANSIENT FOR HINT

Sets the WM_TRANSIENT_FOR property of a specified window.

VAX FORMAT

X$SET_TRANSIENT_FOR_HINT
(display, window_id, prop_window_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
prop_window_id	identifier	uns longword	read	reference

MIT C FORMAT

XSetTransientForHint
(display, window_id, prop_window_id)

argument information

```
XSetTransientForHint(display, window_id, prop_window_id)
      Display *display;
      Window window_id;
      Window prop_window_id;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The window for which you want to set the transient-for property.

prop_window_id
The window identifier that the WM_TRANSIENT_FOR property is to be set to.

DESCRIPTION

SET TRANSIENT FOR HINT sets the transient-for property of a specified window. Window managers may in turn use this information to unmap an application's dialog boxes. A transient window is a temporary window that acts on behalf of another window (for example, a popup dialog box that partially obscures the main application window). Setting the transient-for property on the popup window allows the window manager to automatically iconify the popup window when it iconifies the main application window.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET WM HINTS

Sets the values for the window manager hints.

VAX FORMAT

X$SET_WM_HINTS
(display, window_id, wmhints)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
wmhints	record	x$wm_hints	read	reference

MIT C FORMAT

XSetWMHints
(display, window_id, wmhints)

argument information

```
XSetWMHints(display, window_id, wmhints)
      Display *display;
      Window window_id;
      XWMHints *wmhints;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to set the window manager hints for.

wmhints

The window manager hints data structure in which values will be set.

For more information on the window manager hints data structure, see Section 8.3.

DESCRIPTION

SET WM HINTS sets the values in the WM HINTS property for the window manager hints. SET WM HINTS also tells whether the application relies on the window manager for input, tells what its initial state should be, and the identifier of a related window group. An application program can use this routine to recommend icon information and location to the window manager program. However, the window manager may not accept these recommendations.

Use the GET WM HINTS routine to return the window manager hints that may be set in the WM_HINTS property.

The following window manager hints are set:

- Whether the program relies on the window manager to get keyboard input
- How a program will be started (as a regular window, a zoom window, an icon, or not important how started), or the initial state of the window
- A pixmap to be used for the icon representation
- A window identifier of a window to be used as the icon
- The initial position of the icon

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET ZOOM HINTS

Sets recommended size and location for a window in the zoomed state.

VAX FORMAT

X$SET_ZOOM_HINTS
(display, window_id, zhints_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
zhints_return	record	x$size_hints	read	reference

MIT C FORMAT

XSetZoomHints
(display, window_id, zhints_return)

argument information

```
XSetZoomHints(display, window_id, zhints_return)
      Display *display;
      Window window_id;
      XSizeHints *zhints_return;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
The identifier of the zoom window to set recommended values for.

zhints_return
The size hints data structure containing the recommended values for the zoom window.

For more information on the size hints data structure, see Section 8.1.

DESCRIPTION

SET ZOOM HINTS specifies recommended values for a window's zoom size and location. This information is stored in the WM_ZOOM_HINTS predefined property. A window manager program can use this information to size and locate the zoomed window according to your specifications. However, it is not guaranteed that a window manager will use this information.

The following values are specified in the size hints data structure:

- Which values have been specified (the flags member)
- The x- and y-coordinates of the initial zoom window location
- The desired width and height of the zoom window
- The minimum width and height of the zoom window
- The maximum width and height of the zoom window
- An increment to be added to the minimum width and height
- The preferred aspect ratios

It is important to set the flags member within the size hints data structure to inform the window manager which specific members have been set. If the flags member is not set, a window manager may disregard a call to SET NORMAL HINTS.

Use GET ZOOM HINTS to obtain recommended values that have already been specified.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

STORE NAME

Assigns a name to a window.

VAX FORMAT

X$STORE_NAME
(display, window_id, window_name)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
window_name	char string	char string	read	descriptor

MIT C FORMAT

XStoreName
(display, window_id, window_name)

argument information

```
XStoreName(display, window_id, window_name)
      Display *display;
      Window window_id;
      char *window_name;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to assign the name to.

window_name

The name to assign to the window.

DESCRIPTION

STORE NAME assigns the name specified in **window_name** to the WM_NAME predefined property for the window. The identifier of the window was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW.

Once the name is assigned to the window, a window manager can refer to the window by the name. The window name can be used in an icon display of the window or in a title bar.

After the name has been assigned, you can use FETCH NAME to return the name.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

UNIQUE CONTEXT

Creates a unique context type.

VAX FORMAT

context_id_return = *X$UNIQUE_CONTEXT*

argument information

Argument	Usage	Data Type	Access	Mechanism
context_id_return	identifier	uns longword	read	reference

MIT C FORMAT

context_id_return = *XUniqueContext*

argument information

```
XContext XUniqueContext()
```

RETURNS

context_id_return
The identifier of the context type.

DESCRIPTION

UNIQUE CONTEXT creates a unique context type. This type can then be used in subsequent calls to other context routines, such as FIND CONTEXT and SAVE CONTEXT.

9 Region Routines

The Xlib region routines allow you to specify a pixmap or a list of rectangles to restrict (clip) output to a particular area of a window. The image defined by the pixmap or rectangles can be of any shape and is called a *region*. The region structure is associated with a window by means of the CLIP_X_ORIGIN, CLIP_Y_ORIGIN, and CLIP_MASK members of the graphics context data structure.

You can use the SET CLIP ORIGIN, SET CLIP MASK, and SET CLIP RECTANGLES routines to manipulate the members of the graphics context data structure directly. However, the region routines provide you with a more convenient method to set the clipping region for a window, including the ability to define a region from an arbitrary array of points. The region routines also allow you to perform arithmetic operations on the regions.

To use the region routines, you first create a region using either POLYGON REGION or CREATE REGION. You can associate a region created by POLYGON REGION with a window's graphics context by using the SET REGION routine. The most common use of CREATE REGION is to create an empty region that you later pass to the other region routines as a destination.

For information on how to use the region routines, see the *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 9–1.

Table 9–1 Region Routines

Routine Name	Description
CLIP BOX	Generates the smallest rectangle that encloses a region.
CREATE REGION	Creates a new, empty region and returns the region identifier that defines it.
DESTROY REGION	Deallocates the storage space associated with a specified region.
EMPTY REGION	Indicates whether a specified region contains any points.
EQUAL REGION	Compares the offset, size, and shape of two regions to determine if they are equal.
INTERSECT REGION	Computes the intersection of two regions and stores the result as a region identifier.

(continued on next page)

Table 9–1 (Cont.) Region Routines

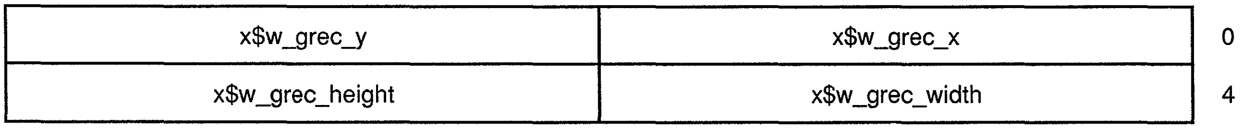
Routine Name	Description
OFFSET REGION	Moves a region by the amount of the offset that you specify.
POINT IN REGION	Determines whether a coordinate that you specify resides in a particular region.
POLYGON REGION	Generates a new region from a polygon.
RECT IN REGION	Determines whether a rectangle that you specify resides in a particular region.
SET REGION	Associates the clip mask of a graphics context with the region that you specify.
SHRINK REGION	Reduces (or expands) the size of a region by the amount that you specify.
SUBTRACT REGION	Subtracts one region from another. Used to determine the portion of the first region that does not lie within the second.
UNION RECT WITH REGION	Creates a region from the union of a source region and a rectangle.
UNION REGION	Calculates the union of two regions and stores the result in another region.
XOR REGION	Calculates the coordinates that fall within the union, but not the intersection, of two regions.

9.1 Rectangle Data Structure

The rectangle data structure describes the origin, width, and height of a rectangle.

The rectangle data structure for the VAX binding is shown in Figure 9–1.

Figure 9–1 Rectangle Data Structure (VAX Binding)



x$w_grec_y	x$w_grec_x	0
x$w_grec_height	x$w_grec_width	4

The members of the VAX binding rectangle data structure are described in Table 9–2.

Table 9–2 Members of the Rectangle Data Structure (VAX Binding)

Member Name	Contents
X$W_GREC_X	Defines the x value of the origin of the rectangle
X$W_GREC_Y	Defines the y value of the origin of the rectangle
X$W_GREC_WIDTH	Defines the width of the rectangle
X$W_GREC_HEIGHT	Defines the height of the rectangle

The rectangle data structure for the MIT C binding is shown in Figure 9–2.

Figure 9–2 Rectangle Data Structure (MIT C Binding)

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

The members of the MIT C binding rectangle data structure are described in Table 9–3.

Table 9–3 Members of the Rectangle Data Structure (MIT C Binding)

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

9.2 Region Routines

The following pages describe the Xlib region routines.

CLIP BOX

Generates the smallest rectangle that encloses a region.

VAX FORMAT

X$CLIP_BOX
(region_id, rectangle_struc_return)

argument information

Argument	usage	Data Type	Access	Mechanism
region_id	identifier	uns longword	read	reference
rectangle_struc_return	record	x$rectangle	write	reference

MIT C FORMAT

XClipBox
(region_id, rectangle_struc_return)

argument information

```
XClipBox(region_id, rectangle_struc_return)
      Region region_id;
      XRectangle *rectangle_struc_return;
```

ARGUMENTS

region_id

The region you want to enclose in a rectangle. The **region_id** argument is returned by CREATE REGION or POLYGON REGION when the region is created.

rectangle_struc_return

The rectangle that encloses the region specified in **region_id**. CLIP BOX returns the smallest enclosing rectangle to this structure.

The rectangle data structure is shown in Section 9.1.

DESCRIPTION

CLIP BOX generates the smallest rectangle that encloses **region_id** and returns it in **rectangle_struc_return**. The rectangle data structure is shown in Section 9.1.

CREATE REGION

Creates a new, empty region and returns the region identifier that defines it.

VAX FORMAT

region_id_return = **X$CREATE_REGION** *()*

argument information

Argument	Usage	Data Type	Access	Mechanism
region_id_return	identifier	uns longword	write	reference

MIT C FORMAT

region_id_return = **XCreateRegion** *()*

argument information

```
Region XCreateRegion()
```

RETURNS

region_id_return
The region identifier that describes the new region. This region identifier is used in routines such as INTERSECT REGION that store the result of a mathematical operation in a region identifier.

DESCRIPTION

CREATE REGION creates a new, empty region and returns a region identifier that you pass as a destination to other routines such as INTERSECT REGION.

DESTROY REGION

Deallocates the storage space associated with a specified region.

VAX FORMAT

X$DESTROY_REGION *(region_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
region_id	identifier	uns longword	read	reference

MIT C FORMAT

XDestroyRegion *(region_id)*

argument information

```
XDestroyRegion(region_id)
      Region region_id;
```

ARGUMENTS

region_id

The region identifier of the region that you want to destroy. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

DESTROY REGION deallocates the region that you specify by deallocating its storage space.

EMPTY REGION

Indicates whether a specified region contains any points.

VAX FORMAT

answer_return = **X$EMPTY_REGION** *(region_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
answer_return	longword	longword	write	value
region_id	identifier	uns longword	read	reference

MIT C FORMAT

answer_return = **XEmptyRegion** *(region_id)*

argument information

```
Bool XEmptyRegion(region_id)
      Region region_id;
```

RETURNS

answer_return

When the value of **answer_return** is zero the region is not empty. When the value of **answer_return** is nonzero the region is empty.

ARGUMENTS

region_id

The region identifier of the region that you want to test. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

EMPTY REGION determines whether the region that you specify is empty.

EQUAL REGION

Compares the offsets, sizes, and shapes of two regions to determine if they are equal.

VAX FORMAT

answer_return = **X$EQUAL_REGION** *(region1_id, region2_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
answer_return	longword	longword	write	value
region1_id	identifier	uns longword	read	reference
region2_id	identifier	uns longword	read	reference

MIT C FORMAT

answer_return = **XEqualRegion** *(region1_id, region2_id)*

argument information

```
Bool XEqualRegion(region1_id, region2_id)
      Region region1_id, region2_id;
```

RETURNS

answer_return

When the value of **answer_return** is zero the regions are not equal. When the value of **answer_return** is nonzero the regions are equal.

ARGUMENTS

region1_id

The region identifier of the first region to be compared. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

region2_id

The region identifier of the second region to be compared. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

EQUAL REGION compares the offsets, sizes, and shapes of two regions to determine if they are equal and returns a value.

INTERSECT REGION

Computes the intersection of two regions and stores the result as a region identifier.

VAX FORMAT

X$INTERSECT_REGION
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
src_region1_id	identifier	uns longword	read	reference
src_region2_id	identifier	uns longword	read	reference
dst_region_id_return	identifier	uns longword	write	reference

MIT C FORMAT

XIntersectRegion
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

```
XIntersectRegion(src_region1_id, src_region2_id,
                 dst_region_id_return)
 Region src_region1_id, src_region2_id, dst_region_id_return;
```

ARGUMENTS

src_region1_id

The region identifier of one of the regions for which you want to compute the intersection. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

src_region2_id

The region identifier of the other region for which you want to compute the intersection. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

dst_region_id_return

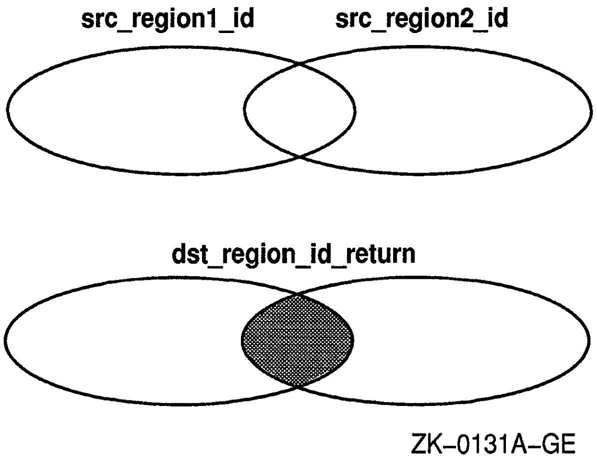
The region identifier in which to store the result of the intersection computation. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

INTERSECT REGION computes the intersection of two regions and stores the value in the region defined by **dst_region_id_return**. The intersection of two regions is the largest area that is common to the two regions.

Figure 9–3 shows the intersection of two regions.

Figure 9–3 Region Intersection



OFFSET REGION

Moves a region by the amount of the offset that you specify.

VAX FORMAT

X$OFFSET_REGION
(region_id, x_offset, y_offset)

argument information

Argument	Usage	Data Type	Access	Mechanism
region_id	identifier	uns longword	read	reference
x_offset	longword	longword	read	reference
y_offset	longword	longword	read	reference

MIT C FORMAT

XOffsetRegion
(region_id, x_offset, y_offset)

argument information

```
XOffsetRegion(region_id, x_offset, y_offset)
      Region region_id;
      int x_offset, y_offset;
```

ARGUMENTS

region_id

The identifier of the region that you want to move. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

x_offset

The x-offset by which you want to move the region. The offset that you specify is relative to the origin of the region.

y_offset

The y-offset by which you want to move the region. The offset that you specify is relative to the origin of the region.

DESCRIPTION

OFFSET REGION uses **region_id**, **x-offset**, and **y-offset** to move a region. The size and shape of the region are not affected. Positive values for the x- and y-offsets move the region along the positive axis; negative values move the region along the negative axis.

POINT IN REGION

Determines whether a point whose coordinates you specify resides in a particular region.

VAX FORMAT

answer_return = **X$POINT_IN_REGION** *(region_id, x_coord, y_coord)*

argument information

Argument	Usage	Data Type	Access	Mechanism
answer_return	longword	longword	write	value
region_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference

MIT C FORMAT

answer_return = **XPointInRegion** *(region_id, x_coord, y_coord)*

argument information

```
Bool XPointInRegion(region_id, x_coord, y_coord)
      Region region_id;
      int x_coord, y_coord;
```

RETURNS

answer_return

When the value of **answer_return** is zero the point is not in the region. When the value of **answer_return** is nonzero the point is within the region.

ARGUMENTS

region_id

The identifier of the region that you want to evaluate. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

x_coord

The x-coordinate of the point that you want to evaluate. The x-coordinate is relative to the region's origin.

y_coord

The y-coordinate of the point that you want to evaluate. The y-coordinate is relative to the region's origin.

DESCRIPTION

POINT IN REGION evaluates a region to determine whether the point whose coordinates you specify resides within it.

POLYGON REGION

Generates a new region from a polygon.

VAX FORMAT

region_id_return = **X$POLYGON_REGION** *(points, num_points, fill_rule)*

argument information

Argument	Usage	Data Type	Access	Mechanism
region_id_return	identifier	uns longword	write	value
points	array	uns longword	read	reference
num_points	longword	longword	read	reference
fill_rule	longword	uns longword	read	reference

MIT C FORMAT

region_id_return = **XPolygonRegion** *(points, num_points, fill_rule)*

argument information

```
Region XPolygonRegion(points, num_points, fill_rule)
      XPoint points[];
      int num_points;
      int fill_rule;
```

RETURNS

region_id_return

POLYGON REGION returns the region identifier when the region is created.

ARGUMENTS

points

A pointer to the array of points used to create the region.

num_points

The number of points in the polygon. The **num_points** argument reflects the number of points in the points array.

fill_rule

The fill rule that you want to set for the specified graphics context. The fill rule defines which pixels are inside (drawn) for paths given in FILL POLYGON requests.

The predefined values for **fill_rule** are described in Table 9–4.

Table 9–4 Fill Rule Constants

VAX Binding	MIT C Binding	Description
X$C_EVEN_ODD_RULE	EvenOddRule	A point is inside if an infinite ray with the point as origin crosses the path an odd number of times.
X$C_WINDING_RULE	WindingRule	A point is inside if an infinite ray with the point as origin crosses an unequal number of clockwise and counterclockwise path segments. A clockwise path segment is one that crosses the ray from left to right as observed from the point. A counterclockwise path 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 that is not coincident with a segment.

For both the Even Odd Rule and Winding Rule constants 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).

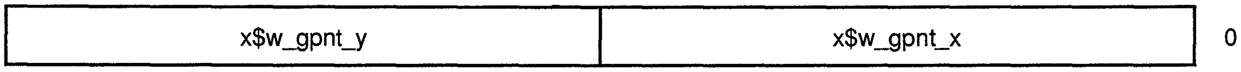
Pixels with centers that are along a horizontal edge are a special case and are inside if, and only if, the polygon interior is immediately below (y increasing direction).

DESCRIPTION

POLYGON REGION returns a region identifier for the polygon defined by the points array and the fill rule that you specify. The point data structure describes the x- and y-coordinates of a point.

The point data structure for the VAX binding is shown in Figure 9–4.

Figure 9–4 Point Data Structure (VAX Binding)



x$w_gpnt_y	x$w_gpnt_x	0

The members of the VAX binding point data structure are described in Table 9–5.

Table 9–5 Members of the Point Data Structure (VAX Binding)

Member Name	Contents
X$W_GPNT_X	Defines the x-coordinate of a point
X$W_GPNT_Y	Defines the y-coordinate of a point

The point data structure for the MIT C binding is shown in Figure 9–5.

Figure 9–5 Point Data Structure (MIT C Binding)

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

The members of the MIT C binding point data structure are described in Table 9–6.

Table 9–6 Members of the Point Data Structure (MIT C Binding)

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

RECT IN REGION

Determines whether a rectangle that you specify resides in a particular region.

VAX FORMAT

answer_return = **X$RECT_IN_REGION**
(region_id, x_coord, y_coord, width, height)

argument information

Argument	Usage	Data Type	Access	Mechanism
answer_return	longword	longword	write	value
region_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference

MIT C FORMAT

answer_return = **XRectInRegion**
(region_id, x_coord, y_coord, width, height)

argument information

```
int XRectInRegion(region_id, x_coord, y_coord, width, height)
      Region region_id;
      int x_coord, y_coord;
      unsigned int width, height;
```

RETURNS

answer_return

RECT IN REGION returns the following values:

VAX Binding	MIT C Binding	Description
X$C_RECTANGLE_IN	RectangleIn	The rectangle is entirely inside the specified region
X$C_RECTANGLE_OUT	RectangleOut	The rectangle is entirely outside the specified region
X$C_RECTANGLE_PART	RectanglePart	The rectangle is partially inside the specified region

ARGUMENTS

region_id

The region identifier of the region to be evaluated. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

x_coord

The x-coordinate of the upper left corner of the rectangle that you want to evaluate.

y_coord

The y-coordinate of the upper left corner of the rectangle that you want to evaluate.

width

The width, in pixels, of the rectangle to be evaluated. The width and height determine the area of the rectangle to be evaluated.

height

The height, in pixels, of the rectangle to be evaluated. The width and height determine the area of the rectangle to be evaluated.

DESCRIPTION

RECT IN REGION evaluates a region to determine whether the rectangle that you specify resides within it and returns a value to indicate the status.

SET REGION

Associates the clip mask of a graphics context with the region that you specify.

VAX FORMAT

X$SET_REGION
(display, gc_id, region_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
region_id	identifier	uns longword	read	reference

MIT C FORMAT

XSetRegion
(display, gc_id, region_id)

argument information

```
XSetRegion(display, gc_id, region_id)
      Display *display;
      GC gc_id;
      Region region_id;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

gc_id
The identifier of the graphics context that you want to associate with the region.

region_id
The region identifier of the region that you want to associate with a graphics context. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

SET REGION sets the clip mask in the graphics context to the specified region. After the clip mask is set in the graphics context, the region can be destroyed. When the window is redrawn, output to the window that is using this graphics context is restricted to the area defined by the region.

SHRINK REGION

Reduces (or expands) the size of a region by the amount that you specify.

VAX FORMAT

X$SHRINK_REGION
(region_id, x_offset, y_offset)

argument information

Argument	Usage	Data Type	Access	Mechanism
region_id	identifier	uns longword	read	reference
x_offset	longword	longword	read	reference
y_offset	longword	longword	read	reference

MIT C FORMAT

XShrinkRegion
(region_id, x_offset, y_offset)

argument information

```
XShrinkRegion(region_id, x_offset, y_offset)
      Region region_id;
      int x_offset, y_offset;
```

ARGUMENTS

region_id

The region identifier of the region that you want to shrink or expand. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

x_offset

The x-offset by which you want to reduce or expand the region. Positive values reduce the size of the region; negative values expand the size of the region.

y_offset

The y-offset by which you want to reduce or expand the region. Positive values reduce the size of the region; negative values expand the size of the region.

DESCRIPTION

SHRINK REGION uses the **region_id**, **x_offset**, and **y_offset** values that you specify to reduce or expand the size of a region while leaving it centered at the same position. Positive values reduce the size of the region; negative values expand the size of the region. SHRINK REGION applies half of the specified x- and y-offsets to the coordinates of each corner in the following way:

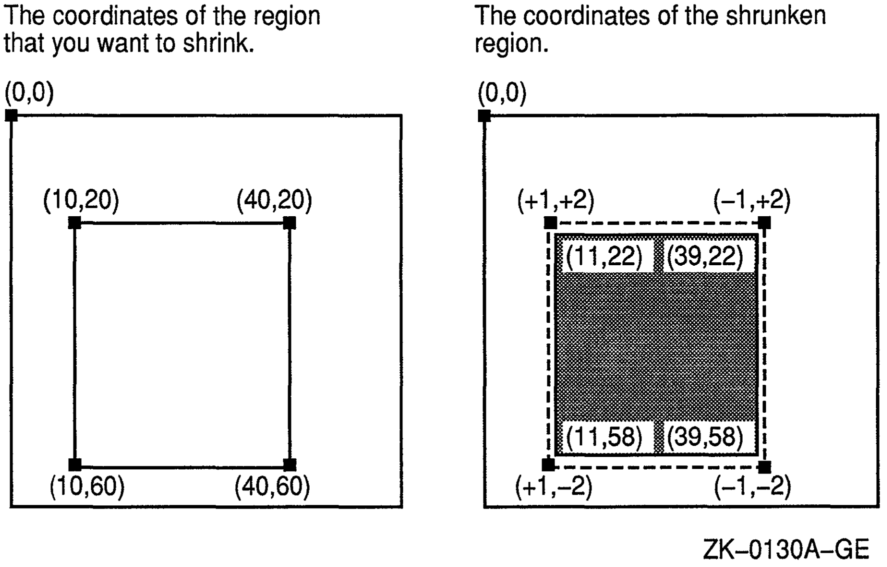
Corner	Shrink Region	Expand Region
Upper left	(+x,+y)	(–x,–y)
Upper right	(–x,+y)	(+x,–y)
Lower left	(+x,–y)	(–x,+y)
Lower right	(–x,–y)	(+x,+y)

For example, assume the relative coordinates (10,20), (40,20), (10,60), and (40,60) define the corners of a region. If you supply SHRINK REGION with the values **x_offset** =2 and **y_offset** =4, SHRINK REGION divides the values by 2 and adds or subtracts them as follows:

Corner	Current Coordinates	Shrink Region	Result
Upper left	(10,20)	(+1,+2)	(11,22)
Upper right	(40,20)	(–1,+2)	(39,22)
Lower left	(10,60)	(+1,–2)	(11,58)
Lower right	(40,60)	(–1,–2)	(39,58)

Figure 9–6 shows the result of SHRINK REGION.

Figure 9–6 Shrinking a Region



SUBTRACT REGION

Subtracts one region from another. Used to determine the portion of the first region that does not lie within the second.

VAX FORMAT

X$SUBTRACT_REGION
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
src_region1_id	identifier	uns longword	read	reference
src_region2_id	identifier	uns longword	read	reference
dst_region_id_return	identifier	uns longword	write	reference

MIT C FORMAT

XSubtractRegion
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

```
XSubtractRegion(src_region1_id, src_region2_id,
                dst_region_id_return)
  Region src_region1_id, src_region2_id, dst_region_id_return;
```

ARGUMENTS

src_region1_id

The region identifier of the minuend. This is the region from which to subtract **src_region2_id**. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

src_region2_id

The region identifier of the subtrahend. This is the region to subtract from **src_region1_id**. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

dst_region_id_return

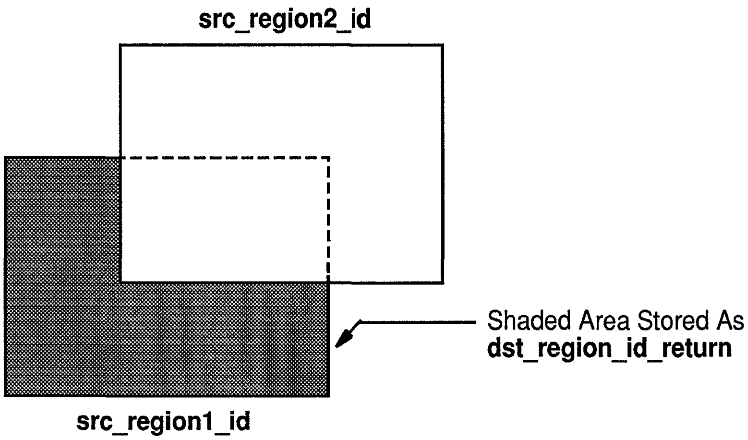
The identifier of the region in which to store the result of the subtraction. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

SUBTRACT REGION subtracts the region specified by **src_region2_id** from the region specified by **src_region1_id**. Any part of **src_region1_id** that is not within **src_region2_id** is stored in **dst_region_id_return**.

Figure 9–7 shows the result of SUBTRACT REGION.

Figure 9–7 Subtracting a Region



ZK–0132A–GE

UNION RECT WITH REGION

Creates a region from the union of a source region and a rectangle.

VAX FORMAT

X$UNION_RECT_WITH_REGION
(rectangle_struc, src_region_id, dst_region_id_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
rectangle_struc	record	x$rectangle	read	reference
src_region_id	identifier	uns longword	read	reference
dst_region_id_return	identifier	uns longword	write	reference

MIT C FORMAT

XUnionRectWithRegion
(rectangle_struc, src_region_id, dst_region_id_return)

argument information

```
XUnionRectWithRegion(rectangle_struc, src_region_id,
                     dst_region_id_return)
     Rectangle *rectangle_struc;
     Region  src_region_id, dst_region_id_return;
```

ARGUMENTS

rectangle_struc

The rectangle for which you want to compute the union.

src_region_id,

The identifier of the region for which you want to compute the union. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

dst_region_id_return

The identifier of the region in which to store the result of the union computation. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

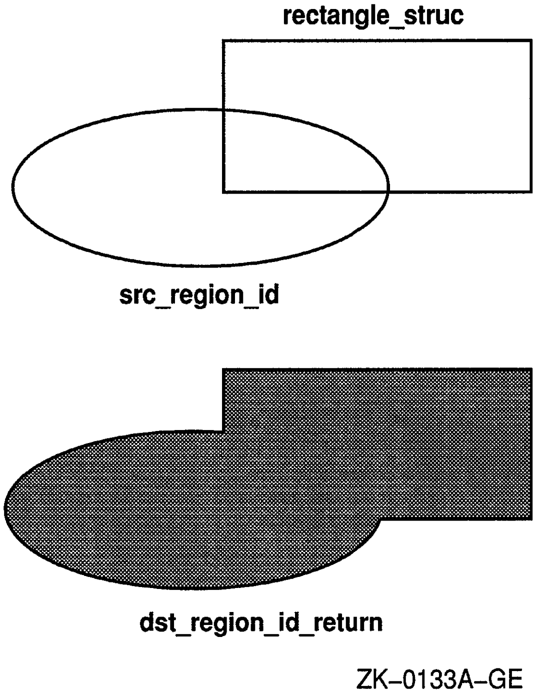
DESCRIPTION

UNION RECT WITH REGION creates a region from the union of a source region and a rectangle. The created region is defined by **dst_region_id_return**. The union includes any area that is in either the rectangle or region or both.

The rectangle data structure is shown in Section 9.1.

Figure 9–8 shows the result of UNION RECT WITH REGION.

Figure 9–8 Union of a Source Region and a Rectangle



UNION REGION

Calculates the union of two regions and stores the result in another region.

VAX FORMAT

X$UNION_REGION
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
src_region1_id	identifier	uns longword	read	reference
src_region2_id	identifier	uns longword	read	reference
dst_region_id_return	identifier	uns longword	write	reference

MIT C FORMAT

XUnionRegion
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

```
XUnionRegion(src_region1_id, src_region2_id,
             dst_region_id_return)
  Region src_region1_id, src_region2_id, dst_region_id_return;
```

ARGUMENTS

src_region1_id

The identifier of one of the regions for which you want to compute the union. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

src_region2_id

The identifier of the other region for which you want to compute the union. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

dst_region_id_return

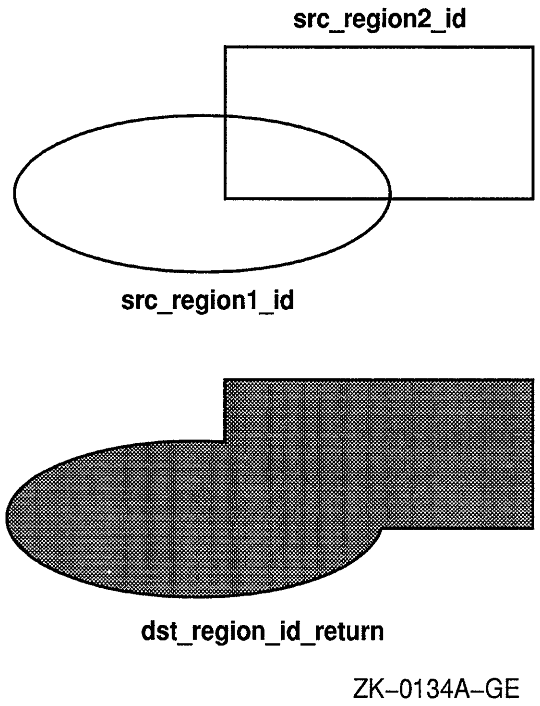
The identifier of the region in which to store the result of the union computation. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

UNION REGION computes the union of two regions and stores its value in the region defined by **dst_region_id_return**. The union of two regions includes any area that is in either or both regions.

Figure 9–9 shows the result of UNION REGION.

Figure 9–9 Union of Two Regions



XOR REGION

Calculates the coordinates that fall within the union, but not the intersection, of two regions.

VAX FORMAT

X$XOR_REGION
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
src_region1_id	identifier	uns longword	read	reference
src_region2_id	identifier	uns longword	read	reference
dst_region_id_return	identifier	uns longword	write	reference

MIT C FORMAT

XXorRegion
(src_region1_id, src_region2_id, dst_region_id_return)

argument information

```
XXorRegion(src_region1_id, src_region2_id,
           dst_region_id_return)
 Region src_regionl_id, src_region2_id, dst_region_id_return;
```

ARGUMENTS

src_region1_id

The region identifier of one of the regions for which you want to calculate the XOR. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

src_region2_id

The region identifier of the other region for which you want to calculate the XOR. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

dst_region_id_return

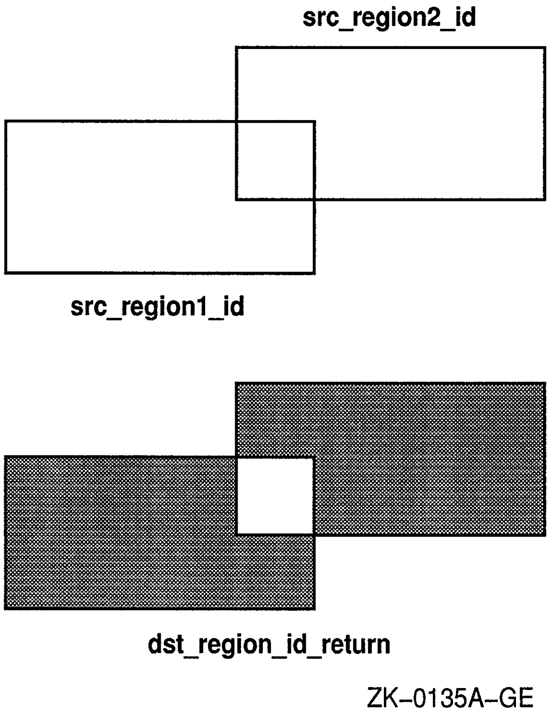
The identifier of the region in which to store the result of the XOR operation. The region identifier is returned by CREATE REGION or POLYGON REGION when the region is created.

DESCRIPTION

XOR REGION calculates the difference between the union and intersection of two regions. XOR REGION performs an exclusive OR operation and stores the region that falls within either region, but not both, in **dst_region_id_return**.

Figure 9–10 shows the result of XOR REGION.

Figure 9–10 Exclusive OR Operation



10 Window and Session Manager Routines

A window or session manager program performs the following types of tasks:

- Manipulating windows
- Manipulating color maps
- Manipulating the pointer
- Manipulating the keyboard
- Manipulating the server
- Controlling processing to other connections
- Manipulating keyboard settings
- Manipulating the screen saver
- Controlling host access
- Parsing window geometry
- Obtaining Xlib environment information

Note: **Most clients are not responsible for window or session management and do not need to use these routines. A client could use these routines if there were no formal window or session manager program. However, using window or session manager routines must be done with great care as they can affect the operation of other applications.**

For information on how to use the window manager routines, see the *VMS DECwindows Xlib Programming Volume.*

The routines described in this chapter are listed in Table 10–1.

Table 10–1 Window and Session Manager Routines

Routine Name	Description
ACTIVATE SCREEN SAVER	Enables the screen saver, even if it is currently disabled.
ADD HOST	Adds a host to the list of hosts that can connect to a display.
ADD HOSTS	Adds more than one host to the list of hosts that can connect to a display.
ADD TO SAVE SET	Adds a window to the client's save set.

(continued on next page)

Table 10–1 (Cont.) Window and Session Manager Routines

Routine Name	Description
ALLOW EVENTS	Releases events that were queued because a device was grabbed.
AUTO REPEAT OFF	Turns off keyboard auto-repeat.
AUTO REPEAT ON	Turns on keyboard auto-repeat.
BELL	Rings the keyboard bell at the base volume that you specify.
CHANGE ACTIVE POINTER GRAB	Changes the dynamic parameters for an active grab.
CHANGE KEYBOARD CONTROL	Changes the keyboard settings for the key click volume, base bell volume, LEDs, and auto-repeat keys.
CHANGE KEYBOARD MAPPING	Specifies key symbols for the selected key codes.
CHANGE POINTER CONTROL	Controls the interactive feel of the pointing device.
CHANGE SAVE SET	Adds or removes a window from the client's save set.
DELETE MODIFIERMAP ENTRY	Deletes an entry from a modifier key map structure.
DISABLE ACCESS CONTROL	Disables access control mode for a display.
ENABLE ACCESS CONTROL	Enables access control mode for a display.
FORCE SCREEN SAVER	Activates the screen saver in the specified mode.
FREE MODIFIERMAP	Destroys the specified modifier key map structure.
GEOMETRY	Parses window geometry.
GET DEFAULT	Returns the default property string for the user environment.
GET INPUT FOCUS	Obtains information about the current input focus.
GET KEYBOARD CONTROL	Obtains the current control values for the keyboard.
GET KEYBOARD MAPPING	Returns the key symbols for one or more than one key code.
GET MODIFIER MAPPING	Returns the key codes for the modifier keys.
GET POINTER CONTROL	Returns the pointer movement values for acceleration and the threshold at which acceleration should be applied.
GET POINTER MAPPING	Returns the mapping list, which defines which buttons are enabled for the pointing device.

(continued on next page)

Table 10–1 (Cont.) Window and Session Manager Routines

Routine Name	Description
GET SCREEN SAVER	Returns the following values for screen saving: the timeout period, the interval, whether to blank the screen, and whether to allow exposures.
GRAB BUTTON	Grabs a pointer button.
GRAB KEY	Passively grabs one key and specifies the processing of the key event.
GRAB KEYBOARD	Actively grabs control of the main keyboard and defines the processing of pointer events.
GRAB POINTER	Actively grabs the specified pointer.
GRAB SERVER	Takes exclusive possession of the server associated with the display.
INSERT MODIFIERMAP ENTRY	Adds a new entry to the modifier key map structure.
INSTALL COLORMAP	Overwrites the current color map with the entries from the specified color map.
KEYCODE TO KEYSYM	Converts the key code that you specify to a defined key symbol.
KEYSYM TO KEYCODE	Converts the key symbol that you specify to a defined key code.
KEYSYM TO STRING	Converts the key-symbol code that you specify to the name of the key symbol.
KILL CLIENT	Disconnects a client associated with the specified resource.
LIST HOSTS	Returns the list of hosts that can access a display.
LIST INSTALLED COLORMAPS	Returns a color map identifier of each installed color map for a window.
LOOKUP KEYSYM	Returns the key symbol from the list that corresponds to the key code in the event that you specify.
LOOKUP STRING	Maps a key event to an ISO-Latin1 string.
NEW MODIFIER MAP	Creates a new modifier key map data structure.
PARSE COLOR	Provides the red, green, and blue values for a named color.
PARSE GEOMETRY	Parses standard geometry strings.
QUERY KEYMAP	Returns a bit vector that describes that state of the keyboard.
REBIND KEYSYM	Rebinds the meaning of a key symbol for a client program.

(continued on next page)

Table 10–1 (Cont.) Window and Session Manager Routines

Routine Name	Description
REFRESH KEYBOARD MAPPING	Refreshes the stored modifier and key map information.
REMOVE FROM SAVE SET	Removes the specified window from the client's save set.
REMOVE HOST	Removes a host from the list of hosts that can connect to a display.
REMOVE HOSTS	Removes multiple hosts from the list of hosts that can connect to a display.
REPARENT WINDOW	Changes the parent window for the specified window and repositions the window within the new parent's hierarchy.
RESET SCREEN SAVER	Resets the screen saver.
SET ACCESS CONTROL	Changes the access control mode of a display to enabled or disabled.
SET CLOSE DOWN MODE	Defines what happens to a client's resources when the client disconnects.
SET INPUT FOCUS	Changes the input focus to the specified window.
SET MODIFIER MAPPING	Specifies the key codes for the modifier keys.
SET POINTER MAPPING	Enables or disables buttons for the pointer.
SET SCREEN SAVER	Sets the following values for screen saving: the timeout period, the interval, whether to blank the screen, and whether to allow exposures.
STRING TO KEYSYM	Converts the name of the key symbol to the name of the key symbol code.
UNGRAB BUTTON	Deactivates the passive grab for a pointing device button press.
UNGRAB KEY	Releases the key combination on the specified window that was grabbed.
UNGRAB KEYBOARD	Releases an active grab on the main keyboard and any queued events.
UNGRAB POINTER	Releases the active grab on the specified pointer and any queued events.
UNGRAB SERVER	Relinquishes exclusive possession of the server associated with the display that you specify.
UNINSTALL COLORMAP	Uninstalls a color map for a screen.
WARP POINTER	Moves the pointer to any specified location on the screen.

10.1 Network Data structure

The network data structure specifies the format of the network address for a display.

The VAX binding network data structure is shown in Figure 10–1.

Figure 10–1 Network Data Structure (VAX Binding)

x$l_host_family	0
x$l_host_length	4
x$a_host_address	8

The members of the VAX binding network data structure are described in Table 10–2.

Table 10–2 Members of the Network Data Structure (VAX Binding)

Member Name	Contents
X$L_HOST_FAMILY	Specifies which protocol address family to use. The constant X$C_FAMILY_DECNET or FamilyDECnet identifies the DECnet protocol.
X$L_HOST_LENGTH	The length of the address, in bytes.
X$A_HOST_ADDRESS	A pointer to host address.

The MIT C binding network data structure is shown in Figure 10–2.

Figure 10–2 Network Data Structure (MIT C Binding)

```
typedef struct {
        int family;
        int length;
        char *address;
}XHostAddress;
```

The members of the MIT C binding network data structure are described in Table 10–3.

Table 10–3 Members of the Network Data Structure (MIT C Binding)

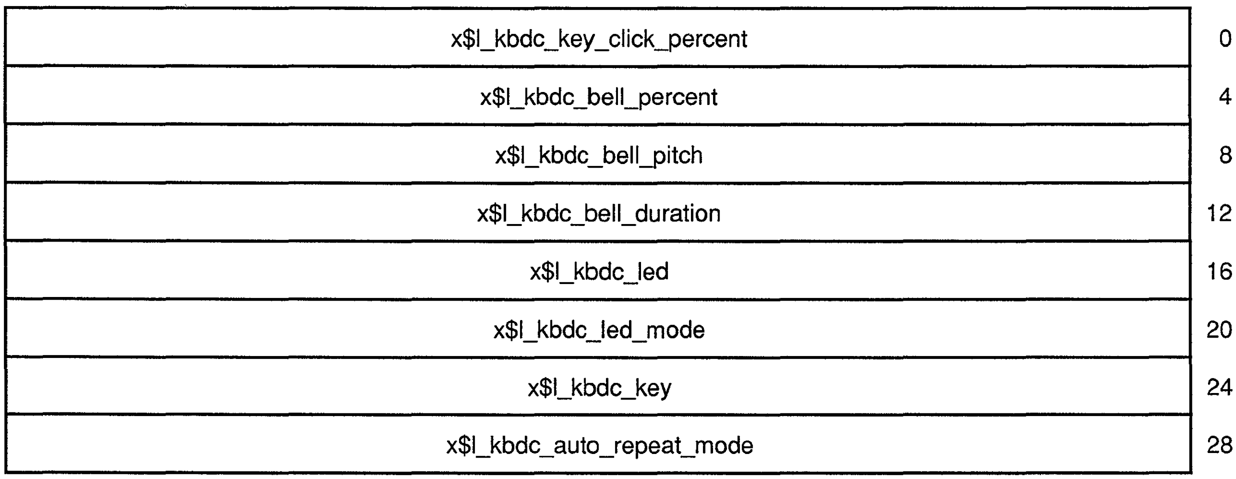
Member Name	Contents
family	Specifies which protocol address family to use. The constant X$C_FAMILY_DECNET or FamilyDECnet identifies the DECnet protocol.
length	The length of the address, in bytes.
address	A pointer to host address.

10.2 Keyboard Control Data Structure

A window or session manager program can set user-controlled keyboard preferences such as key click volume, bell volume, auto-repeat state, and LED state. You use the keyboard control value mask to specify values for the members of the keyboard control data structure. Table 10–6 lists the predefined values and descriptions for setting the value mask.

The VAX binding keyboard control data structure is shown in Figure 10–3.

Figure 10–3 Keyboard Control Data Structure (VAX Binding)



x$l_kbdc_key_click_percent	0
x$l_kbdc_bell_percent	4
x$l_kbdc_bell_pitch	8
x$l_kbdc_bell_duration	12
x$l_kbdc_led	16
x$l_kbdc_led_mode	20
x$l_kbdc_key	24
x$l_kbdc_auto_repeat_mode	28

The members of the VAX binding keyboard control data structure are described in Table 10–4.

Table 10–4 Members of the Keyboard Control Data Structure (VAX Binding)

Member Name	Contents
X$L_KBDC_KEY_CLICK_PERCENT	Controls the volume for key clicks between 0 (off) and 100 (loud), inclusive, if possible.
X$L_KBDC_BELL_PERCENT	Controls the base volume for the bell between 0 (off) and 100 (loud), inclusive, if possible.
X$L_KBDC_BELL_PITCH	Controls the pitch (specified in Hz) of the bell, if possible.
X$L_KBDC_BELL_DURATION	Controls the duration, specified in milliseconds, of the bell, if possible.
X$L_KBDC_LED	Changes the keyboard LED.
X$L_KBDC_LED_MODE	Changes the keyboard LED mode.
X$L_KBDC_KEY	Changes the keyboard auto-repeat key.
X$L_KBDC_AUTO_REPEAT_MODE	Changes the keyboard auto-repeat mode.

The MIT C binding keyboard control data structure is shown in Figure 10–4.

Figure 10–4 Keyboard Control Data Structure (MIT C Binding)

```
typedef struct {
        int key_click_percent;
        int bell_percent;
        int bell_pitch;
        int bell_duration;
        int led;
        int led_mode;
        int key;
        int auto_repeat_mode;
} XKeyboardControl;
```

The members of the MIT C binding keyboard control data structure are described in Table 10–5.

Table 10–5 Members of the Keyboard Control Data Structure (MIT C Binding)

Member Name	Contents
key_click_percent	Controls the volume for key clicks between 0 (off) and 100 (loud), inclusive, if possible.
bell_percent	Controls the base volume for the bell between 0 (off) and 100 (loud), inclusive, if possible.
bell_pitch	Controls the pitch (specified in Hz) of the bell, if possible.
bell_duration	Controls the duration, specified in milliseconds, of the bell, if possible.

(continued on next page)

Table 10–5 (Cont.) Members of the Keyboard Control Data Structure (MIT C Binding)

Member Name	Contents
led	Changes the keyboard LED.
led_mode	Changes the keyboard LED mode.
key	Changes the keyboard auto-repeat key.
auto_repeat_mode	Changes the keyboard auto-repeat mode.

10.2.1 Keyboard Control Value Mask

Table 10–6 lists the predefined values and descriptions for setting the value mask.

Table 10–6 Keyboard Control Value Mask

Bit	VAX Predefined Value	MIT C Predefined Value	Meaning
1	X$M_KB_KEY_ CLICK_PERCENT	KBKeyClickPercent	Sets the volume for key clicks between 0 (off) and 100 (loud), inclusive, if possible. A setting of –1 restores the default. Other negative values generate a Bad Value error.
2	X$M_KB_BELL_ PERCENT	KBBellPercent	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 a Bad Value error.
3	X$M_KB_BELL_ PITCH	KBBellPitch	Sets the pitch (specified in Hz) of the bell, if possible. A setting of –1 restores the default. Other negative values generate a Bad Value error.
4	X$M_KB_BELL_ DURATION	KBBellDuration	Sets the duration, specified in milliseconds, of the bell, if possible. A setting of –1 restores the default. Other negative values generate a Bad Value error.
5	X$M_KB_LED	KBLed	Specifies the keyboard LED.

(continued on next page)

Table 10–6 (Cont.) Keyboard Control Value Mask

Bit	VAX Predefined Value	MIT C Predefined Value	Meaning
6	X$M_KB_LED_ MODE	KBLedMode	Specifies the keyboard LED mode. Valid values are Led Mode On and Led Mode Off.
7	X$M_KB_KEY	KBKey	Specifies the auto-repeat key.
8	X$M_KB_AUTO_ REPEAT_MODE	KBAutoRepeatMode	Specifies the auto-repeat mode. Valid values are as follows: Auto Repeat Mode On Auto Repeat Mode Off Auto Repeat Mode Default

If both LED and LED Mode are specified, the state of those LEDs is changed, if this capability is supported. If only Led Mode is specified, the state of all LEDs is changed if possible. At most, 32 LEDs, numbered from 1, are supported. No standard interpretation is defined. A Bad Match error is generated if an LED is specified without an LED mode.

If both Auto Repeat Mode and Key are specified, the auto-repeat mode of that key is changed (according to Auto Repeat Mode On, Auto Repeat Mode Off, or Auto Repeat Mode Default), if possible. If only Auto Repeat Mode is specified, the global auto-repeat mode for the entire keyboard is changed, if possible, and does not affect the per-key settings. A Bad Match error is generated if a key is specified without an auto-repeat mode.

Each key has a mode that determines whether it should auto-repeat, and a default setting for that mode. In addition, there is a global mode that determines whether auto-repeat for all keys should be enabled and a default setting for that mode. When the global mode is on, keys obey their individual auto-repeat modes; when the global mode is off, no keys auto-repeat.

An auto-repeating key generates alternating Key Press and Key Release events. When a key is used as a modifier, it does not auto-repeat, regardless of the auto-repeat setting for the key.

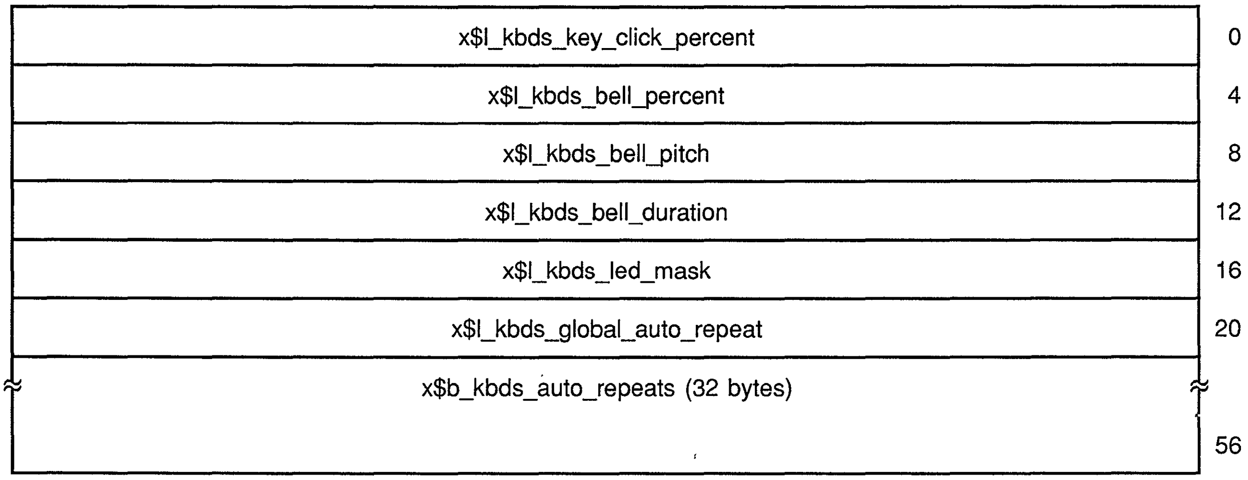
A bell generator that is connected to the console, but is not directly part of the keyboard, is treated as if it were part of the keyboard. The order in which controls are verified and altered is server dependent. If an error is generated, a subset of the controls may have been altered.

10.3 Keyboard State Data Structure

The GET KEYBOARD CONTROL routines returns the current keyboard control values to the keyboard state data structure.

The VAX binding keyboard state data structure is shown in Figure 10–5.

Figure 10–5 Keyboard State Data Structure (VAX Binding)



The members of the VAX binding keyboard state data structure are described in Table 10–7.

Table 10–7 Members of the Keyboard State Data Structure (VAX Binding)

Member Name	Contents
X$L_KBDS_KEY_CLICK_PERCENT	The key click percent value.
X$L_KBDS_BELL_PERCENT	The base volume for the bell.
X$L_KBDS_BELL_PITCH	The bell pitch (specified in Hz).
X$L_KBDS_BELL_DURATION	The bell duration, specified in milliseconds.
X$L_KBDS_LED_MASK	The least significant bit corresponds to LED 1, and each one bit indicates an LED that is lit.

(continued on next page)

Table 10–7 (Cont.) Members of the Keyboard State Data Structure (VAX Binding)

Member Name	Contents
X$L_KBDS_GLOBAL_AUTO_ REPEAT	Global auto-repeat can be set either on or off.
X$B_KBDS_AUTO_REPEATS (32 BYTES)	A bit vector where each one 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 MIT C binding keyboard state data structure is shown in Figure 10–6.

Figure 10–6 Keyboard State Data Structure (MIT C Binding)

```
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;
```

The members of the MIT C binding keyboard state data structure are described in Table 10–8.

Table 10–8 Members of the Keyboard State Data Structure (MIT C Binding)

Member Name	Contents
key_click_percent	The key click percent value.
bell_percent	The base volume for the bell.
bell_pitch	The bell pitch (specified in Hz).
bell_duration	The bell duration, specified in milliseconds.
led_mask	The least significant bit corresponds to LED 1, and each one bit indicates an LED that is lit.
global_auto_repeat	Global auto-repeat can be set either on or off.
auto_repeats[32]	A bit vector where each one 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.

10.4 Compose Data Structure

The compose data structure contains compose-key state information.

The VAX binding compose data structure is shown in Figure 10–7.

Figure 10–7 Compose Data Structure (VAX Binding)

x$a_cmps_compose_ptr	0
x$l_cmps_chars_matched	4

The members of the VAX binding compose data structure are described in Table 10–9.

Table 10–9 Members of the Compose Data Structure (VAX Binding)

Member Name	Contents
X$A_CMPS_COMPOSE_PTR	Compose state table pointer
X$L_CMPS_CHARS_MATCHED	Characters match state

The MIT C binding compose data structure is shown in Figure 10–8.

Figure 10–8 Compose Data Structure (MIT C Binding)

```
typedef struct _XComposeStatus {
    char *compose_ptr;
    int chars_matched;
} XComposeStatus;
```

The members of the MIT C binding compose data structure are described in Table 10–10.

Table 10–10 Members of the Compose Data Structure (MIT C Binding)

Member Name	Contents
compose_ptr	Compose state table pointer
chars_matched	Characters match state

10.5 Modifier Key Map Data Structure

The modifier key map data structure is used to set modifier key codes for keys.

The VAX binding modifier key map data structure is shown in Figure 10–9.

Figure 10–9 Modifier Key Map Data Structure (VAX Binding)

x$l_mdky_max_keypermod	0
x$a_mdky_modifiermap	4

The members of the VAX binding modifier key map data structure are described in Table 10–11.

Table 10–11 Members of the Modifier Key Map Data Structure (VAX Binding)

Member Name	Contents
X$L_MDKY_MAX_KEYPERMOD	The server's maximum number of keys per modifier
X$A_MDKY_MODIFIERMAP	An 8 by X$L_MDKY_MAX_KEYPERMOD array of the modifiers

The MIT C binding modifier key map data structure is shown in Figure 10–10.

Figure 10–10 Modifier Key Map Data Structure (MIT C Binding)

```
typedef struct {
        int max_keypermod;
        KeyCode *modifiermap;
}XModifierKeymap;
```

The members of the MIT C binding modifier key map data structure are described in Table 10–12.

Table 10–12 Members of the Modifier Key Map Data Structure (MIT C Binding)

Member Name	Contents
max_keypermod	The server's maximum number of keys per modifier
modifiermap	An 8 by max_keypermod array of the modifiers

10.6 Window and Session Manager Routines

The following pages describe the Xlib window and session manager routines.

ACTIVATE SCREEN SAVER

Enables the screen saver, even if it is currently disabled.

VAX FORMAT

X$ACTIVATE_SCREEN_SAVER *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT

XActivateScreenSaver *(display)*

argument information

```
XActivateScreenSaver(display)
      Display *display;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

DESCRIPTION

ACTIVATE SCREEN SAVER enables the screen saver, even if it is currently disabled by a previous SET SCREEN SAVER call. When the screen saver is activated on the VMS DECwindows server, the server prevents an image from being burned into the screen.

ADD HOST

Adds a host to the list of hosts that can connect to a display.

VAX FORMAT

X$ADD_HOST *(display, host)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
host	record	x$host_address	read	reference

MIT C FORMAT

XAddHost *(display, host)*

argument information

```
XAddHost(display, host)
      Display *display;
      XHostAddress *host;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

host

A pointer to the network address of the host that you want to add. The network data structure is shown in Section 10.1.

DESCRIPTION

ADD HOST dynamically adds one host to the list of hosts that can connect to the server controlling a display. For this routine to execute successfully, the client issuing the command must reside on the same host as the server or a Bad Access error is generated.

The network data structure is shown in Section 10.1.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

ADD HOSTS

Adds more than one host to the list of hosts that can connect to a display.

VAX FORMAT

X$ADD_HOSTS
(display, hosts, num_hosts)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
hosts	array	uns longword	read	reference
num_hosts	longword	uns longword	read	reference

MIT C FORMAT

XAddHosts
(display, hosts, num_hosts)

argument information

```
XAddHosts(display, hosts, num_hosts)
     Display *display;
     XHostAddress *hosts;
     int num_hosts;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

hosts
A pointer to the network addresses of the hosts that you want to add. The network data structure is shown in Section 10.1.

num_hosts
The number of hosts to be added to the access list.

DESCRIPTION

ADD HOSTS dynamically adds more than one host to the list of hosts that can connect to the server controlling a display. For this routine to execute successfully, the client issuing the command must reside on the same host as the server or a Bad Access error is generated.

The network data structure is shown in Section 10.1.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

ADD TO SAVE SET

Adds a window to the client's save set.

VAX FORMAT X$ADD_TO_SAVE_SET *(display, window_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

MIT C FORMAT XAddToSaveSet *(display, window_id)*

argument information

```
XAddToSaveSet(display, window_id)
      Display *display;
      Window window_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window you want to add to the client's save set. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

DESCRIPTION

ADD TO SAVE SET adds the specified window to the client's save set. The save set is a list of other clients' windows that, if they are inferiors of one of the client's windows, should not be destroyed at connection close and should be remapped if the window is unmapped.

The specified window must have been created by another client or a Bad Match error is generated. The server automatically removes windows from the save set when the windows are destroyed. Refer to the CLOSE DISPLAY routine for information about what happens to the save set when connections are closed.

You can also use the CHANGE SAVE SET routine to add windows to a save set.

Also see the REMOVE FROM SAVE SET routine.

X ERRORS

VAX	MIT C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.
X$C_BAD_MATCH	BadMatch	Possible causes are as follows: • In a graphics request, the root and depth of the graphics context do not match those of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input-only window lacks this attribute.

ALLOW EVENTS

Releases events that were queued because a device was grabbed.

VAX FORMAT

X$ALLOW_EVENTS
(display, event_mode, time)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event_mode	longword	longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

XAllowEvents
(display, event_mode, time)

argument information

```
XAllowEvents(display, event_mode, time)
      Display *display;
      int event_mode;
      Time time;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

event_mode

The events to be released. The predefined values for **event_mode** are as follows:

VAX	MIT C	Description
X$C_ASYNC_ POINTER	AsyncPointer	Allows pointer event processing to continue normally after a pointer has been stopped. If pointer events have been frozen twice by the client on behalf of two separate grabs, both thaw. There is no effect if the pointer is not frozen by the client, but the pointer need not be grabbed by the client.

VAX	MIT C	Description
X$C_SYNC_ POINTER	SyncPointer	Allows pointer event processing to continue normally, after a pointer has been frozen or grabbed, until the next Button Press or Button Release event is reported to the client. At this time, the device may again appear to freeze; however, if the reported event causes the grab to be released, then the device does not freeze. If the pointer has not been frozen or grabbed by the client, there is no effect.
X$C_REPLAY_ POINTER	ReplayPointer	If the pointer is actively grabbed by the client, and is frozen as the result of an event having been sent to the client either from a GRAB BUTTON activation or from a previous call to ALLOW EVENTS with mode Sync Pointer (but not from a call to GRAB POINTER), the pointer grab is released and that event is completely reprocessed. This time, however, the function ignores any passive grabs at or above (towards the root of) the grab window of the grab just released. The request 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.
X$C_ASYNC_ KEYBOARD	AsyncKeyboard	Allows keyboard event processing to continue normally after a keyboard has been frozen. If keyboard events have been frozen twice by the client on behalf of two separate grabs, both thaw. There is no effect if the keyboard is not frozen by the client, but the keyboard need not be grabbed by the client.
X$C_SYNC_ KEYBOARD	SyncKeyboard	If the keyboard is frozen and actively grabbed by the client, keyboard event processing continues as usual until the next Key Press or Key Release event is reported to the client. At this time, the keyboard again appears to freeze. However, if the reported event causes the keyboard grab to be released, the keyboard does not freeze. There is no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

VAX	MIT C	Description
X$C_REPLAY_ KEYBOARD	ReplayKeyboard	If the keyboard is actively grabbed by the client, and is frozen as the result of an event having been sent to the client either from a GRAB KEY activation or from a previous call to ALLOW EVENTS with mode Sync Keyboard (but not from a call to GRAB KEYBOARD), the keyboard grab is released and that event is completely reprocessed. This time, however, the function ignores any passive grabs at or above (towards the root of) the grab window of the grab just released. The request has no effect if the keyboard is not grabbed by the client or if the keyboard is not frozen as the result of an event.
X$C_SYNC_ BOTH	SyncBoth	If the pointer and keyboard are frozen by the client, event processing (for both devices) continues normally until the next Button Press, Button Release, Key Press, or Key Release event is reported to the client for a grabbed device (button event for the pointer, key event for the keyboard), at which time the devices again appear to freeze. However, if the reported event causes the grab to be released, then the devices do not freeze (but if the other device is still grabbed, then a subsequent event for it still causes both devices to freeze.) There is 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, both thaw. A subsequent freeze for Sync Both freezes each device only once.
X$C_ASYNC_ BOTH	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, Async Both thaws both. Async Both has no effect unless both the pointer and keyboard are frozen by the client.

Async Pointer, Sync Pointer, and Replay Pointer have no effect on the processing of keyboard events. Async Keyboard, Sync Keyboard, and Replay Keyboard have no effect on the processing of pointer events. It is possible for both a pointer grab and a keyboard grab (by the same or different clients) to be active simultaneously.

If a device is frozen because of either grab, no event processing is performed for the device. It is possible for a single device to be frozen because of both grabs. In this case, both freezes must be released before events can again be processed.

Other values specified in this argument are not valid.

time

The time when the events are to be released. Either a timestamp, in milliseconds, or the predefined value X$C_CURRENT_TIME or CurrentTime can be specified.

DESCRIPTION

ALLOW EVENTS releases specified queued events after a device is stopped by a previous client action. ALLOW EVENTS does not release any events if the time specified in **time** is earlier than the last grab time, or is later than the current server time.

There can be both a pointer and keyboard grab active, by the same or different clients. If a device is stopped for either grab, no event processing is performed for the device. It is possible for a single device to be stopped because of both grabs. Both freezes must be released before events can again be processed.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

AUTO REPEAT OFF

Turns off keyboard auto-repeat.

VAX FORMAT

X$AUTO_REPEAT_OFF *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT

XAutoRepeatOff *(display)*

argument information

```
XAutoRepeatOff(display)
      Display *display;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

DESCRIPTION

AUTO REPEAT OFF turns off keyboard auto-repeat. Use GET KEYBOARD CONTROL to obtain a list of the keyboard auto-repeat keys.

AUTO REPEAT ON

Turns on keyboard auto-repeat.

VAX FORMAT

X$AUTO_REPEAT_ON *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT

XAutoRepeatOn *(display)*

argument information

```
XAutoRepeatOn(display)
      Display *display;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

DESCRIPTION

AUTO REPEAT ON turns on keyboard auto-repeat. Use GET KEYBOARD CONTROL to obtain a list of the keyboard auto-repeat keys.

BELL

Rings the keyboard bell at the base volume that you specify.

VAX FORMAT X$BELL *(display, percent)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
percent	longword	longword	read	reference

MIT C FORMAT XBell *(display, percent)*

argument information

```
XBell(display, percent)
     Display *display;
     int percent;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

percent

The volume for the bell. Possible values are –100 (off) to 100 (loud) inclusive.

DESCRIPTION

BELL rings the bell on the keyboard of the specified display, if possible. The volume that you specify is relative to the base volume for the keyboard. If the value for the percent argument is not within the range of –100 to 100 inclusive, BELL generates a Bad Value error.

The volume at which the bell is rung when the percent argument is positive is

$$base - [(base*percent)/100] + percent$$

The volume at which the bell is rung when the percent argument is negative is

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

To change the base volume of the bell, use CHANGE KEYBOARD CONTROL.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

CHANGE ACTIVE POINTER GRAB

Changes the dynamic parameters for an active grab.

VAX FORMAT

X$CHANGE_ACTIVE_POINTER_GRAB
(display, event_mask, cursor_id, time)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event_mask	mask_longword	uns longword	read	reference
cursor_id	identifier	uns longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

XChangeActivePointerGrab
(display, event_mask, cursor_id, time)

argument information

```
XChangeActivePointerGrab(display, event_mask, cursor_id, time)
      Display *display;
      unsigned int event_mask;
      Cursor cursor_id;
      Time time;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

event_mask

A bit mask that specifies the pointer events to be reported to the client. The mask can be the inclusive OR of the event mask values listed in Table 10–13.

Table 10–13 Event Mask Description

Bit	VAX Predefined Value	MIT C Predefined Value	Description
2	X$M_BUTTON_PRESS	ButtonPressMask	Pointer button down events wanted
3	X$M_BUTTON_RELEASE	ButtonReleaseMask	Pointer button up events wanted
4	X$M_ENTER_WINDOW	EnterWindowMask	Pointer window entry events wanted
5	X$M_LEAVE_WINDOW	LeaveWindowMask	Pointer window leave events wanted
6	X$M_POINTER_MOTION	PointerMotionMask	Pointer motion events wanted
7	X$M_POINTER_ MOTION_HINT	PointerMotionHintMask	Pointer motion hints wanted
8	X$M_BUTTON1_MOTION	Button1MotionMask	Pointer motion while button 1 down
9	X$M_BUTTON2_MOTION	Button2MotionMask	Pointer motion while button 2 down
10	X$M_BUTTON3_MOTION	Button3MotionMask	Pointer motion while button 3 down
11	X$M_BUTTON4_MOTION	Button4MotionMask	Pointer while button 4 down
12	X$M_BUTTON5_MOTION	Button5MotionMask	Pointer motion while button 5 down
13	X$M_BUTTON_MOTION	ButtonMotionMask	Pointer motion while any button down
14	X$M_KEYMAP_STATE	KeyMapStateMask	Any keyboard state change wanted

cursor_id

The identifier of the cursor to be displayed. If no cursor is to be displayed, use the value None.

time

The time when the events are to be released. Either a timestamp, in milliseconds, or the predefined value X$C_CURRENT_TIME or CurrentTime can be specified.

DESCRIPTION

CHANGE ACTIVE POINTER GRAB changes the specified dynamic parameters if the pointer is actively grabbed by the client. The routine does not finish if the time specified in **time** is earlier than the last pointer grab time or later than the current server time. It has no effect on the passive parameters of GRAB BUTTON.

The interpretation of **event_mask** and **cursor_id** is the same as described in GRAB POINTER.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.

CHANGE KEYBOARD CONTROL

Changes the keyboard settings for the key click volume, base bell volume, LEDs, and auto-repeat keys.

VAX FORMAT

X$CHANGE_KEYBOARD_CONTROL *(display, value_mask, control_values)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
value_mask	mask_longword	uns longword	read	reference
control_values	record	x$keyboard_ control	read	reference

MIT C FORMAT

XChangeKeyboardControl *(display, value_mask, control_values)*

argument information

```
XChangeKeyboardControl(display, value_mask, control_values)
      Display *display;
      unsigned long value_mask;
      XKeyboardControl *control_values;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

value_mask

A bit mask that specifies which keyboard values are to be changed.

Table 10–6 lists the predefined values and descriptions for setting the value mask. This mask is the inclusive OR of the valid control mask bits.

control_values

A keyboard control data structure that specifies the new values for the keyboard control settings. Contains one value for each one bit in **value_mask**.

The keyboard control data structure is shown in Section 10.2.

DESCRIPTION

CHANGE KEYBOARD CONTROL changes the settings for the key click volume, the bell volume, the bell pitch, the bell duration, the LED illuminations, and the auto-repeat keys.

Specify the new values in the keyboard control data structure in **control_values**. Then, set the mask in **value_mask** to specify which values have been changed. Only those values set in the mask are changed.

The keyboard control data structure is shown in Section 10.2.

X ERRORS

VAX	MIT C	Description
X$C_BAD_MATCH	BadMatch	Possible causes are as follows: • In a graphics request, the root and depth of the graphics context do not match those of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input-only window lacks this attribute.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

CHANGE KEYBOARD MAPPING

Specifies key symbols for the selected key codes.

VAX FORMAT

X$CHANGE_KEYBOARD_MAPPING
(display, first_keycode, keysyms_per_keycode, keysyms_ids, num_keycodes)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
first_keycode	longword	uns longword	read	reference
keysyms_per_keycode	longword	longword	read	reference
keysyms_ids	array	uns longword	read	reference
num_keycodes	longword	longword	read	reference

MIT C FORMAT

XChangeKeyboardMapping
(display, first_keycode, keysyms_per_keycode, keysyms_ids, num_keycodes)

argument information

```
XChangeKeyboardMapping(display, first_keycode,
                       keysyms_per_keycode,
                       keysyms_ids, num_keycodes)
      Display *display;
      int first_keycode;
      int keysyms_per_keycode;
      KeySym *keysyms_ids;
      int num_keycodes;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

first_keycode

The first key code to have key symbols. This value must be greater than or equal to the minimum key code value.

keysyms_per_keycode

The number of key symbols to be specified for the key codes. This value must be the same for all key codes specified in a single call to this routine. The number specified should be large enough to accommodate the highest number of key symbols that will be specified with any key code. When there are fewer key symbols for a particular key code, the empty key symbols should be specified as X$C_NO_SYMBOL or NoSymbol.

keysyms_ids

A pointer to a list containing the specified key symbols for the key codes. The total number of key symbols specified must be a multiple of **keysyms_per_keycode**.

VAX only

The list is an array where each element contains a key symbol.

num_keycodes

The number of key codes that are to be changed.

DESCRIPTION

CHANGE KEYBOARD MAPPING defines the symbols for the specified number of key codes. The symbols for key codes outside this range remain unchanged. The number of elements in the key symbols list must be

$$num\_keycodes * keysyms\_per\_keycode$$

The first key code must be greater than or equal to the minimum key code that is supplied at connection setup and stored in the display structure. Otherwise, a Bad Value error is generated. In addition, the following expression must be less than or equal to the maximum key code as returned in the connection setup, or a Bad Value error is generated:

$$first\_keycode + (num\_keycodes/keysyms\_per\_keycode) - 1$$

The key symbol N (counting from zero) for key code K has the following index (counting from zero):

$$(K - first\_keycode) * keysyms\_per\_keycode + N$$

The specified **keysyms_per_keycode** can be chosen arbitrarily by the client to be large enough to hold all desired symbols. A special **keysyms_per_keycode** value of X$C_NO_SYMBOL or NoSymbol should be used to fill in unused elements for individual key codes. X$C_NO_SYMBOL or NoSymbol can appear in nontrailing positions of the effective list for a key code.

CHANGE KEYBOARD MAPPING generates a Mapping Notify event. There is no requirement that the server interpret this mapping; it is stored for reading and writing by clients.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

CHANGE POINTER CONTROL

Controls the interactive feel of the pointing device.

VAX FORMAT

X$CHANGE_POINTER_CONTROL
(display, do_accel, do_threshold, accel_numerator, accel_denominator, threshold)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
do_accel	Boolean	longword	read	reference
do_threshold	Boolean	longword	read	reference
accel_numerator	longword	longword	read	reference
accel_denominator	longword	longword	read	reference
threshold	longword	longword	read	reference

MIT C FORMAT

XChangePointerControl
(display, do_accel, do_threshold, accel_numerator, accel_denominator, threshold)

argument information

```
XChangePointerControl(display, do_accel, do_threshold,
                      accel_numerator, accel_denominator,
                      threshold)
      Display *display;
      Bool do_accel, do_threshold;
      int accel_numerator, accel_denominator;
      int threshold;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

do_accel

The accelerator numerator and denominator values. When **do_accel** is true, the values in **accel_numerator** and **accel_denominator** are used. When **do_accel** is false, the values are not used.

do_threshold

The threshold value. When **do_threshold** is true, the value in **threshold** is used. When **do_threshold** is false, the value is not used.

accel_numerator

The numerator for the acceleration multiplier. The **accel_numerator** and the **accel_denominator** arguments specify the complete acceleration multiplier.

accel_denominator

The denominator for the acceleration multiplier. The **accel_numerator** and the **accel_denominator** arguments specify the complete acceleration multiplier.

threshold

The acceleration threshold, in pixels moved during one movement.

DESCRIPTION

CHANGE POINTER CONTROL defines how the pointing device should move.

An acceleration multiplier is specified as a fraction by **accel_numerator** and **accel_denominator**. For example, if **accel_numerator** is 3 and **accel_denominator** is 1, the acceleration multiplier is 3/1. This value means that the pointer moves three times as fast as normal. The fraction may be rounded by the server arbitrarily.

The threshold value represents the number of pixels the pointer moves in one movement. The acceleration multiplier is applied only when the pointer moves faster than a threshold value specified in **threshold** and applies only to the amount beyond the value in **threshold**. Setting the value to –1 restores the default.

The values of the **do_accel** and **do_threshold** arguments must be true for the pointer values to be set or the parameters are unchanged.

Negative values (other than –1) generate a Bad Value error, as does a zero value for **accel_denominator**.

After the values are set, you can obtain them with GET POINTER CONTROL.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

CHANGE SAVE SET

Adds or removes a window from the client's save set.

VAX FORMAT

X$CHANGE_SAVE_SET
(display, window_id, change_mode)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
change_mode	longword	longword	read	reference

MIT C FORMAT

XChangeSaveSet
(display, window_id, change_mode)

argument information

```
XChangeSaveSet(display, window_id, change_mode)
      Display *display;
      Window window_id;
      int change_mode;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window that you want to add or remove from the save set. The specified window must have been created by some other client. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

change_mode

The predefined values for **change_mode** are as follows:

VAX	MIT C	Description
X$C_SET_ MODE_INSERT	SetModeInsert	Adds the specified windows to the client's save set.
X$C_SET_ MODE_DELETE	SetModeDelete	Removes the specified windows from the client's save set.

DESCRIPTION

CHANGE SAVE SET adds or removes the window from the client's save set depending on the value specified in **change_mode**. The specified window must have been created by some other client. Otherwise, CHANGE SAVE SET generates a Bad Match error. The server automatically removes windows from the server when they are destroyed.

You can also use individual routines to add or remove windows from the save set: use ADD TO SAVE SET to add windows; use REMOVE FROM SAVE SET to remove windows.

X ERRORS

VAX	MIT C	Description
X$C_BAD_MATCH	BadMatch	Possible causes are as follows: • In a graphics request, the root and depth of the graphics context do not match those of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input-only window lacks this attribute.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

DELETE MODIFIERMAP ENTRY

Deletes an entry from a modifier key map structure.

VAX FORMAT

modifier_keys_return = **X$DELETE_MODIFIERMAP_ENTRY** *(modifier_keys, keycode_entry, modifier)*

argument information

Argument	Usage	Data Type	Access	Mechanism
modifier_keys_ return	record	x$modifier_keymap	write	reference
modifier_keys	record	x$modifier_keymap	read	reference
keycode_entry	identifier	uns longword	read	reference
modifier	longword	uns longword	read	reference

MIT C FORMAT

modifier_keys_return = **XDeleteModifiermapEntry** *(modifier_keys, keycode_entry, modifier)*

argument information

```
XModifierKeymap XDeleteModifiermapEntry(modifier_keys,
                                        keycode_entry,
                                        modifier)
  XModifierKeymap *modifier_keys;
  KeyCode  keycode_entry;
  int modifier;
```

RETURNS

modifier_keys_return
A pointer to a modifier keys structure. DELETE MODIFIER MAP ENTRY returns the revised modifier key map structure to this client-supplied structure.

ARGUMENTS

modifier_keys
A pointer to the modifier key map structure from which you want to delete an entry.

keycode_entry
The key code that is to be deleted.

modifier

The modifier for which you want to delete a key symbol. There are eight modifiers in the order (starting from zero) shift, lock, control, mod1, mod2, mod3, mod4, and mod5. You can pass the integer value or one of the following constants:

VAX	MIT C
X$C_SHIFT_MAP_INDEX	Shift
X$C_LOCK_MAP_INDEX	Lock
X$C_CONTROL_MAP_INDEX	Control
X$C_MOD1_MAP_INDEX	Mod1
X$C_MOD2_MAP_INDEX	Mod2
X$C_MOD3_MAP_INDEX	Mod3
X$C_MOD4_MAP_INDEX	Mod4
X$C_MOD5_MAP_INDEX	Mod5

DESCRIPTION

DELETE MODIFIERMAP ENTRY deletes the specified key code from the set that controls the specified modifier. DELETE MODIFIERMAP ENTRY returns the resulting modifier key map structure.

The modifier map is not shrunk if all of the rows in a column are zero and the number of keys per modifier is 1. See the INSERT MODIFIERMAP ENTRY routine for more information.

DISABLE ACCESS CONTROL

Disables access control mode for a display.

VAX FORMAT **X$DISABLE_ACCESS_CONTROL** *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT **XDisableAccessControl** *(display)*

argument information

```
XDisableAccessControl(display)
      Display *display;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

DESCRIPTION

DISABLE ACCESS CONTROL disables the access control list at connection setup. For this routine to execute successfully, the client must reside on the same host as the server.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it

ENABLE ACCESS CONTROL

Enables access control mode for a display.

VAX FORMAT X$ENABLE_ACCESS_CONTROL *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT XEnableAccessControl *(display)*

argument information

```
XEnableAccessControl(display)
      Display *display;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

DESCRIPTION

ENABLE ACCESS CONTROL enables the access control list at connection setup. For this routine to execute successfully, the client must reside on the same host as the server.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it

FORCE SCREEN SAVER

Activates the screen saver in the specified mode.

VAX FORMAT

X$FORCE_SCREEN_SAVER
(display, saver_mode)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
saver_mode	longword	longword	read	reference

MIT C FORMAT

XForceScreenSaver
(display, saver_mode)

argument information

```
XForceScreenSaver(display, saver_mode)
      Display *display;
      int saver_mode;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

saver_mode

How the screen saver is activated. The predefined values for **mode** are as follows:

VAX	MIT C	Description
X$C_SCREEN_ SAVER_ACTIVE	ScreenSaverActive	Activate the screen saver even if it has been disabled
X$C_SCREEN_ SAVER_RESET	ScreenSaverReset	Reset the screen saver to its inital state

DESCRIPTION

FORCE SCREEN SAVER forces the screen saver to one of the two modes specified in **saver_mode**. If you specify the Screen Saver Active mode, the screen saver is activated even if it was previously disabled with a SET SCREEN SAVER call.

If the screen saver is currently enabled and you specify ScreenSaverReset, the screen saver is deactivated. The activation timer is set to its initial state, as if device input had been received.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

FREE MODIFIERMAP

Destroys the specified modifier key map structure.

VAX FORMAT

X$FREE_MODIFIERMAP *(modifier_keys)*

argument information

Argument	Usage	Data Type	Access	Mechanism
modifier_keys	record	x$modifier_keymap	read	reference

MIT C FORMAT

XFreeModifierMap *(modifier_keys)*

argument information

```
XFreeModifierMap(modifier_keys)
          XModifierKeymap *modifier_keys;
```

ARGUMENTS

modifier_keys

A pointer to the modifier key map structure that you want to destroy.

DESCRIPTION

FREE MODIFIERMAP destroys the modifier key map structure that you specify. FREE MODIFIERMAP first frees the modifier map array and then the modifier key map structure.

Use NEW MODIFIERMAP to create one of these structures.

GEOMETRY

Parses window geometry.

VAX FORMAT

mask_return = **X$GEOMETRY**
(display, screen_number, position, default_position, border_width, font_width, font_height, xadd, yadd [,x_coord_return] [,y_coord_return] [,width_return] [,height_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
mask_return	mask_longword	uns longword	write	value
display	identifier	uns longword	read	reference
screen_number	longword	uns longword	read	reference
position	char string	character string	read	descriptor
default_position	char string	character string	read	descriptor
border_width	longword	uns longword	read	reference
font_width	longword	longword	read	reference
font_height	longword	longword	read	reference
xadd	longword	longword	read	reference
yadd	longword	longword	read	reference
x_coord_return	longword	longword	write	reference
y_coord_return	longword	longword	write	reference
width_return	longword	uns longword	write	reference
height_return	longword	uns longword	write	reference

MIT C FORMAT

mask_return = **XGeometry**
(display, screen_number, position, default_position, border_width, font_width, font_height, xadd, yadd, x_coord_return, y_coord_return, width_return, height_return)

argument information

```
int XGeometry(display, screen_id, position, default_position,
              border_width, font_width, font_height, xadd, yadd,
              x_coord_return, y_coord_return, width_return,
              height_return)
      Display *display;
      int screen_id;
      char *position, *default_position;
      unsigned int border_width;
      unsigned int font_width, font_height;
      int xadd, yadd;
      int *x_coord_return, *y_coord_return;
      int *width_return, *height_return;
```

RETURNS

mask_return

A bit mask that specifies which of four values (width, height, x-offset, y-offset) were actually found in the string, and whether the x and y values are negative. Each bit indicates whether the corresponding value was found in the parsed string. For each value found, the corresponding argument is updated; for each value not found, the argument is left unchanged.

Table 10–14 lists the predefined values and their descriptions for the **mask_return**.

Table 10–14 Parse Mask Bits

Bit	VAX	MIT C	Description
1	X$M_NO_VALUE	NoValue	Reserved
2	X$M_X_VALUE	XValue	The x-coordinate of the origin of a window
3	X$M_Y_VALUE	YValue	The y-coordinate of the origin of a window
4	X$M_WIDTH_VALUE	WidthValue	The width of the window in pixels
5	X$M_HEIGHT_VALUE	HeightValue	The height of the window in pixels
6	X$M_ALL_VALUES	AllValues	Indicates if all values are present
7	X$M_X_NEGATIVE_VALUE	XNegativeValue	Indicates if the x-coordinate is negative
8	X$M_Y_NEGATIVE_VALUE	YNegativeValue	Indicates if the y-coordinate is negative

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

screen_number

The identifier of the screen associated with the display.

position

The position string that you want to parse.

VAX only

The **position** argument is the address of a character string descriptor that points to the string.

MIT C only

The **position** argument is a pointer to the null-terminated character string.

default_position

The default geometry specification string that you want to parse.

VAX only

The **default_position** argument is the address of a character string descriptor that points to the string.

MIT C only

The **default_position** argument is a pointer to the null-terminated character string.

border_width

The width, in pixels, of the border associated with the window that you want to parse.

font_width

The width, in pixels, of the font associated with the window that you want to parse.

font_height

The height, in pixels, of the font associated with the window that you want to parse.

xadd

Additional interior padding needed in the window. This coordinate is relative to the origin of the drawable.

yadd

Additional padding needed in the window. This coordinate is relative to the origin of the drawable.

x_coord_return

The x-coordinate to which GEOMETRY returns the x-offset from the specified string. This coordinate is relative to the origin of the drawable.

VAX only

This argument is optional in the VAX binding.

y_coord_return

The y-coordinate to which GEOMETRY returns the y-offset from the specified string. This coordinate is relative to the origin of the drawable.

VAX only

This argument is optional in the VAX binding.

width_return

The width value.

VAX only

This argument is optional in the VAX binding.

height_return

The height value.

VAX only

This argument is optional in the VAX binding.

DESCRIPTION

GEOMETRY determines the placement of a window by using the current format to position windows and returns the x- and y-coordinates and width and height values of the window. Given a fully qualified default geometry specification and an incomplete geometry specification, GEOMETRY returns a bit mask value as defined in the PARSE GEOMETRY routine.

The returned width and height are the width and height that are specified by **default_position**, as overridden by any user-specified position. They are not affected by **font_width**, **font_height**, **xadd**, or **yadd**.

The **x_coord_return** and **y_coord_return** values are computed by using the border width, the screen width and height, any padding specified by **xadd** or **yadd**, and the font width and height multiplied by the width and height from the geometry specifications.

GEOMETRY is usually called by the window manager. Clients typically use the CREATE WINDOW or CREATE SIMPLE WINDOW functions to create a window.

GET DEFAULT

Returns the default property string for the user environment.

VAX FORMAT

status_return = **X$GET_DEFAULT**
(display, program_name, option_name, default_name_return [,default_len_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
program_name	char string	character string	read	descriptor
option_name	char string	character string	read	descriptor
default_name_return	char string	character string	write	descriptor
default_len_return	word	uns word	write	reference

MIT C FORMAT

property_name_return = **XGetDefault**
(display, program_name, option_name)

argument information

```
char *XGetDefault(display, program_name, option_name)
      Display *display;
      char *program_name;
      char *option_name;
```

RETURNS

status_return (VAX only)

Whether the routine completed successfully.

property_name_return (MIT C only)

A pointer to a null-terminated character string that defines the default property string for a user environment. GET DEFAULT returns a null value if the option name you specify in the **option_name** argument does not exist for the program.

The window manager uses the string returned by GET DEFAULT to establish the user environment; other clients should not attempt to modify this string or free the memory that the string occupies.

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

program_name
The name of the program that specifies the default property string for the user environment.

VAX only

The **program_name** argument is the address of a character string descriptor that points to the string.

MIT C only

The **program_name** argument is a pointer to the null-terminated character string.

option_name
The name of the property option for which you want to determine the user environment defaults.

VAX only

The **option_name** argument is the address of a character string descriptor that points to the string.

MIT C only

The **option_name** argument is a pointer to the null-terminated character string.

default_name_return (VAX only)
The address of a character string descriptor that points to the default property string.

default_len_return (VAX only)
The length of the default string minus any padding characters added to fill the string. This argument is optional.

DESCRIPTION

GET DEFAULT returns a pointer to a character string that defines the user default for the window property that you specify. GET DEFAULT checks the resource database, DECW$XDEFAULTS.DAT, for the root window. If the property is defined for the root window, GET DEFAULT uses this definition as the user's default.

If the property is not defined for the root window, GET DEFAULT returns a null value (MIT C binding only). The strings returned by GET DEFAULT are owned by Xlib and should not be modified or freed by clients.

GET INPUT FOCUS

Obtains information about the current input focus.

VAX FORMAT

X$GET_INPUT_FOCUS
(display [,focus_id_return] [,revert_to_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
focus_id_return	identifier	uns longword	write	value
revert_to_return	longword	longword	write	value

MIT C FORMAT

XGetInputFocus
(display, focus_id_return, revert_to_return)

argument information

```
XGetInputFocus(display, focus_id_return, revert_to_return)
      Display *display;
      Window *focus_id_return;
      int *revert_to_return;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

focus_id_return

The identifier of the focus window, Pointer Root, or None. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

VAX only

This argument is optional.

revert_to_return

The current input focus state. One of the following predefined values can be returned:

VAX	MIT C	Description
X$C_REVERT_ TO_PARENT	RevertToParent	The input focus is the parent window, or the closest viewable ancestor.
X$C_REVERT_ TO_POINTER_ ROOT	RevertToPointerRoot	The input focus is Pointer Root. When the focus reverts, the server generates Focus In and Focus Out events, but the last-focus-change time is not affected.
X$C_REVERT_ TO_NONE	RevertToNone	The input focus is None. When the focus reverts, the server generates Focus In and Focus Out events, but the last-focus-change time is not affected.

VAX only

This argument is optional.

DESCRIPTION

GET INPUT FOCUS returns the focus window identifier and the current focus state. The **revert_to_return** argument returns a value that indicates what is done when the focus window becomes unviewable. These values were originally set with SET INPUT FOCUS.

GET KEYBOARD CONTROL

Obtains the current control values for the keyboard.

VAX FORMAT

X$GET_KEYBOARD_CONTROL
(display, state_values_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
state_values_return	record	x$keyboard_state	write	reference

MIT C FORMAT

XGetKeyboardControl
(display, state_values_return)

argument information

```
XGetKeyboardControl(display, state_values_return)
      Display *display;
      XKeyboardState *state_values_return;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

state_values_return
A pointer to a keyboard state structure to which the current keyboard state is returned.

DESCRIPTION

GET KEYBOARD CONTROL returns the settings for the keyboard, including key click volume, bell volume, bell pitch, bell duration, LED illuminations, and the auto-repeat keys.

The keyboard state data structure is shown in Section 10.3.

The CHANGE KEYBOARD CONTROL routine sets the keyboard control values.

GET KEYBOARD MAPPING

Returns the key symbols for one or more than one key code.

VAX FORMAT

status_return = **X$GET_KEYBOARD_MAPPING** *(display, first_keycode_wanted, keycode_count [,keysyms_per_keycode_return] [,keysyms_return] [,buff_size] [,key_buff_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
first_keycode_wanted	identifier	uns longword	read	reference
keycode_count	longword	longword	read	reference
keysyms_per_keycode_ return	longword	longword	write	reference
keysyms_return	address	uns longword	write	reference
buff_size	longword	longword	read	reference
key_buff_return	array	uns longword	write	reference

MIT C FORMAT

keysym_return = **XGetKeyboardMapping** *(display, first_keycode_wanted, keycode_count, keysyms_per_keycode_return)*

argument information

```
KeySym *XGetKeyboardMapping(display, first_keycode_wanted,
                            keycode_count,
                            keysyms_per_keycode_return)
      Display *display;
      KeyCode first_keycode_wanted;
      int keycode_count;
      int *keysyms_per_keycode_return;
```

RETURNS

status_return (VAX only)

Whether the routine completed successfully.

keysym_return (MIT C only)

A pointer to a list of key symbols for the specified key codes.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

first_keycode_wanted

The first key code that will be returned.

keycode_count

The number of key codes that will be returned.

keysyms_per_keycode_return

The number of key symbols per key code. This value is equal for all key codes requested. The number chosen (by the server) is high enough to accommodate the maximum number of key symbols returned with any key code in this request. When there are fewer key symbols within a particular key code, the empty key symbols have the value X$C_NO_SYMBOL or NoSymbol.

keysyms_return (VAX only)

The virtual address of the symbol list. If you specify this optional argument, GET KEYBOARD MAPPING determines the size of the buffer to create for the symbol list. If you specify **keysyms_return**, you do not need to specify **buff_size** and **key_buff_return**.

buff_size (VAX only)

The size of the **key_buff_return** buffer. This argument is optional.

key_buff_return (VAX only)

A pointer to an array in which each element is a key symbol. GET KEYBOARD MAPPING returns the key symbols to this array.

This argument is optional.

DESCRIPTION

GET KEYBOARD MAPPING returns the key symbols for the specified key codes starting with the first key code. The value specified in the **first_keycode_wanted** argument must be equal to or greater than the minimum key code returned in the display structure at connection setup. In addition, the following expression must be less than or equal to the maximum key code returned in the display structure at connection startup:

$$first\_keycode + (keycode\_count) - 1$$

The number of elements in the key list returned by the routine is as follows:

$$keycode\_count * keysyms\_per\_keycode\_return$$

The key symbol N (counting from zero) for key code K has the following index (counting from zero):

$$keysyms\_per\_keycode\_keycode * keysyms\_per\_keycode\_return + N$$

The **keysyms_per_keycode_return** value is chosen arbitrarily by the server to be large enough to report all requested symbols. A special key symbol value of X$C_NO_SYMBOL or NoSymbol is used to fill in unused elements for individual key codes.

Use FREE to free the storage returned by GET KEYBOARD MAPPING.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

GET MODIFIER MAPPING

Returns the key codes for the modifier keys.

VAX FORMAT

X$GET_MODIFIER_MAPPING
(display, modifier_keys_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
modifier_keys_return	record	x$modifier_ keymap	write	reference

MIT C FORMAT

modifier_keys_return = **XGetModifierMapping**
(display)

argument information

```
XModifierKeymap *XGetModifierMapping(display)
      Display *display;
```

RETURNS

modifier_keys_return (MIT C only)

Returns a newly created modifier key map structure that contains the keys being used as modifiers.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

modifier_keys_return (VAX only)

The modifier key map data structure containing the values for the modifier keys. GET MODIFIER MAPPING returns the values in this argument. The modifier key map data structure is shown in Section 10.5.

DESCRIPTION

GET MODIFIER MAPPING returns a newly created modifier key map data structure that contains the keys being used as modifiers, such as the shift and control keys. Clients that use the VAX binding should first call NEW MODIFIER MAP to create a modifier key map data structure.

Clients should use FREE MODIFIER MAP to free the data structure. If only zero values appear in the set for any modifier, that modifier is disabled.

The SET MODIFIER MAPPING routine specifies the key codes for the modifier keys.

The modifier key map data structure is shown in Section 10.5.

GET POINTER CONTROL

Returns the pointer movement values for acceleration and the threshold at which acceleration should be applied.

VAX FORMAT

X$GET_POINTER_CONTROL
(display [,accel_numerator_return] [,accel_denominator_return] [,threshold_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
accel_numerator_return	longword	longword	write	reference
accel_denominator_return	longword	longword	write	reference
threshold_return	longword	longword	write	reference

MIT C FORMAT

XGetPointerControl
(display, accel_numerator_return, accel_denominator_return, threshold_return)

argument information

```
XGetPointerControl(display, accel_numerator_return,
accel_denominator_return, threshold_return)
      Display *display;
      int *accel_numerator_return, *accel_denominator_return;
      int *threshold_return;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

accel_numerator_return

The acceleration numerator. The **accel_numerator_return** and **accel_denominator_return** arguments specify the complete acceleration multiplier.

VAX only

This argument is optional.

accel_denominator_return

The acceleration denominator. The **accel_numerator_return** and **accel_denominator_return** arguments specify the complete acceleration multiplier.

VAX only

This argument is optional.

threshold_return

The acceleration threshold, specified in the number of pixels moved during one movement.

VAX only

This argument is optional.

DESCRIPTION

GET POINTER CONTROL returns the acceleration multiplier and the threshold speed for when to apply the acceleration multiplier. These values were previously set with CHANGE POINTER CONTROL.

An acceleration multiplier is specified as a fraction by **accel_numerator_return** and **accel_denominator_return**. The fraction may be rounded by the server arbitrarily. The threshold value represents the number of pixels the pointer moves in one movement.

GET POINTER MAPPING

Returns the mapping list, which defines which buttons are enabled for the pointer.

VAX FORMAT

num_elements_return = **X$GET_POINTER_MAPPING** *(display, map_return, num_map)*

argument information

Argument	Usage	Data Type	Access	Mechanism
num_elements_return	longword	longword	write	value
display	identifier	uns longword	read	reference
map_return	array	byte	write	reference
num_map	word	uns word	read	reference

MIT C FORMAT

num_elements_return = **XGetPointerMapping** *(display, map_return, num_map)*

argument information

```
int XGetPointerMapping(display, map_return, num_map)
      Display *display;
      unsigned char map_return[];
      int num_map;
```

RETURNS

num_elements_return

The number of elements in **map_return**.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

map_return

A pointer to an array of items that define the mapping list. The array is indexed, starting with one. The index is a core button number. An empty element means the corresponding button is disabled. The length of the array is specified by **num_map**.

num_map

The maximum number of items to be returned in the mapping list. This value specifies the length of the array in **map_return**.

DESCRIPTION

GET POINTER MAPPING returns the mapping list for the pointer. Each item in the mapping list corresponds to a physical button on the pointer. When one of the items has an empty value, the corresponding button is disabled. The nominal mapping for a pointer is the following identity mapping:

```
map[i] = i
```

Use SET POINTER MAPPING to specify the mapping list.

GET SCREEN SAVER

Returns the following values for screen saving: the timeout period, the interval, whether to blank the screen, and whether to allow exposures.

VAX FORMAT

X$GET_SCREEN_SAVER
(display [,timeout_return] [,interval_return] [,prefer_blanking_return] [,allow_exposures_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
timeout_return	longword	longword	write	reference
interval_return	longword	longword	write	reference
prefer_blanking_return	longword	longword	write	reference
allow_exposures_return	longword	longword	write	reference

MIT C FORMAT

XGetScreenSaver
(display, timeout_return, interval_return, prefer_blanking_return, allow_exposures_return)

argument information

```
XGetScreenSaver(display, timeout_return, interval_return,
                prefer_blanking_return, allow_exposures_return)
      Display *display;
      int *timeout_return, *interval_return;
      int *prefer_blanking_return;
      int *allow_exposures_return;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

timeout_return

The time, in seconds, that the screen saver waits before turning on. The time represents the number of seconds when no input from the keyboard or pointing device is received. A value of zero means that the screen saver is disabled.

VAX only

This argument is optional.

interval_return

The time, in seconds, from one screen saver invocation to the next.

VAX only

This argument is optional.

prefer_blanking_return

The screen blanking mode. The predefined values for **prefer_blanking_return** are as follows:

VAX	MIT C	Description
X$C_DONT_PREFER_BLANKING	DontPreferBlanking	Do not blank the screen. If exposures are allowed, or if the screen can be regenerated without sending exposure events to clients, the screen is tiled with the root window background tile. If exposures are not allowed or the exposure events are sent to clients, then the screen does not change.
X$C_PREFER_BLANKING	PreferBlanking	Blank the screen. This can be used only if the hardware supports video blanking.
X$C_DEFAULT_BLANKING	DefaultBlanking	The default is used.

VAX only

This argument is optional.

allow_exposures_return

The current screen saver control values are returned. The predefined values for **allow_exposures_return** are as follows:

VAX	MIT C	Description
X$C_DONT_ALLOW_EXPOSURES	DontAllowExposures	Exposures are not allowed.
X$C_ALLOW_EXPOSURES	AllowExposures	Exposures are allowed.
X$C_DEFAULT_EXPOSURES	DefaultExposures	The default value is used.

VAX only

This argument is optional.

DESCRIPTION

GET SCREEN SAVER returns the screen saver values set by a previous SET SCREEN SAVER call. The following values are returned by the routine:

- The time that elapses from the last device input before the screen saver turns on (**timeout_return**)
- The time between invocations of the screen saver (**interval_return**)
- Whether to blank the screen
- Whether to allow exposures

GRAB BUTTON

Grabs a pointer button.

VAX FORMAT

X$GRAB_BUTTON
(display, button, modifiers, window_id, owner_events, event_mask, pointer_mode, keyboard_mode, confine_id, cursor_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
button	longword	longword	read	reference
modifiers	mask_longword	uns longword	read	reference
window_id	identifier	uns longword	read	reference
owner_events	Boolean	uns longword	read	reference
event_mask	mask_longword	uns longword	read	reference
pointer_mode	longword	longword	read	reference
keyboard_mode	longword	longword	read	reference
confine_id	identifier	uns longword	read	reference
cursor_id	identifier	uns longword	read	reference

MIT C FORMAT

XGrabButton
(display, button, modifiers, window_id, owner_events, event_mask, pointer_mode, keyboard_mode, confine_id, cursor_id)

argument information

```
XGrabButton(display, button, modifiers, window_id, owner_events,
            event_mask, pointer_mode, keyboard_mode,
            confine_id, cursor_id)
      Display *display;
      unsigned int button;
      unsigned int modifiers;
      Window window_id;
      Bool owner_events;
      unsigned int event_mask;
      int pointer_mode, keyboard_mode;
      Window confine_id;
      Cursor cursor_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

button

The button on the pointing device to grab when the specified modifier keys are down. The possible values are as follows:

VAX Predefined Value	MIT C Predefined Value
X$C_BUTTON1	Button1
X$C_BUTTON2	Button2
X$C_BUTTON3	Button3
X$C_BUTTON4	Button4
X$C_BUTTON5	Button5
X$C_ANY_BUTTON	AnyButton

Other buttons pressed are not grabbed. Specify the predefined value X$C_ANY_BUTTON or AnyButton to grab all possible buttons.

modifiers

The set of key masks. This mask is the inclusive OR of the following key mask bits:

Bit	VAX Predefined Value	MIT C Predefined Value
1	X$M_SHIFT	ShiftMask
2	X$M_CAPS_LOCK	LockMask
3	X$M_CONTROL	ControlMask
4	X$M_MOD1	Mod1Mask
5	X$M_MOD2	Mod2Mask
6	X$M_MOD3	Mod3Mask
7	X$M_MOD4	Mod4Mask
8	X$M_MOD5	Mod5Mask

The predefined value X$C_ANY_MODIFIER or AnyModifier can be specified to allow any set of modifiers to be grabbed.

window_id

The identifier of the window in which you want to grab the button. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

owner_events

The owner event flag specifies when a pointer event is reported. When true, all pointer events are reported as usual to the client. When false, pointer events are reported only when they occur on the window specified by **window_id** and only if they are selected by **event_mask**.

event_mask

A bit mask that specifies which pointer events are reported to the client.

Table 10–15 lists the predefined values for the **event_mask**.

Table 10–15 Event Mask Description

Bit	VAX Predefined Value	MIT C Predefined Value	Description
2	X$M_BUTTON_PRESS	ButtonPressMask	Pointer button down events wanted
3	X$M_BUTTON_RELEASE	ButtonReleaseMask	Pointer button up events wanted
4	X$M_ENTER_WINDOW	EnterWindowMask	Pointer window entry events wanted
5	X$M_LEAVE_WINDOW	LeaveWindowMask	Pointer window leave events wanted
6	X$M_POINTER_MOTION	PointerMotionMask	Pointer motion events wanted
7	X$M_POINTER_MOTION_HINT	PointerMotionHintMask	Pointer motion hints wanted
8	X$M_BUTTON1_MOTION	Button1MotionMask	Pointer motion while button 1 down
9	X$M_BUTTON2_MOTION	Button2MotionMask	Pointer motion while button 2 down
10	X$M_BUTTON3_MOTION	Button3MotionMask	Pointer motion while button 3 down
11	X$M_BUTTON4_MOTION	Button4MotionMask	Pointer motion while button 4 down
12	X$M_BUTTON5_MOTION	Button5MotionMask	Pointer motion while button 5 down
13	X$M_BUTTON_MOTION	ButtonMotionMask	Pointer motion while any button down
14	X$M_KEYMAP_STATE	KeyMapStateMask	Any keyboard state change wanted

pointer_mode

A constant that controls further processing of pointer events. Clients can pass one of the following constants:

VAX	MIT C	Description
X$C_GRAB_ MODE_SYNC	GrabModeSync	The pointer event is processed synchronously. The state of the pointer device appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing ALLOW EVENTS request, or until the pointer grab is released. Actual pointer changes are not lost while the keyboard is frozen; they are queued in the server for later processing.
X$C_GRAB_ MODE_ASYNC	GrabModeAsync	The pointer event is processed asynchronously. Pointer event processing is unaffected by activation of the grab.

Other values specified in this argument are not valid.

keyboard_mode

The mode that the keyboard events will use. The predefined values for **keyboard_mode** are as follows:

VAX	MIT C	Description
X$C_GRAB_ MODE_SYNC	GrabModeSync	The keyboard event is processed synchronously. The corresponding device waits until the client that issued the grab request issues a releasing ALLOW EVENTS request. While the device is waiting for ALLOW EVENTS, no further keyboard events are generated by the server. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued in the server for later processing.
X$C_GRAB_ MODE_ASYNC	GrabModeAsync	The keyboard event is processed asynchronously. Keyboard event processing continues normally. If the keyboard is currently frozen by this client, processing of keyboard events is resumed.

confine_id

The identifier of the window in which to confine the pointer. If the pointer can be in any window, specify the predefined value X$C_NONE or None. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

cursor_id

The identifier of the cursor that is displayed during the grab, or the predefined value X$C_NONE or None. The identifier of the cursor was originally returned by CREATE CURSOR.

DESCRIPTION

GRAB BUTTON establishes a passive grab. Consequently, the pointer is actively grabbed when the following conditions 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 grab window contains the pointer.
- The confine-to window (if any) is viewable.
- A passive grab on the same button/key combination does not exist on any ancestor of the grab window.

The pointer is actively grabbed, as for GRAB POINTER. The last-pointer-grab time is set to the time at which the button was pressed (as transmitted in the Button Press event), and the Button Press event is reported.

The interpretation of the remaining arguments is the same as for GRAB POINTER.

The active grab is terminated automatically when the logical state of the pointer has all buttons released (independent of the state of the logical modifier keys).

A modifier of Any Modifier 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 Any Button 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.

GRAB BUTTON overrides all previous passive grabs by the same client on the same button/key combinations on the same window. The request fails and the server generates a Bad Access error if another client has already issued a GRAB BUTTON request with the same button/key combination on the same window. If you specify Any Modifier or Any Button, the request fails and generates a Bad Access error if there is a conflicting grab for any combination. The request has no effect on an active grab.

The logical state of a device (as seen by client applications) might lag behind the physical state if device event processing is frozen.

The UNGRAB BUTTON routine ungrabs a pointing device button.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it
X$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GRAB KEY

Passively grabs one key and specifies the processing of the key event.

VAX FORMAT

X$GRAB_KEY
(display, keycode, modifiers, window_id, owner_events, pointer_mode, keyboard_mode)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
keycode	longword	longword	read	reference
modifiers	longword	uns longword	read	reference
window_id	identifier	uns longword	read	reference
owner_events	Boolean	longword	read	reference
pointer_mode	longword	longword	read	reference
keyboard_mode	longword	longword	read	reference

MIT C FORMAT

XGrabKey
(display, keycode, modifiers, window_id, owner_events, pointer_mode, keyboard_mode)

argument information

```
XGrabKey(display, keycode, modifiers, window_id, owner_events,
         pointer_mode, keyboard_mode)
      Display *display;
      int keycode;
      unsigned int modifiers;
      Window window_id;
      Bool owner_events;
      int pointer_mode, keyboard_mode;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

keycode

The key code that maps to the key to be grabbed. Clients can pass either the key code or the constant X$C_ANY_KEY or AnyKey, which is equivalent to issuing a request for all possible key codes.

modifiers

A bit mask that specifies the set of key masks. This mask is the inclusive OR of the following key mask bits:

Bit	VAX Predefined Value	MIT C Predefined Value
1	X$M_SHIFT	ShiftMask
2	X$M_CAPS_LOCK	LockMask
3	X$M_CONTROL	ControlMask
4	X$M_MOD1	Mod1Mask
5	X$M_MOD2	Mod2Mask
6	X$M_MOD3	Mod3Mask
7	X$M_MOD4	Mod4Mask
8	X$M_MOD5	Mod5Mask

The predefined value X$C_ANY_MODIFIER or AnyModifier can be specified to allow any set of modifiers to be grabbed, including the combination of no modifiers.

window_id

The identifier of the window in which you want to grab the key.

owner_events

The reporting of pointer events. Pointer events are reported normally if this argument is true. If this argument is false, the pointer events are reported with respect to the grab window if selected by the event mask.

pointer_mode

The processing of pointer events. The predefined values for **pointer_mode** are as follows:

VAX	MIT C	Description
X$C_GRAB_MODE_SYNC	GrabModeSync	The pointer event is processed synchronously. The state of the pointer appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing ALLOW EVENTS request, or until the pointer grab is released. Actual pointer changes are not lost while the keyboard is frozen; they are queued in the server for later processing.
X$C_GRAB_MODE_ASYNC	GrabModeAsync	The pointer event is processed asynchronously. Pointer event processing is unaffected by activation of the grab.

Other values specified in this argument are not valid.

keyboard_mode

The processing of keyboard events. The predefined values for **keyboard_mode** are as follows:

VAX	MIT C	Description
X$C_GRAB_ MODE_SYNC	GrabModeSync	The keyboard event is processed synchronously. The corresponding device waits until the client that issued the grab request issues a releasing ALLOW EVENTS request. While the device is waiting for ALLOW EVENTS, no further keyboard events are generated by the server. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued for later processing.
X$C_GRAB_ MODE_ASYNC	GrabModeAsync	The keyboard event is processed asynchronously. Keyboard event processing continues normally. If the keyboard is currently frozen by this client, processing of keyboard events is resumed.

DESCRIPTION

GRAB KEY passively grabs the specified key. Consequently, the keyboard is actively grabbed when 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.
- No other modifier keys are logically down.
- The window specified in **window_id** is, or is an ancestor of, the focus window; or the window 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 grab window.

The last-keyboard-grab time is set to the time at which the key was pressed (as transmitted in the Key Press event) and the Key Press event is reported.

The active keyboard grab is terminated automatically when the logical state of the keyboard has the specified key released, independent of the logical state of the modifier keys.

The logical state of a device, as seen by clients, may lag behind the physical state of the device if device event processing is frozen.

The interpretation of the remaining arguments is the same as for GRAB KEYBOARD.

A modifier of Any Modifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). All specified modifiers do not have to have currently assigned key codes. A key of Any Key is equivalent to issuing the request for all possible key codes. Otherwise, the key must be in the range specified by the minimum and maximim key code in the connection setup.

GRAB KEY overrides all previous passive grabs by the same client on the same key combinations on the same window. GRAB KEY fails if another client has issued a GRAB KEY request with the same key combination on the same window. GRAB KEY also fails when **modifiers** has either the predefined value X$C_ANY_MODIFIER or AnyModifier and there is a conflicting grab for any combination.

A Bad Access error is generated if another client has issued a GRAB KEY with the same key combination on the same window. If you specify Any Modifier or Any Key, the request fails and the server generates a Bad Access error if there is a conflicting grab for any combination.

The UNGRAB KEY routine ungrabs a key.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GRAB KEYBOARD

Actively grabs control of the main keyboard and defines the processing of pointer events.

VAX FORMAT

status_return = **X$GRAB_KEYBOARD** *(display, window_id, owner_events, pointer_mode, keyboard_mode, time)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
owner_events	Boolean	longword	read	reference
pointer_mode	longword	longword	read	reference
keyboard_mode	longword	longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

status_return = **XGrabKeyboard** *(display, window_id, owner_events, pointer_mode, keyboard_mode, time)*

argument information

```
int XGrabKeyboard(display, window_id, owner_events, pointer_mode,
                  keyboard_mode, time)
      Display *display;
      Window window_id;
      Bool owner_events;
      int pointer_mode, keyboard_mode;
      Time time;
```

RETURNS

status_return

Whether the routine completed successfully. GRAB KEYBOARD returns one of the following status values:

VAX	MIT C	Description
X$C_GRAB_ SUCCESS	GrabSuccess	The routine completed successfully.

VAX	MIT C	Description
X$C_ALREADY_ GRABBED	AlreadyGrabbed	The keyboard is actively grabbed by another client.
X$C_GRAB_ FROZEN	GrabFrozen	The keyboard is frozen by an active grab of another client.
X$C_GRAB_ INVALID_TIME	GrabInvalidTime	The time specified in **time** is earlier than the last pointer grab time, or later than the current server time
X$C_GRAB_ NOT_VIEWABLE	GrabNotViewable	The windows specified in **window_id** or **confine_id** are not currently viewable.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window in which you want to grab the keyboard. The window must be viewable in order for this routine to complete successfully. The identifier of the grab window was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW.

owner_events

The owner events flag that specifies whether standard keyboard events are reported. When true, generated key events that are usually reported to the client continue to be reported. If the key event is reported to the window specified in **window_id**, it continues to be reported on that window. When false, key events associated with the window specified in **window_id** are reported.

Both Key Press and Key Release events are always reported.

pointer_mode

The constant that specifies the processing of pointer events. The predefined values for **pointer_mode** are as follows:

VAX	MIT C	Description
X$C_GRAB_ MODE_SYNC	GrabModeSync	The pointer event is processed synchronously. The state of the pointer appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing ALLOW EVENTS request, or until the pointer grab is released. Actual pointer changes are not lost while the keyboard is frozen; they are queued in the server for later processing.
X$C_GRAB_ MODE_ASYNC	GrabModeAsync	The pointer event is processed asynchronously. Pointer event processing is unaffected by activation of the grab.

keyboard_mode

The processing of keyboard events. The predefined values for **keyboard_mode** are as follows:

VAX	MIT C	Description
X$C_GRAB_ MODE_SYNC	GrabModeSync	The keyboard event is processed synchronously. The corresponding device waits until the client that issued the grab request issues a releasing ALLOW EVENTS request. While the device is waiting for ALLOW EVENTS, no further keyboard events are generated by the server. Actual keyboard changes are not lost while the keyboard is frozen; they are queued in the server for later processing.
X$C_GRAB_ MODE_ASYNC	GrabModeAsync	The keyboard event is processed asynchronously. Keyboard event processing continues normally. If the keyboard is currently frozen by this client, processing of keyboard events is resumed.

time

The time when the events are to be released. Either a timestamp, in milliseconds, or the predefined value Current Time can be specified.

DESCRIPTION

GRAB KEYBOARD grabs control of the keyboard and any further key events are reported only to the client that issued this call. This routine generates Focus In and Focus Out events.

The window specified by **window_id** must be viewable when this routine is called.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GRAB POINTER

Actively grabs the specified pointer.

VAX FORMAT

status_return = **X$GRAB_POINTER** *(display, window_id, owner_events, event_mask, pointer_mode, keyboard_mode, confine_id, cursor_id, time)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
owner_events	Boolean	longword	read	reference
event_mask	mask_longword	uns longword	read	reference
pointer_mode	longword	longword	read	reference
keyboard_mode	longword	longword	read	reference
confine_id	identifier	uns longword	read	reference
cursor_id	identifier	uns longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

status_return = **XGrabPointer** *(display, window_id, owner_events, event_mask, pointer_mode, keyboard_mode, confine_id, cursor_id, time)*

argument information

```
int XGrabPointer(display, window_id, owner_events, event_mask,
                 pointer_mode, keyboard_mode, confine_id,
                 cursor_id, time)
      Display *display;
      Window window_id;
      Bool owner_events;
      unsigned int event_mask;
      int pointer_mode, keyboard_mode;
      Window confine_id;
      Cursor cursor_id;
      Time time;
```

RETURNS

status_return

GRAB POINTER returns the following status:

VAX	MIT C	Description
X$C_GRAB_ SUCCESS	GrabSuccess	The routine completed successfully.
X$C_ALREADY_ GRABBED	AlreadyGrabbed	The pointer is actively grabbed by another client.
X$C_GRAB_ FROZEN	GrabFrozen	The pointer is frozen by an active grab of another client.
X$C_GRAB_ INVALID_TIME	GrabInvalidTime	The time specified in **time** is earlier than the last pointer grab time, or later than the current server time. Otherwise, the last pointer grab time is set to the specified time. CurrentTime is replaced by the current server time.
X$C_GRAB_ NOT_VIEWABLE	GrabNotViewable	The windows specified in **window_id** or **confine_id** are not currently viewable, or **confine_id** window lies completely outside the boundaries of the root window.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The window identifier of the window to which events are reported while it is grabbed.

owner_events

The owner event flag that specifies when a pointer event is reported. When true, all pointer events are reported as usual to the client. When false, pointer events are reported only when they occur on the window specified by **window_id** and only if they are selected by **event_mask**.

event_mask

A bit mask that specifies the events.

Table 10–16 lists the predefined values for the event mask.

Table 10–16 Event Mask Description

Bit	VAX Predefined Value	MIT C Predefined Value	Description
2	X$M_BUTTON_PRESS	ButtonPressMask	Pointer button down events wanted
3	X$M_BUTTON_RELEASE	ButtonReleaseMask	Pointer button up events wanted

(continued on next page)

Table 10–16 (Cont.) Event Mask Description

Bit	VAX Predefined Value	MIT C Predefined Value	Description
4	X$M_ENTER_WINDOW	EnterWindowMask	Pointer window entry events wanted
5	X$M_LEAVE_WINDOW	LeaveWindowMask	Pointer window leave events wanted
6	X$M_POINTER_MOTION	PointerMotionMask	Pointer motion events wanted
7	X$M_POINTER_MOTION_ HINT	PointerMotionHintMask	Pointer motion hints wanted
8	X$M_BUTTON1_MOTION	Button1MotionMask	Pointer motion while button 1 down
9	X$M_BUTTON2_MOTION	Button2MotionMask	Pointer motion while button 2 down
10	X$M_BUTTON3_MOTION	Button3MotionMask	Pointer motion while button 3 down
11	X$M_BUTTON4_MOTION	Button4MotionMask	Pointer motion while button 4 down
12	X$M_BUTTON5_MOTION	Button5MotionMask	Pointer motion while button 5 down
13	X$M_BUTTON_MOTION	ButtonMotionMask	Pointer motion while any button down
14	X$M_KEYMAP_STATE	KeyMapStateMask	Any keyboard state change wanted

pointer_mode

The mode that the pointer events will use. The predefined values for **pointer_mode** are as follows:

VAX	MIT C	Description
X$C_GRAB_ MODE_SYNC	GrabModeSync	The pointer event is processed synchronously. The state of the pointer appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing ALLOW EVENTS request, or until the pointer grab is released. Actual pointer changes are not lost while the keyboard is frozen; they are queued in the server for later processing.
X$C_GRAB_ MODE_ASYNC	GrabModeAsync	The pointer event is processed asynchronously. Pointer event processing is unaffected by activation of the grab.

Other values specified in this argument are not valid.

keyboard_mode

The processing of keyboard events. The predefined values for **keyboard_mode** are as follows:

VAX	MIT C	Description
X$C_GRAB_MODE_SYNC	GrabModeSync	The keyboard event is processed synchronously. The corresponding device waits until the client that issued the grab request issues a releasing ALLOW EVENTS request. While the device is waiting for ALLOW EVENTS, no further keyboard events are generated by the server. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued in the server for later processing.
X$C_GRAB_MODE_ASYNC	GrabModeAsync	The keyboard event is processed asynchronously. Keyboard event processing continues normally. If the keyboard is currently frozen by this client, processing of keyboard events is resumed.

Other values specified in this argument are not valid.

confine_id

The identifier of the window that the pointer will be confined to. If there is an attempt to move the pointer out of the window, the pointer will not move beyond the window. If the pointer can be moved to any window, use the predefined value X$C_NONE or None.

The window must be viewable when this routine is called, or the routine will not succeed.

cursor_id

The identifier of the cursor to be displayed during the grab, or the predefined value X$C_NONE or None.

time

The time when the events are to be released. Either a timestamp, in milliseconds, or the predefined value X$C_CURRENT_TIME or CurrentTime can be specified.

DESCRIPTION

GRAB POINTER actively grabs control of the pointer when the conditions specified in the routine have been met and a pointer input event is generated. Further pointer events are reported only to the grabbing client. GRAB POINTER overrides any active pointer grab by this client.

If a pointer cursor is specified, it is displayed regardless of what window the pointer is in. If no pointer cursor is specified, the normal pointer cursor for that window is displayed when the pointer is in **window_id** or one of its child windows. Otherwise, the pointer cursor for **window_id** is displayed. If the pointer is not initially in the window specified by **confine_id**, it is automatically moved to the closest edge just before the grab activates. Standard enter/leave events are generated. If the window specified by **confine_id** is subsequently reconfigured, the pointer is moved automatically as necessary to keep it contained in the window.

The windows specified by **window_id** and **confine_id** do not require any relationship. For example, they could have different parents and be in completely different window hierarchies. However, for the routine to succeed, they both must be viewable at the time GRAB POINTER is called. If they are not viewable, UNGRAB POINTER is automatically called to release the pointer grab.

GRAB POINTER generates Enter Notify and Leave Notify events. The window identifiers were originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. The cursor identifier was originally returned by CREATE CURSOR.

The UNGRAB POINTER routine releases an active pointer grab.

X ERRORS

VAX	MIT C	Description
X$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GRAB SERVER

Takes exclusive possession of the server associated with the display.

VAX FORMAT

X$GRAB_SERVER *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT

XGrabServer *(display)*

argument information

```
XGrabServer(display)
     Display *display;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

DESCRIPTION

GRAB SERVER takes exclusive possession of the server for the display that you specify. No requests are processed, including requests to close connections, while the server is grabbed. A client automatically ungrabs the server when it closes its connection to that server.

GRAB SERVER can be useful for window managers or clients that want to preserve bits on the screen while temporarily suspending processing on other connections. Clients should not grab the server any more than is absolutely necessary.

The UNGRAB SERVER routine releases an active server grab.

INSERT MODIFIERMAP ENTRY

Adds a new entry to the modifier key map structure.

VAX FORMAT

status_return = **X$INSERT_MODIFIERMAP_ENTRY** *(modifier_keys, keycode_entry, modifier, modifier_keys_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
modifier_keys	record	x$modifier_keymap	read	reference
keycode_entry	identifier	uns longword	read	reference
modifier	longword	uns longword	read	reference
modifier_keys_ return	record	x$modifier_keymap	write	reference

MIT C FORMAT

modifier_keys_return = **XInsertModifierKeymapEntry** *(modifier_keys, keycode_entry, modifier)*

argument information

```
XModifierKeymap XInsertModifiermapEntry(modifier_keys,
                                        keycode_entry,
                                        modifier)
  XModifierKeymap *modifier_keys;
  KeyCode  keycode_entry;
  int modifier;
```

RETURNS

status_return (VAX only)

Whether the routine completed successfully.

modifier_keys_return (MIT C only)

The revised modifier key map structure.

ARGUMENTS

modifier_keys

A pointer to the modifier key map structure to which you want to add an entry.

keycode_entry

The key code that is to be added.

modifier

The modifier for which you want to add a key symbol. There are eight modifiers in the order (starting from zero) shift, lock, control, mod1, mod2, mod3, mod4, and mod5. You can pass the integer value or one of the following constants:

VAX	MIT C
X$C_SHIFT_MAP_INDEX	Shift
X$C_LOCK_MAP_INDEX	Lock
X$C_CONTROL_MAP_INDEX	Control
X$C_MOD1_MAP_INDEX	Mod1
X$C_MOD2_MAP_INDEX	Mod2
X$C_MOD3_MAP_INDEX	Mod3
X$C_MOD4_MAP_INDEX	Mod4
X$C_MOD5_MAP_INDEX	Mod5

modifier_keys_return (VAX only)

INSERT MODIFIER MAP ENTRY returns the revised modifier key map data structure.

DESCRIPTION

INSERT MODIFIERMAP ENTRY adds the specified key code to the set that controls the specified modifier. INSERT MODIFIERMAP ENTRY returns the resulting modifier key map data structure (expanded as needed). INSERT MODIFIERMAP ENTRY observes the following rules:

- If the key code to be added is already in the array, the modifier map is not changed.
- If the row (where **modifier** equals row) in which it is to be placed contains a zero, the key code is added to that spot.
- If there is a nonzero entry in the row in which the key code is to be placed, another column is added.

For example, Table 10–17 shows the result of adding a key code with a value of 72 and a modifier value of 5 (mod3) to a modifier map. The number of keys per modifier is 1.

Table 10–17 Adding a Key Code to a Zero Value

Current Map	New Map
65	65
70	70
68	68
0	0
0	0
0	72
0	0
0	0

Table 10–18 shows the result of adding a key code with a value of 80 and a modifier value of 2 (control) to a modifier map. The number of keys per modifier is 2.

Table 10–18 Adding a Key Code to a Nonzero Value

Current Map	New Column
65	0
70	0
68	80
0	0
0	0
72	0
0	0
0	0

INSTALL COLORMAP

Overwrites the current color map with the entries from the specified color map.

VAX FORMAT

X$INSTALL_COLORMAP *(display, colormap_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference

MIT C FORMAT

XInstallColormap *(display, colormap_id)*

argument information

```
XInstallColormap(display, colormap_id)
      Display *display;
      Colormap colormap_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map to be installed.

DESCRIPTION

INSTALL COLORMAP installs the specified color map. The identifier of the color map was originally returned by CREATE COLORMAP or DEFAULT COLORMAP. All windows associated with the color map immediately display the colors specified in the new color map. If the color map being installed has not previously been installed, a Colormap Notify event is sent to each window using this color map.

If this installation replaces a different, previously installed color map, the previous map is uninstalled. Then a Colormap Notify event is sent to each window using the previous color map. Windows that are associated with the previous color map may display incorrect colors when the new color map is installed.

The server maintains a subset of the installed color maps in an ordered list called the **required list**. The maximum length of the required list is limited to the length of the maps installed for the screen by OPEN DISPLAY.

The server maintains the required list as follows:

- If you pass a color map resource identifier to **colormap_id**, it adds the color map to the top of the list. The server truncates a color map at the bottom of the list if the list would exceed its maximum length.
- If you pass a color map resource identifier to the **colormap_id** argument of UNINSTALL COLORMAP, and that color map is in the required list, the color map is removed from the list. A color map is not added to the required list when it is installed implicitly by the server; the server cannot implicitly uninstall a color map that is in the required list.

Initially, only the default color map for a screen is installed, but it is not in the required list.

KEYCODE TO KEYSYM

Converts the key code that you specify to a defined key symbol.

VAX FORMAT

keysym_return = **X$KEYCODE_TO_KEYSYM** *(display, keycode, index)*

argument information

Argument	Usage	Data Type	Access	Mechanism
keysym_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
keycode	word	uns word	read	reference
index	longword	longword	read	reference

MIT C FORMAT

keysym_return = **XKeycodeToKeysym** *(display, keycode, index)*

argument information

```
KeySym XKeycodeToKeysym(display, keycode, index)
      Display *display;
      KeyCode keycode;
      int index;
```

RETURNS

keysym_return

The key symbol defined for the specified key code.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

keycode

The key code that you want to convert to a key symbol.

index

The element of the key-code vector.

DESCRIPTION

KEYCODE TO KEYSYM uses internal Xlib tables to return the key symbol defined for the specified key code and the element of the key-code vector. If no key symbol is defined, KEYCODE TO KEYSYM returns X$C_NO_SYMBOL or NoSymbol.

KEYSYM TO KEYCODE

Converts the key symbol that you specify to a defined key code.

VAX FORMAT

keycode_return = **X$KEYSYM_TO_KEYCODE** *(display, keysym_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
keycode_return	word	uns word	write	value
display	identifier	uns longword	read	reference
keysym_id	identifier	uns longword	read	reference

MIT C FORMAT

keycode_return = **XKeysymToKeycode** *(display, keysym_id)*

argument information

```
KeyCode XKeysymToKeycode(display, keysym_id)
      Display *display;
      KeySym keysym_id;
```

RETURNS

keycode_return

The key code defined for the specified key symbol.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

keysym_id

The key symbol for which to search.

DESCRIPTION

KEYSYM TO KEYCODE converts the key symbol that you specify into the appropriate key code. If the specified key symbol is not defined for any key code, KEYSYM TO KEYCODE returns zero.

KEYSYM TO STRING

Converts the key-symbol identifier that you specify to the name of the key symbol.

VAX FORMAT

status_return = **X$KEYSYM_TO_STRING** *(keysym_id, keysym_name_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond value	uns longword	write	value
keysym_id	identifier	uns longword	read	reference
keysym_name_return	char string	character string	write	descriptor

MIT C FORMAT

char_return = **XKeysymToString** *(keysym_id)*

argument information

```
char *XKeysymToString(keysym_id)
      KeySym keysym_id;
```

RETURNS

status_return (VAX only)

Whether the routine completed successfully.

char_return (MIT C only)

The name of the key symbol.

ARGUMENTS

keysym_id

The key symbol that is to be converted.

keysym_name_return (VAX only)

The name of the key-symbol string.

DESCRIPTION

KEYSYM TO STRING converts the key-symbol identifier that you specify into the appropriate key symbol name. The returned string is in a static area and must not be modified. If the specified key symbol is not defined, KEYSYM TO STRING returns a null value.

KILL CLIENT

Disconnects a client associated with the specified resource.

VAX FORMAT

X$KILL_CLIENT *(display, resource)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
resource	longword	longword	read	reference

MIT C FORMAT

XKillClient *(display, resource)*

argument information

```
XKillClient(display, resource)
      Display *display;
      XID resource;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

resource

The identifier of the resource associated with the client to be disconnected. The predefined value X$C_ALL_TEMPORARY or AllTemporary can be specified in place of an identifier. When All Temporary is specified, all resources associated with clients that disconnected in Retain Temporary mode are destroyed.

DESCRIPTION

KILL CLIENT disconnects the client associated with the resource specified in **resource**. If the client has already disconnected in a Retain Permanent or Retain Temporary mode, then all of the client's resources are freed.

If you specify the predefined value All Temporary for **resource**, then all resources for all clients that have disconnected in Retain Temporary mode are destroyed.

For more information about close-down modes, see the SET CLOSE DOWN MODE and CLOSE DISPLAY routines.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

LIST HOSTS

Returns the list of hosts that can access a display. LIST HOSTS also returns a pointer to the number of hosts in the access control list and the state of the list when the connection was made.

VAX FORMAT

status_return = **X$LIST_HOSTS**
(display, num_hosts_return, state_return [,hosts_return] [,hosts_size] [,hosts_buff_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
num_hosts_return	longword	longword	write	reference
state_return	longword	longword	write	reference
hosts_return	address	uns longword	write	reference
hosts_size	longword	longword	read	reference
hosts_buff_return	array	uns longword	write	reference

MIT C FORMAT

hostaddress_return = **XListHosts**
(display, num_hosts_return, state_return)

argument information

```
XHostAddress *XListHosts(display, num_hosts_return, state_return)
      Display *display;
      int *num_hosts_return;
      Bool *state_return;
```

RETURNS

status_return (VAX only)

Whether the routine completed successfully. Possible status values returned by the VAX binding are as follows:

Value	Description
X$_ERRORREPLY	An error was received from the server.
X$_NOHOSTS	There are no hosts available to make connections.

Value	Description
X$_TRUNCATED	The user buffer specified in **time_buff_return** was not large enough.
SS$_NORMAL	The routine completed successfully.

hostaddress_return (MIT C only)

A pointer to the current access control list defined in the network data structure. The network data structure is shown in Section 10.1.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

num_hosts_return

A pointer to the number of hosts currently in the access control list. This number includes the host structures that were allocated by this routine. The memory occupied by **num_hosts_return** should be freed by using the FREE routine when it is no longer needed.

state_return

The access control state. Access control is enabled if **state_return** is true, and access control is disabled if **state_return** is false.

hosts_return (VAX only)

The virtual address of the hosts buffer, which contains the current access control list, is returned. This argument is optional. If you specify this argument, LIST HOSTS determines the size of the hosts buffer to create. If you specify **hosts_return**, you do not need to specify **hosts_size** and **hosts_buff_return**.

hosts_size (VAX only)

The size of the hosts buffer to which LIST HOSTS returns the list of hosts that can access a display. This argument is optional.

hosts_buff_return (VAX only)

A pointer to an array of addresses in which each element is the address of a host. The length of the array is specified by **num_hosts_return**. This argument is optional.

DESCRIPTION

LIST HOSTS returns the current access control list as well as whether the use of the list at connection setup was enabled or disabled. LIST HOSTS allows a client to find out which hosts can connect to a display and returns a pointer to which the number of hosts currently in the access control list is returned. Clients should use FREE to free the memory used by this routine.

The network data structure is shown in Section 10.1.

LIST INSTALLED COLORMAPS

Returns a color map identifier of each installed color map for a window.

VAX FORMAT

status_return = **X$LIST_INSTALLED_COLORMAPS** *(display, window_id, num_colormaps_return [,colormaps_return] [,colormaps_size] [,colormaps_buff_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
num_colormaps_ return	longword	longword	write	reference
colormaps_return	address	uns longword	write	reference
colormaps_size	longword	longword	read	reference
colormaps_buff_return	array	uns longword	write	reference

MIT C FORMAT

colormap_return = **XListInstalledColormaps** *(display, window_id, num_colormaps_return)*

argument information

```
Colormap *XListInstalledColormaps(display, window_id,
                                  num_colormaps_return)
      Display *display;
      Window window_id;
      int *num_colormaps_return;
```

RETURNS

status_return (VAX only)

Whether the routine completed successfully. Possible status values returned by the VAX binding are as follows:

Value	Description
0	None
X$_TRUNCATED	The user buffer specified in **time_buff_return** was not large enough.
SS$_NORMAL	The routine completed successfully.

colormap_return (MIT C only)

A pointer to the list of color map identifiers for the screen of the specified window.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window for which to obtain the color map list. The window identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

num_colormaps_return

The number of currently installed color maps.

colormaps_return (VAX only)

The address of the buffer where the list of color map identifiers is returned. This argument is optional. If you specify this argument, LIST INSTALLED COLORMAPS determines the size of the buffer to create. If you specify **colormaps_return**, you do not need to specify **colormaps_size** and **colormaps_buff_return**.

colormaps_size (VAX only)

The size of the color map buffer.

colormaps_buff_return (VAX only)

A pointer to a buffer where the list of color map identifiers is returned.

DESCRIPTION

LIST INSTALLED COLORMAPS returns a list of the currently installed color maps for the screen of the specified window. The order in which the color maps appear in the list is not significant, and the required color map list is not explicitly indicated.

When you no longer need the list, use FREE (X$FREE or XFree) to deallocate the storage.

X ERRORS

VAX	MIT C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

LOOKUP KEYSYM

Returns the key symbol from the list that corresponds to the key code in the event that you specify.

VAX FORMAT

keysym_id_return = **X$LOOKUP_KEYSYM** *(key_event, index)*

argument information

Argument	Usage	Data Type	Access	Mechanism
keysym_id_return	identifier	uns longword	write	value
key_event	record	x$key_event	read	reference
index	longword	longword	read	reference

MIT C FORMAT

keysym_id_return = **XLookupKeysym** *(key_event, index)*

argument information

```
KeySym XLookupKeysym(key_event, index)
      XKeyEvent *key_event;
      int index;
```

RETURNS

keysym_id_return

The key symbol identifier returned from the list that corresponds to the *key code* member in the Key Pressed or Key Released event data structures.

ARGUMENTS

key_event

A pointer to the Key Event structure that is to be used. The event is either a Key Pressed or Key Released event.

index

The element of the key-code vector.

DESCRIPTION

LOOKUP KEYSYM uses a given keyboard event and the index that you specify to return the key-symbol identifier. If no key symbol is defined for the key code of the event, LOOKUP KEYSYM returns X$C_NO_SYMBOL or NoSymbol.

LOOKUP STRING

Maps a key event to an ISO-Latin1 string.

VAX FORMAT

buflen_return = **X$LOOKUP_STRING** *(key_event, buff_return, num_bytes, keysym_id_return, compose_status_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
buflen_return	longword	longword	write	value
key_event	record	x$key_event	read	reference
buff_return	address	longword	write	reference
num_bytes	longword	uns longword	read	reference
keysym_id_return	identifier	uns longword	write	reference
compose_status_return	record	x$compose_status	write	reference

MIT C FORMAT

buflen_return = **XLookupString** *(key_event, buff_return, num_bytes, keysym_id_return, compose_status_return)*

argument information

```
int XLookupString(key_event, buff_return, num_bytes,
                  keysym_id_return, compose_status_return)
      XKeyEvent *key_event;
      char *buff_return;
      int num_bytes;
      KeySym *keysym_id_return;
      XComposeStatus *compose_status_return;
```

RETURNS

buflen_return

The length of the string stored in the buffer.

ARGUMENTS

key_event

The Key Pressed or Key Released event that you want to map to an ISO-Latin1 string.

buff_return

The translated characters are returned to this buffer. You pass in a buffer to which LOOKUP STRING returns the translated characters.

num_bytes

The length of the buffer. No more than **num_bytes** of translation are returned.

keysym_id_return

The key symbol computed from the event, if the key symbol is not a null value.

compose_status_return

A pointer to the compose status data structure used to track compose processing information. This should be declared by the caller as a global structure to allow consistent compose processing across an application.

The argument can be null if the caller does not want compose processing information.

The compose status data structure is shown in Section 10.4.

DESCRIPTION

LOOKUP STRING is a convenience routine that can be used to map a key event to an ISO-Latinl string, using the modified bits in the key event to handle the Shift, Lock, and Control keys. It returns the translated string to the user's buffer. It also detects any rebound key symbols (see REBIND KEYSYM) and returns the specified bytes.

LOOKUP STRING returns, as its value, the length of the string stored in the tag buffer. If the lock modifier has a Caps Lock key associated with it, LOOKUP STRING interprets the lock modifier.

The compose status structure records the compose key state, which is private to Xlib. The compose key state is preserved to implement compose key processing. The compose status data structure is shown in Section 10.4.

NEW MODIFIER MAP

Creates a new modifier key map data structure.

VAX FORMAT

status_return = **X$NEW_MODIFIERMAP** *(max_keys_per_mod, mkeymap_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond value	uns longword	write	value
max_keys_per_mod	longword	longword	read	reference
mkeymap_return	record	x$modifier_ keymap	write	reference

MIT C FORMAT

xmodifierkeymap_return = **XNewModifiermap** *(max_keys_per_mod)*

argument information

```
XModifierKeymap XNewModifiermap(max_keys_per_mod)
        int max_keys_per_mod;
```

RETURNS

status_return (VAX only)

Whether the routine completed successfully.

xmodifierkeymap_return (MIT C only)

The new modifier key map data structure.

ARGUMENTS

max_keys_per_mod

The maximum number of key codes assigned to any modifier in the map.

mkeymap_return (VAX only)

The new modifier key map structure.

DESCRIPTION

NEW MODIFIERMAP returns a modifier key map data structure. The modifier key map data structure is shown in Section 10.5.

PARSE COLOR

Provides the red, green, and blue color values for a named color.

VAX FORMAT

status_return = **X$PARSE_COLOR**
(display, colormap_id, color_name, screen_def_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
color_name	char string	character string	read	descriptor
screen_def_return	record	x$color	write	reference

MIT C FORMAT

status_return = **XParseColor**
(display, colormap_id, color_name, screen_def_return)

argument information

```
Status XParseColor(display, colormap_id, color_name,
                   screen_def_return)
        Display *display;
        Colormap colormap_id;
        char *color_name;
        XColor *screen_def_return;
```

RETURNS

status_return

Whether the routine completed successfully. When the value of **status** is zero, the routine did not complete successfully. When the value of **status** is nonzero, the routine completed successfully.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map containing the requested color definition.

color_name

The name of the color. The string can be either a color name string or a numeric specification. If you use a text string, the name must be supported by the color database maintained by the server. See the SYS$MANAGER:DECW$RGB.COM file for more information. Case is not significant. The numeric specification allows you to define the color according to red, green, and blue values in four levels of detail:

- #RGB, where you use one number to define each color component
- #RRGGBB, where you use two numbers to define each color component
- #RRRGGGBBB, where you use three numbers to define each color component
- #RRRRGGGGBBBB, where you use four numbers to define each color component

Each number represents a single hexadecimal digit. When you use fewer than four digits to represent a value, each digit represents the most significant bit of the value.

For example, suppose you have the following color definition:

Red—1234
Green—1234
Blue—1234

The first level of representation is single numbers: #111. The corresponding value is #100010001000.

The second level of representation is two numbers: #121212. The corresponding value is #120012001200.

The third level of representation is three numbers: #123123123. The corresponding value is #123012301230.

The fourth level of representation is four numbers: #123412341234. The corresponding value is #123412341234.

This routine fails if you do not follow these numeric formats. For example, if you specify a number sign (#) as the initial character but the rest of the string is incorrect, or if you do not specify a number sign even though the rest of the string is correct, the routine fails.

VAX only

The **color_name** argument is the address of a character string descriptor that points to the string.

MIT C only

The **color_name** argument is a pointer to the null-terminated character string.

screen_def_return

The exact color value used in the color map. The flags member of **screen_def_return** is set (all three flags are set).

DESCRIPTION

PARSE COLOR reads a string specification of a color, usually from a command line or the GET DEFAULT routine. Then it returns the specific color values for that color in the color definition data structure. After these values are obtained, you can then use the color definition data structure with the ALLOC COLOR routine to obtain a color map entry, or STORE COLOR to set the color in a color map entry.

You can use the numeric specification if you want to be more precise about the color you request. Depending on the level of detail you want, you can specify the red, green, and blue values in four levels of detail in the **color_name** argument. The color definition for a numeric specification is also returned in the color definition data structure.

The color definition data structure for the VAX binding is shown in Figure 10–11.

Figure 10–11 Color Definition Data Structure (VAX Binding)

<table>
<tr><td colspan="4">x$l_colr_pixel</td><td>0</td></tr>
<tr><td colspan="2">x$w_colr_green</td><td colspan="2">x$w_colr_red</td><td>4</td></tr>
<tr><td>x$b_colr_pad</td><td>x$b_colr_flags</td><td colspan="2">x$w_colr_blue</td><td>8</td></tr>
</table>

The members of the VAX binding color definition data structure are described in Table 10–19.

Table 10–19 Members of the Color Definition Data Structure (VAX Binding)

Member Name	Contents
X$L_COLR_PIXEL	Defines a pixel value.
X$W_COLR_RED	Defines the red value of the pixel.[1]
X$W_COLR_GREEN	Defines the green value of the pixel.[1]
X$W_COLR_BLUE	Defines the blue value of the pixel.[1]
X$B_COLR_FLAGS	Defines which color components are to be defined in the color map. Possible flags are as follows: x$m_do_red Sets red values x$m_do_green Sets green values x$m_do_blue Sets blue values
X$B_COLR_PAD	Makes the structure an even length.

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

The color definition data structure for the MIT C binding is shown in Figure 10–12.

Figure 10–12 Color Definition Data Structure (MIT C Binding)

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

The members of the MIT C binding color definition data structure are described in Table 10–20.

Table 10–20 Members of the Color Definition Data Structure (MIT C Binding)

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

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

X ERRORS

VAX	MIT C	Description
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.

PARSE GEOMETRY

Parses standard geometry strings.

VAX FORMAT

mask_return = **X$PARSE_GEOMETRY** *(parse_string [,x_coord_return] [,y_coord_return] [,width_return] [,height_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
mask_return	mask_longword	uns longword	write	value
parse_string	char string	character string	read	descriptor
x_coord_return	longword	longword	write	reference
y_coord_return	longword	longword	write	reference
width_return	longword	uns longword	write	reference
height_return	longword	uns longword	write	reference

MIT C FORMAT

mask_return = **XParseGeometry** *(parse_string, x_coord_return, y_coord_return, width_return, height_return)*

argument information

```
int XParseGeometry(parse_string, x_coord_return, y_coord_return,
                   width_return, height_return)
      char *parse_string;
      int *x_coord_return, *y_coord_return;
      int *width_return, *height_return;
```

RETURNS

mask_return

A bit mask that specifies which of four values (width, height, x-offset, and y-offset) were actually found in the string, and whether the x and y values are negative. Each bit indicates whether the corresponding value was found in the parsed string. For each value found, the corresponding argument is updated; for each value not found, the argument is left unchanged.

Table 10–21 lists the predefined values and their descriptions for the mask.

Table 10–21 Parse Mask Bits

Bit	VAX	MIT C	Description
1	X$M_NO_VALUE	NoValue	Reserved
2	X$M_X_VALUE	XValue	The x-coordinate of the origin of a window
3	X$M_Y_VALUE	YValue	The y-coordinate of the origin of a window
4	X$M_WIDTH_VALUE	WidthValue	The width of the window in pixels
5	X$M_HEIGHT_VALUE	HeightValue	The height of the window in pixels
6	X$M_ALL_VALUES	AllValues	Indicates if all values are present
7	X$M_X_NEGATIVE_VALUE	XNegativeValue	Indicates if the x-coordinate is negative
8	X$M_Y_NEGATIVE_VALUE	YNegativeValue	Indicates if the y-coordinate is negative

ARGUMENTS

parse_string

The name of the string that you want to parse.

VAX only

The **parse_string** argument is the address of a character string descriptor that points to the string.

MIT C only

The **parse_string** argument is a pointer to the null-terminated character string.

x_coord_return

The x-coordinate to which to return the x-offset from the specified string. This coordinate is relative to the origin of the drawable.

VAX only

This argument is optional in the VAX binding.

y_coord_return

The y-coordinate to which to return the y-offset from the specified string. This coordinate is relative to the origin of the drawable.

VAX only

This argument is optional in the VAX binding.

width_return

The width, in pixels, from the specified string.

VAX only

This argument is optional in the VAX binding.

height_return

The height, in pixels, from the specified string.

VAX only

This argument is optional in the VAX binding.

DESCRIPTION

PARSE GEOMETRY parses the string that you specify and returns a bit mask to indicate status. When you enter a command line request to perform a window operation, the client calls PARSE GEOMETRY to extract x- and y-coordinates, width, and height from the command line string.

By convention, Xlib clients use a standard string to indicate window size and placement. PARSE GEOMETRY allows you to parse the standard window geometry to conform to this standard. Specifically, this function allows you to parse strings of the following form:

$$[=]\,[\langle width\rangle x\langle height\rangle]\,[\{+-\}\,\langle xoffset\rangle\,\{+-\}\,\langle yoffset\rangle]$$

Note: **Items enclosed in < > are integers, items in [ ] are optional, and items enclosed in { } indicate that you must choose one of the items.**

The items in this form map into the arguments associated with PARSE GEOMETRY.

PARSE GEOMETRY parses the string that you specify and returns a bit mask that indicates which of the four values (width, height, x-offset, y-offset) are actually found in the string, and whether the x and y values are negative. (–0 is not equal to +0.)

For each value found, the corresponding argument is updated; for each value not found, the argument is left unchanged. The bits are set whenever one of the values is defined or a sign is set.

If the function returns either the x-value or the y-value flag, you should place the window at the requested position.

See the GEOMETRY routine for more information.

QUERY KEYMAP

Returns a bit vector that describes the state of the keyboard.

VAX FORMAT

X$QUERY_KEYMAP
(display, keys_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
keys_return	array	byte	write	reference

MIT C FORMAT

XQueryKeymap
(display, keys_return)

argument information

```
XQueryKeymap(display, keys_return)
      Display *display;
      char keys_return[32];
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

keys_return

An array of 32 bytes that identifies which keys are pressed down. Each bit represents one key of the keyboard.

DESCRIPTION

QUERY KEYMAP returns a bit vector for the logical state of the keyboard, where each one bit indicates that the corresponding key is currently pressed down. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N+7, with the least significant bit in the byte representing key 8N.

The logical state of a device, as seen by the client, may lag behind the physical state if device event processing is frozen.

REBIND KEYSYM

Rebinds the meaning of a key symbol for a client program.

VAX FORMAT

X$REBIND_KEYSYM
(display, keysym_id, keysym_names, mod_count, lookup_string, num_bytes)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
keysym_id	identifier	uns longword	read	reference
keysym_names	array	uns longword	read	reference
mod_count	longword	longword	read	reference
lookup_string	word	uns word	read	reference
num_bytes	word	uns word	read	reference

MIT C FORMAT

XRebindKeySym
(display, keysym_id, keysym_names, mod_count, lookup_string, num_bytes)

argument information

```
XRebindKeysym(display, keysym_id, keysym_names, mod_count,
              lookup_string, num_bytes)
      Display *display;
      KeySym keysym_id;
      KeySym keysym_names[];
      int mod_count;
      unsigned char *lookup_string;
      int num_bytes;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

keysym_id

The key symbol that you want to rebind.

keysym_names

The key symbols that are being used as modifiers. The length of the array is specified by **mod_count**.

mod_count

The number of modifiers in the modifier list.

lookup_string

A pointer to the string that is copied and returned by LOOKUP STRING.

num_bytes

The length of the string that is returned by LOOKUP STRING.

DESCRIPTION

REBIND KEYSYM rebinds the meaning of a key symbol for a client. REBIND KEYSYM does not rebind any key in the server, but it provides a way to attach long strings to keys. LOOKUP STRING returns these strings when the appropriate set of modifier keys is pressed and when the key symbol would have been used for translation. Note that you can rebind a key symbol that does not exist.

REFRESH KEYBOARD MAPPING

Refreshes the stored modifier and key map information.

VAX FORMAT

X$REFRESH_KEYBOARD_MAPPING
(event_map)

argument information

Argument	Usage	Data Type	Access	Mechanism
event_map	record	x$mapping_event	read	reference

MIT C FORMAT

XRefreshKeyboardMapping
(event_map)

argument information

```
XRefreshKeyboardMapping(event_map)
      XMappingEvent *map_event;
```

ARGUMENTS

event_map

The mapping event for which you want to refresh the keyboard mapping. The mapping event structure is a member of the event structure, which is shown in Section 4.1.

DESCRIPTION

REFRESH KEYBOARD MAPPING refreshes the stored modifier and key map information. When a Mapping Notify event occurs, you can use REFRESH KEYBOARD MAPPING to cause the library to refresh the stored modifier and keyboard information.

You usually call this routine when a Mapping Notify event with a request member of MAPPING KEYBOARD or MAPPING MODIFIER occurs. REFRESH KEYBOARD MAPPING updates Xlib's keyboard information.

REMOVE FROM SAVE SET

Removes the specified window from the client's save set.

VAX FORMAT

X$REMOVE_FROM_SAVE_SET *(display, window_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

MIT C FORMAT

XRemoveFromSaveSet *(display, window_id)*

argument information

```
XRemoveFromSaveSet(display, window_id)
      Display *display;
      Window window_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window you want to remove from the client's save set. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

DESCRIPTION

REMOVE FROM SAVE SET removes the specified window from the client's save set. The specified window must have been created by some other client or a Bad Match error is generated. The server automatically removes windows from the save set when they are destroyed.

Also see the CHANGE SAVESET and ADD TO SAVESET routines.

X ERRORS

VAX	MIT C	Description
X$C_BAD_MATCH	BadMatch	Possible causes are as follows: • In a graphics request, the root and depth of the graphics context do not match those of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input-only window lacks this attribute.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

REMOVE HOST

Removes a host from the list of hosts that can connect to a display.

VAX FORMAT **X$REMOVE_HOST** *(display, host)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
host	record	x$host_address	read	reference

MIT C FORMAT **XRemoveHost** *(display, host)*

argument information

```
XRemoveHost(display, host)
      Display *display;
      XHostAddress *host;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

host

A pointer to the network address of the host that you want to remove. The network data structure is shown in Section 10.1.

DESCRIPTION

REMOVE HOST dynamically removes a single host from the list of hosts that can connect to the server controlling a display. For this routine to execute successfully, the client issuing the command must reside on the same host as the server or a Bad Access error is generated. If you remove your host from the access list, you can no longer connect to that server, and this operation cannot be reversed without resetting the server.

See also the REMOVE HOSTS routine.

The network data structure is shown in Section 10.1.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

REMOVE HOSTS

Removes multiple hosts from the list of hosts that can connect to a display.

VAX FORMAT

X$REMOVE_HOSTS
(display, hosts, num_hosts)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
hosts	array	uns longword	read	reference
num_hosts	longword	longword	read	reference

MIT C FORMAT

XRemoveHosts
(display, hosts, num_hosts)

argument information

```
XRemoveHosts(display, hosts, num_hosts)
      Display *display;
      XHostAddress *hosts;
      int num_hosts;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

hosts

A pointer to the network addresses of the hosts that you want to remove. The network data structure is shown in Section 10.1.

num_hosts

The number of hosts to be removed from the access list.

DESCRIPTION

REMOVE HOSTS dynamically removes more than one host from the list of hosts that can connect to the server controlling a display. For this routine to execute successfully, the client issuing the command must reside on the same host as the server, or a Bad Access error is generated. If you remove your host from the access list, you can no longer connect to that server, and this operation cannot be reversed without resetting the server.

See also the REMOVE HOST routine.

The network data structure is shown in Section 10.1.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

REPARENT WINDOW

Changes the parent window for the specified window and repositions the window within the new parent's hierarchy.

VAX FORMAT

X$REPARENT_WINDOW
(display, window_id, parent_id, x_coord, y_coord)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
parent_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference

MIT C FORMAT

XReparentWindow
(display, window_id, parent_id, x_coord, y_coord)

argument information

```
XReparentWindow(display, window_id, parent_id, x_coord, y_coord)
      Display *display;
      Window window_id;
      Window parent_id;
      int x_coord, y_coord;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to receive the new parent window.

parent_id

The identifier of the new parent window for the window specified in **window_id**. The routine fails under the following conditions:

- If the new parent window is not on the same screen as the old parent window
- If the new parent window is the specified window or an inferior of the specified window
- If the specified window has a Parent Relative background and if the new parent window is not the same depth as the specified window

x_coord

The x-coordinate of the new position of the window relative to the new parent window's origin. The x- and y-coordinates define the upper left corner of the new position for the specified window.

y_coord

The y-coordinate of the new position of the window relative to the new parent window's origin. The x- and y-coordinates define the upper left corner of the new position for the specified window.

DESCRIPTION

REPARENT WINDOW reparents the specified window by inserting it as the child of the specified parent. If the window specified in **window_id** is mapped, REPARENT WINDOW automatically unmaps it. REPARENT WINDOW then moves 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 sibling windows.

After reparenting the specified window, REPARENT WINDOW causes the server to generate a Reparent Notify event. The override redirect member of the Reparent Event structure returned by this event is set to the window's corresponding override redirect attribute.

The override redirect member specifies whether the window manager should intercept any map or configuration request for the window. Window manager clients normally ignore the Reparent Notify event if the override redirect member is set to true.

If the specified window was originally mapped, the server performs a MAP WINDOW request on it. The server performs normal exposure processing on formerly obscured windows. The server might not generate exposure events for regions from the initial UNMAP WINDOW request that are immediately obscured by the final MAP WINDOW request.

X ERRORS

VAX	MIT C	Description
X$C_BAD_MATCH	BadMatch	Possible causes are as follows: • In a graphics request, the root and depth of the graphics context do not match those of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input-only window lacks this attribute.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

RESET SCREEN SAVER

Resets the screen saver.

VAX FORMAT

X$RESET_SCREEN_SAVER *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT

XResetScreenSaver *(display)*

argument information

```
XResetScreenSaver(display)
     Display *display;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

DESCRIPTION

RESET SCREEN SAVER resets the screen saver as if device input had been received. The timeout period is started over.

See the SET SCREEN SAVER routine.

SET ACCESS CONTROL

Changes the access control mode of a display to enable or disable.

VAX FORMAT

X$SET_ACCESS_CONTROL
(display, access_mode)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
access_mode	longword	longword	read	reference

MIT C FORMAT

XSetAccessControl
(display, access_mode)

argument information

```
XSetAccessControl(display, access_mode)
      Display *display;
      int access_mode;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

access_mode

Whether you want to change the access control mode. The predefined values for **access_mode** are as follows:

VAX	MIT C	Description
X$C_ENABLE_ACCESS	EnableAccess	Enables access control for the display.
X$C_DISABLE_ACCESS	DisableAccess	Disables access control for the display.

DESCRIPTION

SET ACCESS CONTROL either enables or disables the use of the access control list at connection setups. For this routine to execute successfully, the client must reside on the same host as the server.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ACCESS	BadAccess	Possible causes are as follows: • An attempt to grab a key/button combination that has already been grabbed by another client • An attempt to free a color map entry that was not allocated by the client • An attempt to store in a read-only or unallocated color map entry • An attempt to modify the access control list from other than the local host • An attempt to select an event type that at most one client can select at a time, when another client has already selected it
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.

SET CLOSE DOWN MODE

Defines what happens to a client's resources when the client disconnects.

VAX FORMAT

X$SET_CLOSE_DOWN_MODE
(display, close_mode)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
close_mode	longword	longword	read	reference

MIT C FORMAT

XSetCloseDownMode
(display, close_mode)

argument information

```
XSetCloseDownMode(display, close_mode)
      Display *display;
      int close_mode;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

close_mode

The close-down mode for the client's resources. The predefined values for **close_mode** are as follows:

VAX	MIT C	Description
X$C_DESTROY_ALL	DestroyAll	All client resources are freed.
X$C_RETAIN_ PERMANENT	RetainPermanent	All client resources are marked as permanent.
X$C_RETAIN_ TEMPORARY	RetainTemporary	All client resources are marked as temporary.

DESCRIPTION

SET CLOSE DOWN MODE defines what happens to a client's resources when the client disconnects from the server. The default mode is Destroy All, which frees all client resources. See the CLOSE DISPLAY routine for more information about close-down modes.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SET INPUT FOCUS

Changes the input focus to the specified window.

VAX FORMAT

X$SET_INPUT_FOCUS
(display, focus_id, revert_to, time)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
focus_id	identifier	uns longword	read	reference
revert_to	longword	longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

XSetInputFocus
(display, focus_id, revert_to, time)

argument information

```
XSetInputFocus(display, focus_id, revert_to, time)
      Display *display;
      Window focus_id;
      int revert_to;
      Time time;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

focus_id
The window identifier of the window in which you want to set the input focus.

The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. When the window identifier is specified, that window becomes the keyboard's focus window. When keyboard events are normally reported to this window (or its inferiors), the events continue to be reported. Otherwise, the event is reported with respect to the focus window.

One of the following predefined values can be specified instead of the window identifier:

VAX	MIT C	Description
X$C_POINTER_ ROOT	PointerRoot	The focus window is the root window of the screen that the pointer is on at each keyboard event. The **revert_to** argument is not taken into account.
X$C_NONE	None	All keyboard events are discarded until a new focus window is set. The window specified in **revert_to** is not taken into account.

revert_to

Where the input focus moves to when the focus window becomes unviewable.

One of the following predefined values can be specified:

VAX	MIT C	Description
X$C_REVERT_ TO_PARENT	RevertToParent	The input focus is changed to the parent window, or the closest viewable ancestor, and the new **revert_to** value is taken to be Revert To None.
X$C_REVERT_ TO_POINTER_ ROOT	RevertToPointerRoot	The input focus reverts to Pointer Root. When the focus reverts, the server generates Focus In and Focus Out events, but the last-focus-change time is not affected.
X$C_REVERT_ TO_NONE	RevertToNone	The input focus reverts to None. When the focus reverts, the server generates Focus In and Focus Out events, but the last-focus-change time is not affected.

time

The time when the events are to be released. Either a timestamp, in milliseconds, or the predefined value X$C_CURRENT_TIME or CurrentTime can be specified.

DESCRIPTION

SET INPUT FOCUS changes the input focus and the last-focus-change time. The specified window must be viewable when SET INPUT FOCUS is called or a Bad Match error is generated. If the window specified (in **focus_id**) later becomes unviewable, the routine determines a new focus window according to the **revert_to** argument.

The routine generates Focus In and Focus Out events.

If the time specified in **time** is earlier than the current last-focus-change time, or if it is later than the current server time, the input focus does not change.

Use the GET INPUT FOCUS routine to obtain the values specified in **focus_id** and **revert_to**.

X ERRORS

VAX	MIT C	Description
X$C_BAD_MATCH	BadMatch	Possible causes are as follows: • In a graphics request, the root and depth of the graphics context do not match those of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input-only window lacks this attribute.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET MODIFIER MAPPING

Specifies the key codes for the modifier keys.

VAX FORMAT

status_return = **X$SET_MODIFIER_MAPPING** *(display, modifier_keys)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
modifier_keys	record	x$modifier_keymap	read	reference

MIT C FORMAT

status_return = **XSetModifierMapping** *(display, modifier_keys)*

argument information

```
int  XSetModifierMapping(display, modifier_keys)
        Display *display;
        XModifierKeymap *modifier_keys;
```

RETURNS

status_return

A server can impose restrictions on how modifiers can be changed. If such a restriction is violated, SET MODIFIER MAPPING returns a status message. SET MODIFIER MAPPING returns the following status:

VAX	MIT C	Description
X$C_MAPPING_ SUCCESS	MappingSuccess	The routine completed successfully.
X$C_MAPPING_ FAILED	Mapping Failed	None of the modifiers is changed.
X$C_MAPPING_ BUSY	MappingBusy	New key codes specified for a modifier differ from those currently defined and any (current or new) keys for that modifier are in the logically down state. None of the modifiers is changed.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

modifier_keys

A pointer to the modifier key map data structure.

DESCRIPTION

SET MODIFIER MAPPING specifies the key codes of the keys, if any, that are to be used as modifiers. Up to eight modifier keys are allowed; if more that eight are specified in the modifier key map data structure, a Bad Length error is generated.

There are eight modifiers, and the modifier map member of the modifier key map data structure contains eight sets of maximum key-per-modifier key codes, one for each modifier in the order shift, lock, control, mod1, mod2, mod3, mod4, and mod5. Only nonzero key codes have meaning in each set.

Nonzero key codes must be in the range specified by minimum key code and maximum key code in the display structure or else a Bad Value error is generated. No key code may appear twice in the entire map, or else a Bad Value error is generated.

See the GET MODIFIER MAPPING routine.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ALLOC	BadAlloc	The server failed to allocate the requested resource for any cause.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SET POINTER MAPPING

Enables or disables buttons for the pointer.

VAX FORMAT

status_return = **X$SET_POINTER_MAPPING** *(display, pointer_map, num_maps)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
pointer_map	array	byte	read	reference
num_maps	word	uns word	read	reference

MIT C FORMAT

status_return = **XSetPointerMapping** *(display, pointer_map, num_maps)*

argument information

```
int XSetPointerMapping(display, pointer_map, num_maps)
      Display *display;
      unsigned char pointer_map[];
      int num_maps;
```

RETURNS

status_return

Whether the routine completed successfully. SET POINTER MAPPING returns one of the following status messages:

VAX	MIT C	Description
X$C_MAPPING_ SUCCESS	MappingSuccess	The mapping list is successfully defined.
X$C_MAPPING_ BUSY	MappingBusy	One of the buttons to be altered is in the logically down state. The mapping list is not changed.
X$C_MAPPING_ FAILED	MappingFailed	The mapping list is not changed.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

pointer_map

A pointer to an array of elements that define the mapping list. The array is indexed, starting from one. The index is a "core" button number. An empty element disables its corresponding button. No two elements can have the same nonzero value. The length of the array is specified by **num_maps**.

num_maps

The number of items in the mapping list. Each item corresponds to a physical button. The value must match that returned by the GET POINTER MAPPING routine. This value specifies the length of the array in **pointer_map**.

DESCRIPTION

SET POINTER MAPPING defines the mapping list for the pointer. Each item in the mapping list corresponds to a physical button. When one of the items has an empty value, the corresponding button is disabled.

Use GET POINTER MAPPING to obtain the mapping list once it is defined.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SET SCREEN SAVER

Sets the following values for screen saving: the timeout period, the interval, whether to blank screen, and whether to allow exposures.

VAX FORMAT

X$SET_SCREEN_SAVER
(display, timeout, interval, prefer_blanking, allow_exposures)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
timeout	longword	longword	read	reference
interval	longword	longword	read	reference
prefer_blanking	longword	longword	read	reference
allow_exposures	longword	longword	read	reference

MIT C FORMAT

XSetScreenSaver
(display, timeout, interval, prefer_blanking, allow_exposures)

argument information

```
XSetScreenSaver(display, timeout, interval, prefer_blanking,
                allow_exposures)
      Display *display;
      int timeout, interval;
      int prefer_blanking;
      int allow_exposures;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

timeout

The time, in seconds, that the screen saver waits before being invoked. The time represents the number of seconds when no input from the keyboard or pointing device is received. A value of –1 restores the default value. If **timeout** is nonzero, the screen saver is enabled.

interval

The time, in seconds, from one screen saver invocation to the next. A value of 0 indicates that no periodic change is made.

prefer_blanking

The mode for whether to blank the screen during a screen save operation. The predefined values for **prefer_blanking** are as follows:

VAX	MIT C	Description
X$C_DONT_ PREFER_ BLANKING	DontPreferBlanking	Do not blank the screen. If exposures are allowed, or if the screen can be regenerated without sending exposure events to clients, the screen is tiled with the root window background tile. If exposures are not allowed or the exposure events are sent to clients, then the screen does not change.
X$C_PREFER_ BLANKING	PreferBlanking	Blank the screen. This can be used only if the hardware supports video blanking.
X$C_DEFAULT_ BLANKING	DefaultBlanking	Use the default.

Other values specified in this argument are not valid.

allow_exposures

The screen saver control values. The predefined values for **allow_exposures** are as follows:

VAX	MIT C	Description
X$C_DONT_ALLOW_ EXPOSURES	DontAllowExposures	Exposures are not allowed.
X$C_ALLOW_ EXPOSURES	AllowExposures	Exposures are allowed.
X$C_DEFAULT_ EXPOSURES	DefaultExposures	Use the default value.

Other values specified in this argument are not valid.

DESCRIPTION

SET SCREEN SAVER specifies how the screen saver should work. You set the following values:

- The time that elapses from the last device input before the screen saver is invoked (**timeout**)
- The time between invocations of the screen saver (**interval**)

  If the server-dependent screen saver method supports periodic change, **interval** serves as an indication as to how long the change period should be. Zero indicates that no periodic change should be made.
- Whether to blank the screen (**prefer_blanking**)

  For each screen, if blanking is preferred and the hardware supports video blanking, the screen becomes blank. Otherwise, either if exposures are allowed or if the screen can be regenerated without sending exposure events to the clients, the screen is tiled with the root window background tile randomly re-originated at **interval** minutes.

Otherwise, the state of the screen does not change and the screen saver is not activated.

- Whether to allow exposures (**allow_exposures**)

All screen states are restored at the next input from a device or at the next call to FORCE SCREEN SAVER with a mode of Screen Saver Reset. Examples of ways to change the screen include scrambling the color map periodically, moving an icon image around the screen periodically, or tiling the screen with the root window background tile.

See the RESET SCREEN SAVER routine.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

STRING TO KEYSYM

Converts the name of the key symbol to the name of the key symbol code.

VAX FORMAT

keysym_return = **X$STRING_TO_KEYSYM** *(keysym_name)*

argument information

Argument	Usage	Data Type	Access	Mechanism
keysym_return	identifier	uns longword	write	value
keysym_name	char string	character string	read	descriptor

MIT C FORMAT

keysym_return = **XStringToKeysym** *(keysym_name)*

argument information

```
KeySym XStringToKeysym(keysym_name)
      char *keysym_name;
```

RETURNS

keysym_return

The key symbol code for the key symbol name that you specify.

ARGUMENTS

keysym_name

The name of the key symbol that is to be converted.

DESCRIPTION

STRING TO KEYSYM converts the name of the key symbol to the key-symbol code. Valid key-symbol names are listed in SYS$LIBRARY:DECW$XLIBDEF and DECW$INCLUDE:KEYSYMDEF.H. If the specified string does not match a valid key symbol, STRING returns X$C_NO_SYMBOL or NoSymbol.

UNGRAB BUTTON

Deactivates the passive grab for a pointing device button press.

VAX FORMAT

X$UNGRAB_BUTTON
(display, button, modifiers, window_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
button	longword	longword	read	reference
modifiers	mask_longword	uns longword	read	reference
window_id	identifier	uns longword	read	reference

MIT C FORMAT

XUngrabButton
(display, button, modifiers, window_id)

argument information

```
XUngrabButton(display, button, modifiers, window_id)
      Display *display;
      unsigned int button;
      unsigned int modifiers;
      Window window_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

button

The button on the pointing device that is no longer grabbed. The possible values are as follows:

VAX Predefined Value	MIT C Predefined Value
X$C_BUTTON1	Button1
X$C_BUTTON2	Button2
X$C_BUTTON3	Button3
X$C_BUTTON4	Button4
X$C_BUTTON5	Button5
X$C_ANY_BUTTON	AnyButton

The predefined value X$C_ANY_BUTTON or AnyButton can be specified to allow any pointer button to be released.

modifiers

A bit mask that specifies the set of key masks associated with the button grab. This mask is the inclusive OR of these key mask bits:

Bit	VAX Predefined Value	MIT C Predefined Value
1	X$M_SHIFT	ShiftMask
2	X$M_CAPS_LOCK	LockMask
3	X$M_CONTROL	ControlMask
4	X$M_MOD1	Mod1Mask
5	X$M_MOD2	Mod2Mask
6	X$M_MOD3	Mod3Mask
7	X$M_MOD4	Mod4Mask
8	X$M_MOD5	Mod5Mask

Clients can also pass the X$_ANY_MODIFIER or AnyModifier constants, which is equivalent to issuing the ungrab request for all possible modifier combinations (including the combination of no modifiers).

window_id

The window associated with the button to be ungrabbed. The window identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

DESCRIPTION

UNGRAB BUTTON releases a passive grab on a specified pointing device button press. This routine does not affect any active grab.

See the GRAB BUTTON routine.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

UNGRAB KEY

Releases the key combination on the specified window that was grabbed.

VAX FORMAT

X$UNGRAB_KEY
(display, keycode, modifiers, window_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
keycode	longword	longword	read	reference
modifiers	mask_longword	uns longword	read	reference
window_id	identifier	uns longword	read	reference

MIT C FORMAT

XUngrabKey
(display, keycode, modifiers, window_id)

argument information

```
XUngrabKey(display, keycode, modifiers, window_id)
      Display *display;
      int keycode;
      unsigned int modifiers;
      Window window_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

keycode

The key code that maps to the specific key to be released. You can pass either the key code or predefined value X$C_ANY_KEY or AnyKey, which is equivalent to issuing the request for all possible key codes.

modifiers

A bit mask that specifies the set of key masks associated with the button grab. This mask is the inclusive OR of these key mask bits:

Bit	VAX Predefined Value	MIT C Predefined Value
1	X$M_SHIFT	ShiftMask
2	X$M_CAPS_LOCK	LockMask
3	X$M_CONTROL	ControlMask

Bit	VAX Predefined Value	MIT C Predefined Value
4	X$M_MOD1	Mod1Mask
5	X$M_MOD2	Mod2Mask
6	X$M_MOD3	Mod3Mask
7	X$M_MOD4	Mod4Mask
8	X$M_MOD5	Mod5Mask

Clients can also pass the X$_ANY_MODIFIER or AnyModifier constants, which is equivalent to issuing the ungrab request for all possible modifier combinations (including the combination of no modifiers).

window_id

The identifier of the window associated with the keys to be ungrabbed. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

DESCRIPTION

UNGRAB KEY releases the key combination on the specified window from a previous GRAB KEY by the same client.

See the GRAB KEY routine.

X ERRORS

VAX	MIT C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

UNGRAB KEYBOARD

Releases an active grab on the main keyboard and any queued events.

VAX FORMAT **X$UNGRAB_KEYBOARD** *(display, time)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT **XUngrabKeyboard** *(display, time)*

argument information

```
XUngrabKeyboard(display, time)
     Display *display;
     Time time;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

time

The time when the events are to be released. Either a timestamp, in milliseconds, or the predefined value X$C_CURRENT_TIME or CurrentTime can be specified.

MIT C only

If the time specified in **time** is earlier than the last-pointer-grab time, or if it is later than the current server time, the keyboard is not released.

DESCRIPTION

UNGRAB KEYBOARD releases an active grab on the main keyboard and any queued events from a GRAB KEYBOARD or GRAB KEY request by the same client. It generates Focus In and Focus Out events.

See the GRAB KEYBOARD routine.

UNGRAB POINTER

Releases the active grab on the specified pointer and any queued events.

VAX FORMAT

X$UNGRAB_POINTER *(display, time)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
time	longword	uns longword	read	reference

MIT C FORMAT

XUngrabPointer *(display, time)*

argument information

```
XUngrabPointer(display, time)
      Display *display;
      Time time;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

time

The time when the events are to be released. Either a timestamp, in milliseconds, or the predefined value X$C_CURRENT_TIME or CurrentTime can be specified.

MIT C only

If the time specified in **time** is earlier than the last-pointer-grab time, or if it is later than the current server time, the pointer is not released.

DESCRIPTION

UNGRAB POINTER releases the pointer and any queued events if this client has actively grabbed the pointer with a GRAB POINTER, GRAB BUTTON or normal button press. It generates Enter Notify and Leave Notify events. The server automatically ungrabs the pointer when the event window or confine-to window for an active pointer grab is not viewable, or if window reconfiguration causes the confine-to window to lie completely outside the boundaries of the root window.

See the GRAB POINTER routine.

UNGRAB SERVER

Relinquishes exclusive possession of the server associated with the display that you specify.

VAX FORMAT **X$UNGRAB_SERVER** *(display)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

MIT C FORMAT **XUnGrabServer** *(display)*

argument information

```
XUngrabServer(display)
      Display *display;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

DESCRIPTION

UNGRAB SERVER relinquishes possession of a server that you grabbed in a previous call to GRAB SERVER. A client automatically ungrabs the server when it closes its connection to that server.

See the GRAB SERVER routine.

UNINSTALL COLORMAP

Uninstalls a color map for a screen.

VAX FORMAT

X$UNINSTALL_COLORMAP
(display, colormap_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference

MIT C FORMAT

XUninstallColormap
(display, colormap_id)

argument information

```
XUninstallColormap(display, colormap_id)
      Display *display;
      Colormap colormap_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map to be replaced. If you pass a color map identifier, and that color map is in the required list, the color map is removed from the required list.

DESCRIPTION

UNINSTALL COLORMAP removes the specified color map from the required list for the associated screen. As a result, the specified color map might be uninstalled, and the server might implicitly install or uninstall additional color maps. The color maps that are installed or uninstalled are server-dependent, except that the required list must remain installed. The identifier of the color map was originally returned by CREATE COLORMAP.

If the specified color map becomes uninstalled, the server generates a Colormap Notify event on every window having **colormap_id** as the color map. In addition, for every other color map that is installed or uninstalled as a result of calling UNINSTALL COLORMAP, the server generates a Colormap Notify event on every window having **colormap_id** as the color map.

As soon as the replacement color map is installed, the colors are immediately displayed on the windows associated with that color map.

X ERRORS

VAX	MIT C	Description
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.

WARP POINTER

Moves the pointer to any specified location on the screen.

VAX FORMAT

X$WARP_POINTER
(display, src_window_id, dst_window_id, src_x_coord, src_y_coord, src_width, src_height, dst_x_coord, dst_y_coord)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
src_window_id	identifier	uns longword	read	reference
dst_window_id	identifier	uns longword	read	reference
src_x_coord	longword	longword	read	reference
src_y_coord	longword	longword	read	reference
src_width	longword	uns longword	read	reference
src_height	longword	uns longword	read	reference
dst_x_coord	longword	longword	read	reference
dst_y_coord	longword	longword	read	reference

MIT C FORMAT

XWarpPointer
(display, src_window_id, dst_window_id, src_x_coord, src_y_coord, src_width, src_height, dst_x_coord, dst_y_coord)

argument information

```
XWarpPointer(display, src_window_id, dest_window_id, src_x_coord,
             src_y_coord, src_width, src_height, dst_x_coord,
             dst_y_coord)
        Display *display;
        Window src_window_id, dest_window_id;
        int src_x_coord, src_y_coord;
        unsigned int src_width, src_height;
        int dst_x_coord, dst_y_coord;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

src_window_id

The identifier of the window where the pointer is currently located. Clients can pass the window identifier or the constant X$C_NONE or None. If none is specified, the move is independent of the current pointer position. If **src_window_id** is a window, the move takes place only if **src_window_id** contains the pointer, and the pointer is currently contained in the specified rectangle of the source window.

The identifiers of the windows were originally returned by CREATE SIMPLE WINDOW or WINDOW.

dst_window_id

The identifier of the window where the pointer will be located. Clients can pass the window identifier or the constant X$C_NONE or None. If a window identifier is specified, WARP POINTER moves the pointer to **dst_x_coord** and **dst_y_coord** relative to the origin of **dst_window_id**.

If none is specified, the pointer is moved by offsets specified by **dst_x_coord** and **dst_y_coord** relative to the current position of the pointer.

The identifiers of the windows were originally returned by CREATE SIMPLE WINDOW or WINDOW.

src_x_coord

The x-coordinate within the source window. The source x- and y-coordinates define the current location of the pointer and are relative to the origin of **src_window_id**.

src_y_coord

The y-coordinate within the source window. The source x- and y-coordinates define the current location of the pointer and are relative to the origin of **src_window_id**.

src_width

The width of a rectangle in the source window. The width and height define the area of the source window. If **src_width** is zero, WARP POINTER replaces it with the current width of the source window, minus **src_x_coord**.

src_height

The height of a rectangle in the source window. The width and height define the area of the source window. If **src_height** is zero, WARP POINTER replaces it with the current height of the source window, minus **src_y_coord**.

dst_x_coord

The x-coordinate within the destination window. The destination x- and y-coordinates define the new location of the pointer.

dst_y_coord

The y-coordinate within the destination window. The destination x- and y-coordinates define the new location of the pointer.

DESCRIPTION

WARP POINTER moves the pointer to any location on the screen. This task is usually done with the mouse or other pointing device and there is seldom any need to call this routine. However, if pointer motion must be done with this routine, the same events are generated as if the pointer had been physically moved.

Note that you cannot use WARP POINTER to move the pointer outside the confine_to window of an active pointer grab; an attempt to do so moves the pointer only as far as the closest edge of the confine_to window.

X ERRORS

VAX	MIT C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

11 Pixmap and Bitmap Routines

The pixmap and bitmap routines allow you to create and work with offscreen images. The format of the file name by which a bitmap is read or written is operating system specific. The format of the bitmap itself is as follows:

```
#define name_width width
#define name_height height
#define name_x_hot  x
#define name_y_hot  y
static char name_bits[]={ 0xNN,...}
```

Note: If you are using a language other than C, your program must translate the bitmap format into something that it can interpret.

Variables ending with the suffixes x_hot and y_hot are optional and are present only if a hotspot is defined for the bitmap. The other variables are required. The bit array must be large enough to contain the bitmap. The bitmap unit is 8. (Refer to the description of the image data structure in the *VMS DECwindows Xlib Programming Volume* for more information about the bitmap_unit member.) The bitmap file name is derived by deleting the path and file extension from the file name that you specify.

For information on how to use the pixmap routines, see the *VMS DECwindows Xlib Programming Volume.*

The routines described in this chapter are listed in Table 11–1.

Table 11–1 Pixmap and Bitmap Routines

Routine Name	Description
CREATE BITMAP FROM DATA	Includes a bitmap written by WRITE BITMAP FILE.
CREATE PIXMAP	Creates a pixmap of a given size.
CREATE PIXMAP FROM BITMAP DATA	Creates a pixmap and then calls PUT IMAGE to format the bitmap data.
FREE PIXMAP	Frees pixmap storage.
READ BITMAP FILE	Reads a file that contains a bitmap into separate in-memory components.
WRITE BITMAP FILE	Writes an existing bitmap to a file.

11.1 Pixmap and Bitmap Routines

The following pages describe the Xlib pixmap and bitmap routines.

CREATE BITMAP FROM DATA

Includes a bitmap written by the WRITE BITMAP FILE routine.

VAX FORMAT

pixmap_id_return = **X$CREATE_BITMAP_FROM_DATA** *(display, drawable_id, data, width, height)*

argument information

Argument	Usage	Data Type	Access	Mechanism
pixmap_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
data	array	byte	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference

MIT C FORMAT

pixmap_id_return = **XCreateBitmapFromData** *(display, drawable_id, data, width, height)*

argument information

```
Pixmap XCreateBitmapFromData(display, drawable_id, data, width,
                             height)
                 Display *display;
                 Drawable drawable_id;
                 char *data;
                 unsigned int width, height;
```

RETURNS

pixmap_id_return

The identifier of the pixmap that CREATE BITMAP FROM DATA returns.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

drawable_id

The identifier of the drawable. This is used to determine the screen for which to create the bitmap.

data

The data from which to create the bitmap.

width

The width of the bitmap to be created.

height

The height of the bitmap to be created.

DESCRIPTION

CREATE BITMAP FROM DATA creates a bitmap file from data written by WRITE BITMAP FILE, or from data that you produced yourself in the X11 bitmap format. For example, the following MIT C binding example shows how to get a gray bitmap:

```
#include "gray.bitmap"
```

```
Pixmap XCreateBitmapFromData(display, window, gray_bits,
gray_width, gray_height);
```

CREATE BITMAP FROM DATA should be used to create bitmaps for specifying stipple patterns, clipping regions, cursor shades, and icon shapes.

If insufficient working storage was allocated, CREATE BITMAP FROM DATA returns a null value. Clients must free the bitmap by using FREE PIXMAP when done.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.

CREATE PIXMAP

Creates a pixmap of a given size.

VAX FORMAT

pixmap_id_return = **X$CREATE_PIXMAP** *(display, drawable_id, width, height, depth)*

argument information

Argument	Usage	Data Type	Access	Mechanism
pixmap_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
depth	longword	longword	read	reference

MIT C FORMAT

pixmap_id_return = **XCreatePixmap** *(display, drawable_id, width, height, depth)*

argument information

```
Pixmap XCreatePixmap(display, drawable_id, width, height, depth)
      Display *display;
      Drawable drawable_id;
      unsigned int width, height;
      unsigned int depth;
```

RETURNS

pixmap_id_return

The identifier of the pixmap created. This identifier is used to manipulate the pixmap in subsequent routines.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

drawable_id

The identifier of the drawable on which the new pixmap is created. The drawable can be an input-only window.

width

The width, in pixels, of the pixmap. This must be a positive value. The width and height define the two-dimensional size of the pixmap.

height

The height, in pixels, of the pixmap. This must be a positive value. The width and height define the two-dimensional size of the pixmap.

depth

The depth of the pixmap. The depth must be supported by the root of the drawable, specified by **drawable_id**. A pixmap of depth 1 is a bitmap.

DESCRIPTION

CREATE PIXMAP creates a pixmap of the width, height, and depth that you specify and assigns a pixmap identifier to it. It is valid to pass a window whose class is input only to the drawable argument. The width and height arguments must be nonzero. Otherwise, a Bad Value error is generated. The depth argument must be one of the depths supported by the screen of the specified drawable or a Bad Value error is generated.

The server uses the identifier specified in **drawable_id** to determine the screen on which to store the new pixmap. The new pixmap can be used only on this screen and only with other drawables of the same depth. (See the COPY PLANE routine for an exception to this rule.) The initial contents of the pixmap are undefined.

X ERRORS

VAX	MIT C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

CREATE PIXMAP FROM BITMAP DATA

Creates a pixmap of the specified depth and then calls PUT IMAGE to format the bitmap data into the pixmap.

VAX FORMAT

pixmap_id_return =
X$CREATE_PIX_FROM_BITMAP_DATA
(display, drawable_id, data, width, height, foreground, background, depth)

argument information

Argument	Usage	Data Type	Access	Mechanism
pixmap_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
data	array	byte	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
foreground	longword	uns longword	read	reference
background	longword	uns longword	read	reference
depth	longword	longword	read	reference

MIT C FORMAT

pixmap_id_return =
XCreatePixmapFromBitmapData
(display, drawable_id, data, width, height, foreground, background, depth)

argument information

```
Pixmap XCreatePixmapFromBitmapData(display, drawable_id,
                                   data, width, height,
                                   foreground, background,
                                   depth)
      Display *display;
      Drawable drawable_id;
      char *data
      unsigned int width, height;
      unsigned long foreground, background
      unsigned int depth;
```

RETURNS

pixmap_id_return

The identifier of the pixmap created. This identifier is used to manipulate the pixmap in subsequent routines.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

drawable_id

The identifier of the drawable for which the new pixmap is created.

data

Specifies the data in bitmap format.

width

The width, in pixels, of the pixmap. The width and height define the two-dimensional size of the pixmap.

height

The height, in pixels, of the pixmap. The width and height define the two-dimensional size of the pixmap.

foreground

The foreground pixel values to use.

background

The background pixel values to use.

depth

The depth of the pixmap. The depth must be supported by the root of the drawable, specified by **drawable_id**. A pixmap of depth 1 is a bitmap.

DESCRIPTION

CREATE PIXMAP FROM BITMAP DATA creates a pixmap of the given depth and then calls PUT IMAGE to format the bitmap data into the pixmap. CREATE PIXMAP FROM BITMAP DATA should be used to create pixmaps for tiles and images in bitmap format.

Note that in the VAX binding the name of this routine has been shortened to stay within the 32 character limit.

FREE PIXMAP

Dissociates the pixmap storage as well as the identifier.

VAX FORMAT

X$FREE_PIXMAP *(display, pixmap_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
pixmap_id	identifier	uns longword	read	reference

MIT C FORMAT

XFreePixmap *(display, pixmap_id)*

argument information

```
XFreePixmap(display, pixmap_id)
      Display *display;
      Pixmap pixmap_id;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

pixmap_id
The identifier of the pixmap to be freed.

DESCRIPTION

FREE PIXMAP dissociates the identifier from the specified pixmap. The server frees the pixmap storage when no other resources reference the pixmap. The pixmap should never be referenced again. The identifier of the pixmap was originally returned by CREATE PIXMAP, CREATE BITMAP FROM DATA or CREATE PIXMAP FROM BITMAP DATA.

X ERRORS

VAX	MIT C	Description
X$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.

READ BITMAP FILE

Reads a file that contains a bitmap into separate in-memory components.

VAX FORMAT

status_return = **X$READ_BITMAP_FILE** *(display, drawable_id, filename [,width_return] [,height_return] [,bitmap_id_return] [,x_hot_coord_return] [,y_hot_coord_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond value	uns longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
filename	char string	character string	read	descriptor
width_return	longword	uns longword	write	reference
height_return	longword	uns longword	write	reference
bitmap_id_return	identifier	uns longword	write	reference
x_hot_coord_return	longword	longword	write	reference
y_hot_coord_return	longword	longword	write	reference

MIT C FORMAT

status_return = **XReadBitmapFile** *(display, drawable_id, filename, width_return, height_return, bitmap_id_return, x_hot_coord_return, y_hot_coord_return)*

argument information

```
int  XReadBitmapFile(display, drawable_id, filename,
                     width_return, height_return,
                     bitmap_id_return, x_hot_coord_return,
                     y_hot_coord_return)
      Display *display;
      Drawable drawable_id;
      char *filename;
      unsigned int *width_return, *height_return;
      Pixmap *bitmap_id_return;
      int *x_hot_coord_return, *y_hot_coord_return;
```

RETURNS

status_return

Returns one of the following values to indicate the status:

VAX	C	Description
X$C_BITMAP_SUCCESS	BitmapSuccess	The file is readable and valid.
X$C_BITMAP_OPEN_FAILED	BitmapOpenFailed	READ BITMAP FILE cannot open the file.
X$C_BITMAP_FILE_INVALID	BitmapFileInvalid	READ BITMAP FILE opens the file but the file does not contain valid bitmap data.
X$C_BITMAP_NO_MEMORY	BitmapNoMemory	There is not enough memory to load the bitmap file.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

drawable_id

The identifier of the drawable. READ BITMAP FILE uses this argument to determine the screen that is being used.

filename

The file specification of the bitmap file. The format of the file is dependent on the operating system on the client side of the client-server connection. VAX binding file names are parsed using RMS$PARSE and logical names, search strings, and so on are supported. MIT C binding file names are parsed using **fopen**. The maximum length of a file specification is 255 bytes. Wildcards are not supported. The default file name is [ ]bitmap.dat.

VAX only

The **filename** argument is the address of a character string descriptor that points to the string.

MIT C only

The **filename** argument is a pointer to a null-terminated character string.

width_return

The width of the read-in bitmap file.

VAX only

The **width_return** argument is optional in the VAX binding.

height_return

The height of the read-in bitmap file.

VAX only

The **height_return** argument is optional in the VAX binding.

bitmap_id_return

The bitmap identifier.

VAX only

The **bitmap_id_return** argument is optional in the VAX binding.

x_hot_coord_return

The x-coordinate of the hotspot, which is defined as the point in the cursor that corresponds to the x- and y-coordinates reported for the pointer, is returned. If **x_hot_coord_return** and **y_hot_coord_return** are not null, READ BITMAP FILE sets **x_hot_coord_return** and **y_hot_coord_return** to the value of the hotspot as defined in the file.

If no hotspot is defined, READ BITMAP FILE sets **x_hot_coord_return** and **y_hot_coord_return** to (–1, –1).

VAX only

The **x_hot_coord_return** argument is optional in the VAX binding.

y_hot_coord_return

The y-coordinate of the hotspot, which is defined as the point in the cursor that corresponds to the x- and y-coordinates reported for the pointer, is returned. If **x_hot_coord_return** and **y_hot_coord_return** are not null, READ BITMAP FILE sets **x_hot_coord_return** and **y_hot_coord_return** to the value of the hotspot as defined in the file.

If no hotspot is defined, READ BITMAP FILE sets **x_hot_coord_return** and **y_hot_coord_return** to (–1, –1).

VAX only

The **y_hot_coord_return** argument is optional in the VAX binding.

DESCRIPTION

READ BITMAP FILE reads in a file that contains a bitmap and assigns the bitmap's height, width, and hotspot coordinates, as read from the file, to the caller's height and width variables and hotspot coordinates. READ BITMAP FILE creates a pixmap of the appropriate size, reads the bitmap data from the file into the pixmap, and assigns the pixmap to the caller's bitmap variable. READ BITMAP FILE reads files in the format output by WRITE BITMAP FILE.

The caller must free the bitmap by using FREE PIXMAP when done.

See the WRITE BITMAP FILE routine.

WRITE BITMAP FILE

Writes an existing bitmap to a file in X11 format.

VAX FORMAT

status_return = **X$WRITE_BITMAP_FILE** *(display, filename, bitmap_id, width, height, x_hot_coord, y_hot_coord)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond value	uns longword	write	value
display	identifier	uns longword	read	reference
filename	char string	character string	read	descriptor
bitmap_id	identifier	uns longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
x_hot_coord	longword	longword	read	reference
y_hot_coord	longword	longword	read	reference

MIT C FORMAT

status_return = **XWriteBitmapFile** *(display, filename, bitmap_id, width, height, x_hot_coord, y_hot_coord)*

argument information

```
int XWriteBitmapFile(display, filename, bitmap_id, width,
                     height, x_hot_coord, y_hot_coord)
      Display *display;
      char *filename;
      Pixmap bitmap_id;
      unsigned int width, height;
      int x_hot_coord, y_hot_coord;
```

RETURNS

status

Returns one of the following values to indicate the status:

VAX	C	Description
X$C_BITMAP_SUCCESS	BitmapSuccess	The write operation was successful.

VAX	C	Description
X$C_BITMAP_OPEN_FAILED	BitmapOpenFailed	WRITE BITMAP FILE cannot open the file.
X$C_BITMAP_NO_MEMORY	BitmapNoMemory	There is not enough memory to load the bitmap file.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

filename

The name of the file in which WRITE BITMAP FILE writes the bitmap. The format of the file is dependent on the operating system on the client side of the client-server connection. VMS logical names, search strings, and so on are supported. The maximum length of a file specification is 255 bytes. Wildcards are not supported. The default file name is [ ]bitmap.dat.

VAX only

The **filename** argument is the address of a character string descriptor that points to the string.

MIT C only

The **filename** argument is a pointer to a null-terminated character string.

bitmap_id

The bitmap that you want to write to a file.

width

The width of the bitmap to be written.

height

The height of the bitmap to be written.

x_hot_coord

The x-coordinate at which to place the hotspot, which is defined as the point in the cursor that corresponds to the x- and y-coordinates reported for the pointer. If **x_hot_coord** and **y_hot_coord** are not (–1, –1), WRITE BITMAP FILE writes them out as the hotspot coordinates for the bitmap.

y_hot_coord

The y-coordinate at which to place the hotspot, which is defined as the point in the cursor that corresponds to the x- and y-coordinates reported for the pointer. If **x_hot_coord** and **y_hot_coord** are not (–1, –1), WRITE BITMAP FILE writes them out as the hotspot coordinates for the bitmap.

DESCRIPTION

WRITE BITMAP FILE writes out a bitmap to the file that you specify. See the READ BITMAP FILE routine.

12 Color Routines

There are two basic concepts to understand in order to work with color, as follows:

- The type of color device you are working with and its associated color map data structure
- Allocating and defining colors in color maps

The Xlib color routines perform the following operations:

- Creating and manipulating color maps

  For most color implementations, the default color map provides adequate resources. For unusual implementations, you might consider creating and using your own color map.
- Allocating color definitions from the color map
- Defining colors
- Specifying the red, green, and blue values of a specific color

This chapter covers how to use the color routines to accomplish color tasks. For concepts related to color routines and information on how to use color routines, see the *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 12–1.

Table 12–1 Color Routines

Routine Name	Description
ALLOC COLOR	Returns the shareable color index for a given color definition. The routine also returns the color definition closest to the one specified that can be supported by the hardware.
ALLOC COLOR CELLS	Allocates reserved color definitions from the color map using a pseudocolor model.
ALLOC COLOR PLANES	Allocates the specified number of entries and planes from the color map for a direct color type device.
ALLOC NAMED COLOR	Returns the shareable color index for a given named color. The routine also returns the color definition closest to the one specified that can be supported by the hardware.

(continued on next page)

Table 12–1 (Cont.) Color Routines

Routine Name	Description
COPY COLORMAP AND FREE	Creates a color map with the same allocations for a client as exist in the current color map. Other color definitions in the new color map are undefined. The definitions allocated by the client in the current color map are freed.
CREATE COLORMAP	Creates a color map and returns a color map identifier. The entries in the color map are undefined.
FREE COLORMAP	Deletes the specified color map.
FREE COLORS	Deallocates the specified color index or plane.
GET STANDARD COLORMAP	Obtains a standard color map that may have been specified by another client.
GET VISUAL INFO	Obtains a list of visual information structures that match a specified template.
LOOKUP COLOR	Obtains the color values for a specified color name.
MATCH VISUAL INFO	Obtains a visual that can be used with a specified screen, depth, and class.
QUERY COLOR	Obtains the red, green, and blue values for the specified color index.
QUERY COLORS	Obtains the red, green, and blue values for each color index specified.
SET STANDARD COLORMAP	Specifies a standard color map.
SET WINDOW COLORMAP	Specifies the current color map for the specified window.
STORE COLOR	Sets a color map entry, previously allocated, to a specified color.
STORE COLORS	Sets more than one color map entry, previously allocated, to a specified color.
STORE NAMED COLOR	Sets the specified color map entry to the named color.

12.1 Standard Color Map Data Structure

Standard color map routines are provided to allow clients to exchange information about color maps. This information is stored in a window property, formatted as a standard colormap data structure.

There are six predefined color map properties. Each property uses the predefined property type of RGB_COLOR_MAP.

Note that VMS DECwindows software does not currently make use of standard color maps.

The standard colormap data structure for the VAX binding is shown in Figure 12–1, and members of the data structure are described in Table 12–2.

Figure 12–1 Standard Color Map Data Structure (VAX Binding)

x$l_scmp_colormap	0
x$l_scmp_red_max	4
x$l_scmp_red_mult	8
x$l_scmp_green_max	12
x$l_scmp_green_mult	16
x$l_scmp_blue_max	20
x$l_scmp_blue_mult	24
x$l_scmp_base_pixel	28

Table 12–2 Members of the Standard Color Map Data Structure (VAX Binding)

Member Name	Contents
X$L_SCMP_COLORMAP	A color map identifier returned by CREATE COLORMAP.
X$L_SCMP_RED_MAX, X$L_SCMP_GREEN_MAX, X$L_SCMP_BLUE_MAX	The maximum number of red, green, and blue values that are being used. The value actually used can range from 0 to the maximum specified in these members and is referred to as the *coefficient*. The coefficient is used with the multipliers (red_mult, green_mult, and blue_mult) and base_pixel to compute a full color index. For example, on an 8-plane display with 3 planes allocated for red, 3 planes for green and 2 planes for blue, red_max is 7, green_max is 7, and blue_max is 3.
X$L_SCMP_RED_MULT, X$L_SCMP_GREEN_MULT, X$L_SCMP_BLUE_MULT	The scale factors used to compose a full color index. For example, on an 8-plane display with 3 planes allocated for red, 3 planes allocated for green, and 2 planes allocated for blue, red_mult could be 32, green_mult could be 4, and blue_mult could be 1.

(continued on next page)

Table 12–2 (Cont.) Members of the Standard Color Map Data Structure (VAX Binding)

Member Name	Contents

X$L_SCMP_BASE_PIXEL	The base color index used to compose a full color index. The color index is returned by ALLOC COLOR PLANES.
	The equation to compute a full color index from the coefficients, the scale factors, and the base pixel is as follows:
	`(red_coefficient * red_mult)`
	`+ (green_coefficient * green_mult)`
	`+ (blue_coefficient * blue_mult)`
	`+ base_pixel`

The standard color map data structure for the MIT C binding is shown in Figure 12–2, and members of the data structure are described in Table 12–3.

Figure 12–2 Standard Color Map Data Structure (MIT C Binding)

```
typedef struc {
        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;
```

Table 12–3 Members of the Standard Color Map Data Structure (MIT C Binding)

Member Name	Contents
colormap	A color map identifier returned by CREATE COLORMAP.
red_max, green_max, blue_max	The maximum number of red, green, and blue values that are being used. The value actually used can range from 0 to the maximum specified in these members and is referred to as the *coefficient*. The coefficient is used with the multipliers (red_mult, green_mult, and blue_mult) and base_pixel to compute a full color index. For example, on an 8-plane display with 3 planes allocated for red, 3 planes for green and 2 planes for blue; red_max is 7, green_max is 7, and blue_max is 3.

(continued on next page)

Table 12–3 (Cont.) Members of the Standard Color Map Data Structure (MIT C Binding)

Member Name	Contents
red_mult, green_mult, blue_mult	The scale factors used to compose a full color index. For example, on an 8-plane display with 3 planes allocated for red, 3 planes allocated for green, and 2 planes allocated for blue; red_mult could be 32, green_mult could be 4, and blue_mult could be 1.

base_pixel	The base color index used to compose a full color index. The color index is returned by ALLOC COLOR PLANES.
	The equation to compute a full color index from the coefficients, the scale factors, and the base pixel is as follows:
	`(red_coefficient * red_mult)`
	`+ (green_coefficient * green_mult)`
	`+ (blue_coefficient * blue_mult)`
	`+ base_pixel`

12.2 Color Definition Data Structure

The routines use a color definition data structure to specify and receive the following information about a color:

- The color index for the color definition (pixel)
- The red, green, and blue values
- A flag that identifies whether to refer to the red, green, or blue values (relevant for direct color and true color visual types)

Use the data structure as follows:

- When you use a shared color, use the data structure to specify the red, green, and blue values you want with the ALLOC COLOR routine. This routine returns the color index in the pixel member that points to the color definition with the closest color to yours that can be displayed on the screen. The flag field is ignored.
- When you use a reserved color, use STORE COLOR or STORE COLORS to specify in the data structure the red, green and blue values for the color you want. You also supply the color index that was previously returned by ALLOC COLOR CELLS or ALLOC COLOR PLANES. You use the flags to specify which part of the color specification should actually be changed.
- When you want to know what color is stored in a particular color definition, use QUERY COLOR or QUERY COLORS to specify the color index in the pixel member of the data structure. QUERY COLOR or QUERY COLORS returns the color values in the red, green, and blue members.

- When you want to know the color values for a particular color name (such as red), use LOOKUP COLOR to specify the color name. LOOKUP COLOR returns the color values in the red, green and blue members of the data structure.

The color definition data structure for the VAX binding is shown in Figure 12–3, and members of the data structure are described in Table 12–4.

Figure 12–3 Color Definition Data Structure (VAX Binding)

<table>
<tr><td colspan="3">x$l_colr_pixel</td><td>0</td></tr>
<tr><td colspan="2">x$w_colr_green</td><td>x$w_colr_red</td><td>4</td></tr>
<tr><td>x$b_colr_pad</td><td>x$b_colr_flags</td><td>x$w_colr_blue</td><td>8</td></tr>
</table>

Table 12–4 Members of the Color Definition Data Structure (VAX Binding)

Member Name	Contents	
X$L_COLR_PIXEL	Defines a pixel value.	
X$W_COLR_RED	Defines the red value of the pixel.[1]	
X$W_COLR_GREEN	Defines the green value of the pixel.[1]	
X$W_COLR_BLUE	Defines the blue value of the pixel.[1]	
X$B_COLR_FLAGS	Defines which color members are to be defined in the color map. Possible flags are as follows:	
	X$M_DO_RED	Sets red values
	X$M_DO_GREEN	Sets green values
	X$M_DO_BLUE	Sets blue values
X$B_COLR_PAD	Makes the structure an even length.	

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

The color definition data structure for the MIT C binding is shown in Figure 12–4, and members of the data structure are described in Table 12–5.

Figure 12–4 Color Definition Data Structure (MIT C Binding)

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

Table 12–5 Members of the Color Definition Data Structure (MIT C Binding)

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

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

12.3 Color Routines

The following pages describe the Xlib color routines.

ALLOC COLOR

Returns the shareable color index for a given color definition. The routine also returns the color definition closest to the one specified that can be supported by the hardware.

VAX FORMAT

status_return = **X$ALLOC_COLOR** *(display, colormap_id, screen_def_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
screen_def_return	record	x$color	read/write	reference

MIT C FORMAT

*status_return=***XAllocColor** *(display, colormap_id, screen_def_return)*

argument information

```
Status XAllocColor(display, colormap_id, screen_def_return)
      Display *display;
      Colormap colormap_id;
      XColor *screen_def_return;
```

RETURNS

status_return

Specifies whether or not the routine completed successfully. If there was a problem, usually because of lack of resources, **status_return** is zero. If the routine was successful, **status_return** is nonzero.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map where the color definition will be stored. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

screen_def_return

The color definition data structure that defines the color requested and that receives the exact color and the index that were allocated. The red, green, and blue color values of the color requested are specified in this data structure. The color index (pixel value) is returned in the pixel member of the data structure. The red, green, and blue values of the closest color supported by the hardware are returned in the red, green, and blue members.

The color definition data structure is described in Section 12.2.

DESCRIPTION

ALLOC COLOR returns the color index in the color definition data structure. ALLOC COLOR stores the color definition closest to the one you specified that can be supported by the hardware. It stores the definition in the specified color map and returns its color index.

You specify the red, green, and blue values for the color that you want in the red, green, and blue members of the color definition data structure. The routine returns the red, green, and blue values that are closest to those you specified that can be supported by the hardware. The red, green, and blue values you specified in the color definition data structure are overwritten by the returned values.

The entry identified by ALLOC COLOR is a read-only shared color definition. After you specify the color, you cannot change it. Subsequent calls to ALLOC COLOR for the same color by any client will return the same pixel value. The color definition is not deallocated until the last client using it has deallocated it.

To allocate an entry for your exclusive use, use ALLOC COLOR CELLS or ALLOC COLOR PLANES.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.

ALLOC COLOR CELLS

Allocates reserved color indexes from the color map using a pseudocolor model.

VAX FORMAT

status_return = **X$ALLOC_COLOR_CELLS** *(display, colormap_id, contig, plane_masks_return, num_planes, pixels_return, num_colors)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
contig	Boolean	longword	read	reference
plane_masks_return	array	longword	write	reference
num_planes	longword	longword	read	reference
pixels_return	array	longword	write	reference
num_colors	longword	longword	read	reference

MIT C FORMAT

status_return = **XAllocColorCells** *(display, colormap_id, contig, plane_masks_return, num_planes, pixels_return, num_colors)*

argument information

```
Status XAllocColorCells(display, colormap_id, contig,
plane_masks_return, num_planes, pixels_return, num_colors)
      Display *display;
      Colormap colormap_id;
      Bool contig;
      unsigned long plane_masks_return[];
      unsigned int num_planes;
      unsigned long pixels_return[];
      unsigned int num_colors;
```

RETURNS

status_return

Return value that specifies whether or not the routine completed successfully. If there was a problem, usually because of lack of server resources or of available color cells in the color map, **status_return** is zero. If the routine was successful, **status_return** is nonzero.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map to allocate entries from. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

contig

Specifies whether the bits in the plane mask are contiguous. When true, the bits are contiguous. When false, the bits may not be contiguous.

plane_masks_return

A pointer to an array of plane masks in which each element contains a plane mask returned by the routine. The length of the array is specified by **num_planes**. No mask has any bits in common with any other mask, or with any of the color indexes.

num_planes

The number of planes requested. This value must be nonnegative. Specify 0 for no planes. The number of planes allocated must be supported by the device you are working with. The **num_planes** argument specifies the number of plane masks that is returned in **plane_masks_return** and thereby specifies the length of that array.

pixels_return

A pointer to an array of color indexes in which each element is a color index, returned by the routine. The color index (pixel value) is an index into the color map for the color definition allocated for the program's use. The color definition allocated is read/write. The length of the array is specified by **num_colors**.

num_colors

The number of color indexes to be set in the color map. This value specifies the number of data structures and thereby specifies the length of **pixels_return** and thereby the length of that array.

DESCRIPTION

ALLOC COLOR CELLS allocates color indexes and color planes for reserved use. Use this routine primarily for pseudocolor type devices, or when you want the screen to act like a pseudocolor device.

The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

The color definitions allocated are read and write entries. The colors are not defined. You can define the colors for these entries with STORE COLOR, STORE COLORS, or STORE NAMED COLOR.

When you allocate both color definitions and planes, you must combine the planes and indexes to get actual pixel values. The actual pixel values you have allocated can be determined by taking each pixel value and combining it with all possible combinations of plane masks. The actual number of color definitions allocated can be computed by (**num_colors**$*2^{\mathbf{num\_planes}}$). The bits in the color index do not have any bits in common with any of the plane masks or any other color index.

For example, if you wanted to allocate one color map entry (**num_colors** equals 1) and two planes (**num_planes** equals 2), the following might be returned:

- Color index = 6 (00110)
- Plane mask 1 = 8 (01000)
- Plane mask 2 = 16 (10000)

The color definitions allocated would be as follows:

- Index 6 (pixel value)
- Index 14 (pixel-value OR plane-mask-1 is 0000 1110)
- Index 22 (pixel-value OR plane-mask-2 is 0001 0110)
- Index 30 (pixel-value OR plane-mask-1 OR plane-mask-2 is 0001 1110)

Four color definitions are reserved (or 1 pixel-value * $2^{2\text{planes}}$).

You ask for contiguous planes (**contig** is true) if you want to perform arithmetic on the plane values. If you do not want to do calculations, then the planes can be noncontiguous (**contig** is false).

For gray-scale and pseudocolor visual types, each plane mask has one bit. For direct color, each plane mask has three bits. When **contig** is true and the logical OR operation of the masks is calculated, a single contiguous set of bits is formed for gray-scale, pseudocolor, or visual types, and three contiguous sets of bits (one within each pixel submember) for direct color.

If you have a direct color type device, or you want the screen to act like a direct color device, use ALLOC COLOR PLANES to allocate color definitions and planes.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

ALLOC COLOR PLANES

Allocates the specified number of entries and planes from the color map for a direct color type device.

VAX FORMAT

status_return = **X$ALLOC_COLOR_PLANES** *(display, colormap_id, contig, pixels_return, num_colors, num_reds, num_greens, num_blues, rmask_return, gmask_return, bmask_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
contig	Boolean	uns longword	read	reference
pixels_return	array	longword	write	reference
num_colors	longword	longword	read	reference
num_reds	longword	longword	read	reference
num_greens	longword	longword	read	reference
num_blues	longword	longword	read	reference
rmask_return	mask_longword	uns longword	write	reference
gmask_return	mask_longword	uns longword	write	reference
bmask_return	mask_longword	uns longword	write	reference

MIT C FORMAT

status_return = **XAllocColorPlanes** *(display, colormap_id, contig, pixels_return, num_colors, num_reds, num_greens, num_blues, pixels_return, rmask_return, gmask_return, bmask_return)*

argument information

```
Status XAllocColorPlanes(display, colormap_id, contig,
pixels_return, num_colors, num_reds, num_greens, num_blues,
pixels_return, rmask_return, gmask_return, bmask_return)
      Display *display;
      Colormap colormap_id;
      Bool contig;
      unsigned long pixels_return[];
      int num_colors;
      int num_reds, num_greens, num_blues;
      unsigned long *rmask_return, *gmask_return, *bmask_return;
```

RETURNS

status_return

Specifies whether or not the routine completed successfully. If there was a problem, usually because of lack of server resources or of available color cells in the color map, **status_return** is zero. If the routine was successful, **status_return** is nonzero.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map to allocate entries from. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

contig

Boolean argument that specifies whether the bits in each plane mask must be contiguous. When true, the bits in each plane mask must be contiguous. When false, the bits need not be contiguous.

pixels_return

A pointer to an array of color indexes where each element is a color index returned by the routine. The color index (pixel value) is an index into the color map for the color definition allocated for the program's use. The length of the array is specified by **num_colors**.

num_colors

The number of color definitions to be set in the color map. This value specifies the number of data structures and thereby specifies the length of **pixels_return** and thereby the length of that array.

num_reds

The number of planes required to control red values. This value must be nonnegative.

num_greens

The number of planes required to control green values. This value must be nonnegative.

num_blues

The number of planes required to control blue values. This value must be nonnegative.

rmask_return

The mask that identifies the planes allocated for the red values.

gmask_return

The mask that identifies the planes allocated for the green values.

bmask_return

The mask that identifies the planes allocated for the blue values.

DESCRIPTION

ALLOC COLOR PLANES allocates reserved color definitions and color planes for a direct color type device. Use this routine primarily for direct color type devices, or when you want the screen to act as a direct color device.

The color definitions allocated are read and write entries. The colors are not defined. You can define the colors for these entries with STORE COLOR, STORE COLORS, or STORE NAMED COLOR.

Specify the number of color definitions you want to allocate in the **num_colors** argument. For each color definition requested, a color index into the color map is returned in **pixels_return**.

To allocate color planes, use the **num_reds**, **num_greens**, and **num_blues** arguments. For each color member, a mask is returned that identifies the planes which have been allocated for the red, green, and blue members.

Each plane mask lies within the corresponding portion of the color index that points to the same color member. For example, the red plane mask lies within the portion of the color index that points to the red structure. If the logical OR operation of subsets of masks with color indexes is calculated, different indexes are derived. You can reference and use each of these indexes. The following number of indexes can be produced:

$$num\_colors * \left(2^{num\_reds+num\_greens+num\_blues}\right)$$

In the color map there are only $num\_colors * 2^{num\_reds}$ independent red entries, $num\_colors * 2^{num\_greens}$ independent green entries, and $num\_colors * 2^{num\_blues}$ independent blue entries, even for a device that is really a pseudocolor device.

If you change the color definition within a color map (using STORE COLOR, STORE COLORS, or STORE NAMED COLOR), its index is decomposed according to the masks, and the corresponding independent entries are updated. If you have a pseudocolor device, or you want the screen to act as a pseudocolor device, use ALLOC COLOR CELLS to allocate color definitions and planes.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

ALLOC NAMED COLOR

Returns the shareable color index for a given named color. The routine also returns the color definition closest to the one specified that can be supported by the hardware.

VAX FORMAT

status_return = **X$ALLOC_NAMED_COLOR** *(display, colormap_id, color_name [,screen_def_return] [,exact_def_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
color_name	char string	char string	read	descriptor
screen_def_return	record	x$color	write	reference
exact_def_return	record	x$color	write	reference

MIT C FORMAT

status_return = **XAllocNamedColor** *(display, colormap_id, color_name, screen_def_return, exact_def_return)*

argument information

```
Status XAllocNamedColor(display, colormap_id, color_name,
screen_def_return, exact_def_return)
      Display *display;
      Colormap colormap_id;
      char *color_name;
      XColor *screen_def_return, *exact_def_return;
```

RETURNS

status_return

Specifies whether or not the routine completed successfully. If there was a problem, usually because of lack of server resources or of available color cells in the color map, **status_return** is zero. If the routine was successful, **status_return** is nonzero.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map where the requested color will be stored. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

color_name

The name of the requested color. The color name you specify must be supported by the color database. Use the ISO Latin-1 encoding. Uppercase or lowercase characters do not matter.

C only

The **color_name** argument is a null-terminated character string.

screen_def_return

The color definition data structure that returns the color definition of the color actually used by the server. The color definition and the color index are returned in the color definition data structure.

For more information about the color definition data structure, see Section 12.2.

VAX only

This argument is optional.

exact_def_return

The color definition data structure that defines the exact color as specified in the color database. The color definition is returned in the color definition data structure. The color definition data structure is shown in Section 12.2.

VAX only

This argument is optional.

DESCRIPTION

ALLOC NAMED COLOR provides two color definitions (red, green, and blue values) for the color name you specify:

- The exact color definition as it is defined in the color database.
- The closest color definition available for the hardware you are using. The color definition is read-only and can be shared among application programs.

ALLOC NAMED COLOR also provides the color index where the closest color definition is stored.

Because the color definition may vary between the exact definition and the closest definition available, use this routine to determine what the variation is.

To obtain the color definition data, you supply the name of the color as a text string. A database that associates the color name with the correct color structure is maintained by the server to resolve the correct definition for the color name string.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_NAME	BadName	The font or color that you specified does not exist.

COPY COLORMAP AND FREE

Creates a color map with the same allocations for a client as exist in the specified color map. Other color definitions in the new color map are undefined. The definitions allocated by the client in the current color map are freed.

VAX FORMAT

X$COPY_COLORMAP_AND_FREE
(display, colormap_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
colormap_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference

MIT C FORMAT

colormap_id_return = **XCopyColormapAndFree**
(display, colormap_id)

argument information

```
Colormap XCopyColormapAndFree(display, colormap_id)
      Display *display;
      Colormap colormap_id;
```

RETURNS

colormap_id_return

The identifier of the new color map.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map to be freed. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE. Color definitions that have been reserved in this map using the specified display are copied to the new color map. Then the reserved color definitions in the current color map are freed.

DESCRIPTION

COPY COLORMAP AND FREE creates a new color map and returns its identifier in **colormap**. The new color map is of the same visual type and for the same screen as the current color map (**colormap_id**).

The new color map has the same color definitions reserved for the client program as were reserved for the client, with ALLOC COLOR CELLS or ALLOC COLOR PLANES, in the current color map (**colormap_id**). All color definitions specified for the reserved colors are the same in the new color map. Any other color definitions in the new color map are undefined. The reserved color definitions in the current color map (**colormap_id**) are deallocated.

You are likely to use this routine when there are no more color definitions available for reserved use in the default color map. This routine creates a private color map for your use.

This routine does not dissociate the current color map identifier (**colormap_id**) from the current color map. To dissociate a color map, use the FREE COLORMAP routine.

The current color map identifier was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.

CREATE COLORMAP

Creates a color map and returns a color map identifier. The entries in the color map are undefined.

VAX FORMAT

colormap_id_return = **X$CREATE_COLORMAP** *(display, window_id, visual_struc, alloc)*

argument information

Argument	Usage	Data Type	Access	Mechanism
colormap_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
visual_struc	record	x$visual	read	reference
alloc	longword	longword	read	reference

MIT C FORMAT

colormap_id_return = **XCreateColormap** *(display, window_id, visual_struc, alloc)*

argument information

```
Colormap XCreateColormap(display, window_id, visual_struc, alloc)
      Display *display;
      Window window_id;
      Visual *visual_struc;
      int alloc;
```

RETURNS

colormap_id_return

The identifier of the new color map.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window that identifies the root for which the color map will be created.

visual_struc

A pointer to a visual structure associated with the window.

alloc

The allocation mode of color map entries. The values for **alloc** are as follows:

VAX	C	Description
X$C_ALLOC_NONE	AllocNone	No entries are preallocated.
X$C_ALLOC_ALL	AllocAll	All entries were preallocated to the current client. No more entries can be allocated from the color map by any client.

Other values specified in this argument are not valid.

If static gray, static color, or true color is specified in **visual_struc**, no entries can be allocated from the color map.

DESCRIPTION

CREATE COLORMAP creates a private color map and returns a color map identifier. For most applications, you do not need to use a private color map; the default color map has sufficient resources. However, for some applications, you use this routine to create a private color map for your program's use.

After the color map identifier is returned, use this identifier to refer to the color map in any subsequent routines. The contents of the new colormap are undefined for the visual types gray scale, pseudocolor, or direct color.

After you create the color map for the visual types gray scale, pseudocolor, or direct color, you can specify colors within it. If the **alloc** argument requires you to allocate (reserve) color definitions (that is, **alloc** is Alloc None), you use the ALLOC COLOR CELLS or ALLOC COLOR PLANES routines. Then use STORE COLOR, STORE COLORS, or STORE NAMED COLOR to specify color definitions.

If the **alloc** argument specified that all entries are allocated to you, (in other words, **alloc** is Alloc All), use STORE COLOR or STORE NAMED COLOR to define any indexes in the color map.

Use the DEFAULT VISUAL routine to determine the visual type of the screen.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested entries.
X$C_BAD_MATCH	BadMatch	The specified visual type does not match the specified window.

VAX	C	Description
X$C_BAD_VALUE	BadValue	Either **visual_struc** or **alloc** has an incorrect value.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

FREE COLORMAP

Deletes the specified color map.

VAX FORMAT

X$FREE_COLORMAP *(display, colormap_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference

MIT C FORMAT

XFreeColormap *(display, colormap_id)*

argument information

```
XFreeColormap(display, colormap_id)
      Display *display;
      Colormap colormap_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map to be freed. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

DESCRIPTION

FREE COLORMAP deletes the color map. However, it cannot free the default color map for a screen.

If the color map is an installed map, it is uninstalled and another color map is installed. If **colormap_id** is currently the color map attribute of a window (set using CREATE WINDOW or CHANGE WINDOW ATTRIBUTES), the color map attribute for the window is changed to None and a Color Map Notify event is generated.

X ERRORS

VAX	C	Description
X$C_BAD_COLOR	BadColor	The value that you specified for the color map argument does not name a defined color map.

FREE COLORS

Deallocates the specified color definition or plane.

VAX FORMAT

X$FREE_COLORS
(display, colormap_id, pixels, num_pixels, planes)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
pixels	array	longword	read	reference
num_pixels	longword	longword	read	reference
planes	longword	uns longword	read	reference

MIT C FORMAT

XFreeColors
(display, colormap_id, pixels, num_pixels, planes)

argument information

```
XFreeColors(display, colormap_id, pixels, num_pixels, planes)
      Display *display;
      Colormap colormap_id;
      unsigned long pixels[];
      int num_pixels;
      unsigned long planes;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map from which the color definitions and planes will be deallocated. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

pixels

A pointer to an array of color indexes (pixel values) where each element points to a color definition in the color map to be freed. The length of the array is specified by **num_pixels**.

num_pixels

The number of color definitions in the color map to be freed. This value specifies the number of elements in **pixels**.

planes

The planes that are to be freed. The logical OR of the planes to be freed is calculated. If there are no planes to be freed, this argument must be zero.

DESCRIPTION

FREE COLORS deallocates color definitions in the color map and frees planes. After FREE COLORS is used, the color definitions that are freed are available for allocation. However, a color definition originally allocated with ALLOC COLOR PLANES is not completely freed until all related color definitions are also freed.

If you incorrectly specified a color index, all other specified color definitions are freed anyway.

The color indexes (pixel values) and planes were originally returned when you allocated color definitions and planes with ALLOC COLOR, ALLOC COLOR CELLS, ALLOC COLOR PLANES, or ALLOC NAMED COLOR.

X ERRORS

VAX	C	Description
X$C_BAD_ACCESS	BadAccess	You attempted to free a color map entry that you did not allocate.
X$C_BAD_COLOR	BadColor	The value that you specified for a color map argument does not name a defined color map.
X$C_BAD_VALUE	BadValue	The planes or pixels you specified fall outside the available range.

GET STANDARD COLORMAP

Obtains a standard color map.

VAX FORMAT

status_return = **X$GET_STANDARD_COLORMAP** *(display, window_id, standard_colormap_return, property_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
standard_ colormap_return	record	x$standard_ colormap	write	reference
property_id	identifier	uns longword	read	reference

MIT C FORMAT

status_return = **XGetStandardColormap** *(display, window_id, standard_colormap_return, property_id)*

argument information

```
Status XGetStandardColormap(display, window_id,
                            standard_colormap_return,
                            property_id)
      Display *display;
      Window window_id;
      XStandardColormap *standard_colormap_return;
      Atom property_id; /* RGB_BEST_MAP, etc. */
```

RETURNS

status_return

Return value that specifies whether or not the routine completed successfully. The routine fails if the standard color map property has not been defined for the specified window.

C only

This argument returns zero if the routine completes successfully, and nonzero if it does not complete successfully.

VAX only

This argument returns one of the following values.

Value	Description
SS$_NORMAL	Routine completed successfully.
X$_PROPUNDEF	The standard color map property has not been defined for the specified window.
X$_ERRORREPLY	Error received from the server.
X$_TRUNCATED	The buffer is not big enough, therefore the results are truncated.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window that the standard color map properties are attached to. The root window usually has the standard color map properties.

standard_colormap_return

The standard color map data structure that defines the standard color map is returned by the routine.

For more information about the standard color map data structure, see Section 12.1.

property_id

The identifier of the property type for the standard color map to be obtained. A property type is specified by an atom. There are six predefined property types for standard color maps:

- RGB_DEFAULT_MAP
- RGB_BEST_MAP
- RGB_RED_MAP
- RGB_GREEN_MAP
- RGB_BLUE_MAP
- RGB_GRAY_MAP

DESCRIPTION

GET STANDARD COLORMAP obtains a predefined standard color map that has been stored on a window by another client.

X ERRORS

VAX	C	Description
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

GET VISUAL INFO

Obtains a list of visual information structures that match a specified template.

VAX FORMAT

status_return = **X$GET_VISUAL_INFO**
(display, vinfo_mask, vinfo_template,
num_items_return [,items_return] [,items_size]
[,items_buff_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
vinfo_mask	mask_longword	uns longword	read	reference
vinfo_template	record	x$visual_info	read	reference
num_items_return	longword	longword	write	reference
items_return	address	uns longword	write	reference
items_size	longword	longword	read	reference
items_buff_return	record	uns longword	read	reference

MIT C FORMAT

XVisualInfo = **XGetVisualInfo**
(display, vinfo_mask, vinfo_template,
num_items_return)

argument information

```
XVisualInfo *XGetVisualInfo(display, vinfo_mask, vinfo_template,
num_items_return)
      Display *display;
      long vinfo_mask;
      XVisualInfo *vinfo_template;
      int *num_items_return;
```

RETURNS

status_return (VAX only)

Specifies whether or not the routine completed successfully.

XVisualInfo (MIT C only)

A list of visual information structures that match the specified template.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

vinfo_mask

A bit mask that specifies which visual mask attributes this routine will try to match.

Table 12–6 lists each bit for **vinfo_mask**, its predefined value, and its description.

Table 12–6 Visual Information Mask Bits

Predefined Bit Value	Meaning When Set
VisualNoMask	Use no mask attributes
VisualIDMask	Use the identifier attribute
VisualScreenMask	Use the screen attribute
VisualDepthMask	Use the depth attribute
VisualClassMask	Use the class attribute
VisualRedMaskMask	Use the red mask attribute
VisualGreenMaskMask	Use the green mask attribute
VisualBlueMaskMask	Use the blue mask attribute
VisualColormapSizeMask	Use the color map size attribute
VisualBitsPerRGBMask	Use the bits-per-RGB attribute
VisualAllMask	Use all attributes

vinfo_template

A template of visual attributes that are to be used in matching the visual information structures. Each attribute corresponds to a bit that is set in the **vinfo_mask**. See the description section for information on the visual information structure.

num_items_return

The number of matching visual information structures in **items_return**.

items_return (VAX only)

The virtual address of a pointer to an array of visual information data returned by the routine and residing in space reserved by Xlib.

items_size (VAX only)

The size of the buffer specified in **items_buff_return**.

items_buff_return (VAX only)

A pointer to a data buffer, residing in space you have reserved, where each entry is one visual information element. The size of the buffer is specified by **items_size**.

DESCRIPTION

GET VISUAL INFO obtains a list of visual information structures that match the attributes specified by a template. If there are no visual information structures that match the template, GET VISUAL INFO returns a null value.

To specify arguments that describe the visual information data returned by the routine, use **items_return** to access data owned by Xlib, or **items_size** and **items_buff_return** to obtain a private copy of the data.

LOOKUP COLOR

Obtains the color values for a specified color name.

VAX FORMAT

status_return = **X$LOOKUP_COLOR** *(display, colormap_id, color_name [,screen_def_return] [,exact_def_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
color_name	char string	char string	read	descriptor
screen_def_return	record	x$color	write	reference
exact_def_return	record	x$color	write	reference

MIT C FORMAT

status_return = **XLookupColor** *(display, colormap_id, color_name, screen_def_return, exact_def_return)*

argument information

```
Status XLookupColor(display, colormap_id, color_name,
                    screen_def_return, exact_def_return)
      Display *display;
      Colormap colormap_id;
      char *color_name;
      XColor *screen_def_return, *exact_def_return;
```

RETURNS

status_return

Return value that specifies whether or not the routine completed successfully. If there was a problem, usually because of lack of resources, **status_return** is zero. If the routine was successful, **status_return** is nonzero.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map that the server will use to compute **screen_def_return**. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

color_name

The name of the requested color. The color name you specify must be supported by the color database. Use the ISO Latin-1 encoding. Uppercase or lowercase characters do not matter.

C only

The **color_name** argument is a null-terminated character string.

screen_def_return

The color definition data structure where the red, green, and blue values of the closest color supported by the screen hardware are returned. The screen is determined from the specified color map.

For more information about the color definition data structure, see Section 12.2.

VAX only

This argument is optional.

exact_def_return

The color definition data structure where the red, green, and blue values of the exact color as defined in the color database are returned. The pixel, pad, and flags members are not used.

For more information about the color definition data structure, see Section 12.2.

VAX only

This argument is optional.

DESCRIPTION

LOOKUP COLOR obtains the color values for the specified color name. To obtain the color definition data, you supply the name of the color as a text string. A database that associates the color name with the correct color structure is maintained by the server to resolve the correct definition for the color name string.

The routine returns the color definition for the exact color, as maintained in the color database, and the color definition for the closest color that can be supported by the screen associated with the specified color map.

X ERRORS

VAX	C	Description
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_NAME	BadName	The color that you specified does not exist.

MATCH VISUAL INFO

Obtains visual information that can be used with a specified screen, depth, and class.

VAX FORMAT

status_return = **X$MATCH_VISUAL_INFO** *(display, screen_number, depth, class, vinfo_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
screen_number	uns longword	uns longword	read	reference
depth	longword	longword	read	value
class	longword	longword	read	value
vinfo_return	record	x$visual_info	write	reference

MIT C FORMAT

status_return = **XMatchVisualInfo** *(display, screen_number, depth, class, vinfo_return)*

argument information

```
Status XMatchVisualInfo(display, screen_number, depth, class,
                        vinfo_return)
      Display *display;
      int screen_number;
      int depth;
      int class;
      XVisualInfo *vinfo_return;
```

RETURNS

status_return

Return value that specifies whether or not the routine completed successfully.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

screen_number

The number of the screen for which the visual information is to be obtained.

depth

The depth to be matched on the specified screen.

class

The class to be matched on the specified screen.

vinfo_return

A pointer to which MATCH VISUAL INFO returns the matching visual information for the specified class and screen.

DESCRIPTION

MATCH VISUAL INFO obtains the visual information for a visual that matches a specified screen depth and class for a screen. Multiple visuals that match a specified depth and class can exist; in this case, the exact visual chosen by MATCH VISUAL INFO is undefined.

If a matching visual exists, MATCH VISUAL INFO returns true and the information on the visual is returned to the **vinfo_return** argument. If a matching visual is not found, MATCH VISUAL INFO returns false.

QUERY COLOR

Provides the red, green, and blue values for the index specified in the color definition data structure.

VAX FORMAT

X$QUERY_COLOR
(display, colormap_id, screen_def_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
screen_def_return	record	x$color	read/write	reference

MIT C FORMAT

XQueryColor
(display, colormap_id, screen_def_return)

argument information

```
XQueryColor(display, colormap_id, screen_def_return)
      Display *display;
      Colormap colormap_id;
      XColor *screen_def_return;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map containing the specified color definition. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

screen_def_return

The color definition data structure for the specified color definition. The color index is specified in the pixel member of the data structure. The color values stored at the color definition referenced by the color index are returned in the red, green, and blue members. The flags member is ignored.

When the red, green, and blue values are returned by the routine, they overwrite any current values specified in the data structure.

For more information about the color definition data structure, see Section 12.2.

DESCRIPTION

QUERY COLOR provides the color definition for a specified color index. The color index is specified in the pixel member of the color definition data structure. The red, green, and blue values are returned in the same data structure for the specified color index. The flags member in the data structure is ignored.

If the color definition specified in **screen_def_return** is unallocated, the red, green, and blue values returned are undefined.

When you want the color definitions for more than one color index, use the QUERY COLORS routine.

X ERRORS

VAX	C	Description
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_VALUE	BadValue	The index you specified lies outside the range of the colormap.

QUERY COLORS

Provides the red, green, and blue values for each color index specified.

VAX FORMAT

X$QUERY_COLORS *(display, colormap_id, screen_defs_return, num_colors)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
screen_defs_return	record	x$color	read/write	reference
num_colors	longword	longword	read	reference

MIT C FORMAT

XQueryColors *(display, colormap_id, screen_defs_return, num_colors)*

argument information

```
XQueryColors(display, colormap_id, screen_defs_return,
             num_colors)
      Display *display;
      Colormap colormap_id;
      XColor screen_defs_return[];
      int num_colors;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map containing the specified color indexes. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

screen_defs_return

An array of color definition data structures where each element in the array defines the color for one color index. There is one array entry for each color index requested by **num_colors**.

The color indexes are specified in the pixel member of each color definition data structure. The color values are returned in the red, green, and blue members of each data structure. The flags member in each data structure is ignored.

For more information about the color definition data structure, see Section 12.2.

num_colors

The number of color indexes to be set in the color map. This value specifies the number of data structures and thereby specifies the length of the **screen_defs_return** array.

DESCRIPTION

QUERY COLORS provides the color definitions for more than one color index in an array of color definition data structures. Use the pixel member in the color definition data structures to specify the color indexes you want definitions for. The red, green, and blue values are returned in the same data structures for the specified entries. The flags members in the data structures are ignored.

If a color definition specified in **screen_defs_return** is unallocated, the red, green, and blue values returned are undefined.

When you want the color definitions for one color index, use the QUERY COLOR routine.

The color map identifier was originally returned by DEFAULT COLORMAP or CREATE COLORMAP.

X ERRORS

VAX	C	Description
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_VALUE	BadValue	The index you specified lies outside the range of the color map.

SET STANDARD COLORMAP

Specifies a standard color map.

VAX FORMAT

X$SET_STANDARD_COLORMAP *(display, window_id, standard_colormap, property_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
standard_ colormap	record	x$standard_ colormap	read	reference
property_id	identifier	uns longword	read	reference

MIT C FORMAT

XSetStandardColormap *(display, window_id, standard_colormap, property_id)*

argument information

```
XSetStandardColormap(display, window_id, standard_colormap,
                    property_id)
      Display *display;
      Window window_id;
      XStandardColormap *standard_colormap;
      Atom property_id; /* RGB_BEST_MAP, etc. */
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to associate with the standard color map, once created. This window is usually the root window.

standard_colormap

The standard color map data structure that defines the standard color map to attach to the window.

For more information about the standard color map data structure, see Section 12.1.

property_id

The identifier of the property type for the standard color map to be associated with the window. A property type is specified by an atom. There are six predefined property types for standard color maps:

- RGB_DEFAULT_MAP
- RGB_BEST_MAP
- RGB_RED_MAP
- RGB_GREEN_MAP
- RGB_BLUE_MAP
- RGB_GRAY_MAP

For more information about properties, see Chapter 8.

DESCRIPTION

SET STANDARD COLORMAP associates a standard color map with the specified window (usually the root window). This routine is usually only used by a window manager program.

First, you must create a standard color map and specify the colors for it. After it has been created and a standard color map data structure has been specified for it, use this routine to associate it as a property with the window.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

SET WINDOW COLORMAP

Specifies the current color map for the specified window.

VAX FORMAT

X$SET_WINDOW_COLORMAP
(display, window_id, colormap_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference

MIT C FORMAT

XSetWindowColormap
(display, window_id, colormap_id)

argument information

```
XSetWindowColormap(display, window_id, colormap_id)
      Display *display;
      Window window_id;
      Colormap colormap_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window to be associated with the color map specified in **colormap_id**.

colormap_id

The identifier of the new color map to be associated with the window specified in **window_id**. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

DESCRIPTION

SET WINDOW COLORMAP associates the specified color map with the window. The identifier for the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

Note that associating a new color map with a window may not immediately change the window colors.

X ERRORS

VAX	C	Description
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_MATCH	BadMatch	The color map depth or visual type does not match the specified window depth or visual type.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

STORE COLOR

Sets a color map entry, previously allocated, to the closest color supported by the hardware.

VAX FORMAT

X$STORE_COLOR
(display, colormap_id, color_def)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
color_def	record	x$color	modify	reference

MIT C FORMAT

XStoreColor
(display, colormap_id, color_def)

argument information

```
XStoreColor(display, colormap_id, color_def)
      Display *display;
      Colormap colormap_id;
      XColor *color_def;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map where the color definition will be stored. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

color_def

The color definition data structure that specifies the desired red, green, and blue color values; the color index where the color definition should be stored; and the flags member that specifies whether to set the red, green, or blue entries in the color map. To set the flags member, do a bitwise OR with these predefined members:

VAX	C	Description
X$M_DO_RED	DoRed	Modify the red definition

VAX	C	Description
X$M_DO_GREEN	DoGreen	Modify the green definition
X$M_DO_BLUE	DoBlue	Modify the blue definition

The color stored is the closest color supported by the hardware. The color definition must be a read/write entry.

For more information about the color definition data structure, see Section 12.2.

DESCRIPTION

STORE COLOR sets one color definition in the specified color map. To change more than one color, use STORE COLORS.

In using this routine:

- You must have already allocated the color index using ALLOC COLOR CELLS or ALLOC COLOR PLANES. These routines allow read and write access to the allocated color definition and return the color index.
- You specify the color index, where you want the color set, in the pixel member of the color definition data structure.
- You specify the color you want by defining the red, green, and blue values in the color definition data structure.
- You specify which of the color definitions will be modified by doing a bitwise OR on the flag member with the predefined values DoRed, DoGreen, and DoBlue.

The color that is set in the color map is the one closest to what you specified that is supported by the hardware. If the color map specified is an installed color map, the new color is visible immediately.

The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

X ERRORS

VAX	C	Description
X$C_BAD_ACCESS	BadAccess	An attempt was made to store into a read-only or unallocated color map entry.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_VALUE	BadValue	The index value you specified is not valid for the color map.

STORE COLORS

Sets more than one color definition, previously allocated, to the closest colors supported by the hardware.

VAX FORMAT

X$STORE_COLORS
(display, colormap_id, color_defs, num_colors)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
color_defs	array	x$color	write	reference
num_colors	longword	longword	read	reference

MIT C FORMAT

XStoreColors
(display, colormap_id, color_defs, num_colors)

argument information

```
XStoreColors(display, colormap_id, color_defs, num_colors)
      Display *display;
      Colormap colormap_id;
      XColor color_defs[];
      int num_colors;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map where the color definitions will be set. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

color_defs

The color definition data structure that specifies the desired red, green, and blue color values; the color index where the color definition should be stored; and the flags member that specifies whether to set the red, green, or blue entries in the color map. To set the flags member, do a bitwise OR with these predefined members:

VAX	C	Description
X$M_DO_RED	DoRed	Modify the red definition
X$M_DO_GREEN	DoGreen	Modify the green definition
X$M_DO_BLUE	DoBlue	Modify the blue definition

The color stored is the closest color supported by the hardware. The color definition must be a read/write entry.

For more information about the color definition data structure, see Section 12.2.

num_colors

The number of color definitions to be set in the color map. This value specifies the number of color definition data structures and thereby specifies the length of the **color_defs** array.

DESCRIPTION

STORE COLORS sets more than one color definition in the specified color map. To change only one color, use STORE COLOR.

To use this routine:

- You must have already allocated the color indexes using ALLOC COLOR CELLS or ALLOC COLOR PLANES. These routines allow read and write access to the allocated color definitions and return the color indexes.
- You specify the color indexes where you want the colors set in the pixel members of the color definition data structures. If the color map is installed, the new colors are visible immediately.
- You specify the colors you want by defining the red, green, and blue values in the color definition data structures.
- You specify which color definition members to set by doing a bitwise OR operation on the flag members with the predefined values do red, do green, and do blue.

The colors that are set in the color definitions are as close to what you specified as the hardware supports.

X ERRORS

VAX	C	Description
X$C_BAD_ACCESS	BadAccess	An attempt was made to store in to a read-only or unallocated color map entry.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_VALUE	BadValue	The index value you specified is not valid for the color map.

STORE NAMED COLOR

Sets the specified color map entry to the named color.

VAX FORMAT

X$STORE_NAMED_COLOR
(display, colormap_id, color_name, pixel, flags)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
colormap_id	identifier	uns longword	read	reference
color_name	char string	char string	read	descriptor
pixel	uns longword	uns longword	read	reference
flags	uns longword	uns longword	read	reference

MIT C FORMAT

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

argument information

```
XStoreNamedColor(display, colormap_id, color_name, pixel, flags)
      Display *display;
      Colormap colormap_id;
      char *color_name;
      unsigned long pixel;
      int flags;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

colormap_id

The identifier of the color map where the color will be set. The identifier of the color map was originally returned by DEFAULT COLORMAP, CREATE COLORMAP, or COPY COLORMAP AND FREE.

color_name

The name of the new color. The color name specified must be supported by the color database. Use the ISO Latin-1 encoding. Case is not significant.

C only

The **color_name** argument is a null-terminated character string.

pixel

The color index (pixel value) of the entry to be set in the color map.

flags

Specifies whether to write the red, green, or blue values. Do a bitwise OR operation with these predefined values:

VAX	C	Description
X$M_DO_RED	DoRed	Use the red member
X$M_DO_GREEN	DoGreen	Use the green member
X$M_DO_BLUE	DoBlue	Use the blue member

DESCRIPTION

STORE NAMED COLOR sets one color definition in the specified color map according to the color you specify by name.

To use this routine:

- You must have already allocated the color definition using ALLOC COLOR CELLS or ALLOC COLOR PLANES. These routines allow read and write access to the allocated color definition and return the color index.
- You specify the color index, where you want the color set, in **pixel**.
- You specify the color you want in **color_name**. The name you specify must be supported in the color database maintained by the server. The color database provides the red, green, and blue values for the named color to the specified color definition.
- You specify which colors to set by doing a bitwise OR operation in the **flags** argument with the predefined values do red, do green, and do blue.

X ERRORS

VAX	C	Description
X$C_BAD_ACCESS	BadAccess	An attempt was made to store in to a read-only or unallocated color map entry.
X$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X$C_BAD_NAME	BadName	The font or color that you specified does not exist.
X$C_BAD_VALUE	BadValue	The index value you specified is not valid for the color map.

13 Font Routines

This chapter describes routines that perform the following functions:

- Loading fonts
- Freeing fonts
- Defining fonts
- Getting information about fonts

For concepts related to font routines and information on how to use font routines, see the *VMS DECwindows Xlib Programming Volume.*

The routines described in this chapter are listed in Table 13–1.

Table 13–1 Window and Session Font Routines

Routine Name	Description
FREE FONT	Frees all storage associated with the specified font and closes the font.
FREE FONT INFO	Frees storage created for font information returned by LIST FONTS WITH INFO. This routine is used only with the MIT C binding.
FREE FONT NAMES	Releases the storage occupied by the specified list of font names. This routine is used only with the MIT C binding.
FREE FONT PATH	Releases the storage occupied by the specified font path. This routine is used only with the MIT C binding.
GET CHAR STRUCT	Fetches character data structure information from a font data structure. This routine is used only with the VAX binding.
GET FONT PATH	Returns the current directory path used by the server to locate fonts.
GET FONT PROPERTY	Returns the value of a specified font property, given the property's associated atom and the font data structure address.
LIST FONT	Returns the font name of a specified font, if the font exists. This routine is used only with the VAX binding.
LIST FONT WITH INFO	Returns the font name of a specified font, if the font exists, and information about that font. This routine is used only with the VAX binding.
LIST FONTS	Returns a list of font names that match the specified naming pattern.

(continued on next page)

Table 13–1 (Cont.) Window and Session Font Routines

Routine Name	Description
LIST FONTS WITH INFO	Returns a list of font names of loaded fonts and information about those fonts.
LOAD FONT	Loads the specified font into server memory.
LOAD QUERY FONT	Loads a specified font and returns information about it in a font data structure.
QUERY FONT	Returns information about a loaded font. How the returned information is accessed differs depending on the binding you use.
SET FONT PATH	Defines the directory path used by the server to locate fonts.
UNLOAD FONT	Closes the specified font and, if no other processes are referencing the font, unloads it from server memory.

The following three structures define a font:

VAX	C
X$FONT_STRUCT	XFontStruct
X$CHAR_STRUCT	XCharStruct
X$FONT_PROP	XFontProp

The sections that follow describe these structures.

13.1 Font Data Structure

The font data structure contains information about a font associated with a display.

The font data structure for the VAX binding is shown in Figure 13–1, and members of the data structure are described in Table 13–2.

Figure 13–1 Font Data Structure (VAX Binding)

x$a_fstr_ext_data	0
x$l_fstr_fid	4
x$l_fstr_direction	8
x$l_fstr_min_char_or_byte2	12
x$l_fstr_max_char_or_byte2	16
x$l_fstr_min_byte1	20

(continued on next page)

Figure 13–1 (Cont.) Font Data Structure (VAX Binding)

x$l_fstr_max_byte1	24
x$l_fstr_all_chars_exist	28
x$l_fstr_default_char	32
x$l_fstr_n_properties	36
x$a_fstr_properties	40
x$a_fstr_min_bounds	44
x$a_fstr_max_bounds	48
x$a_fstr_per_char	52
x$l_fstr_ascent	56
x$l_fstr_descent	60

Table 13–2 Members of the Font Data Structure (VAX Binding)

Member Name	Contents
X$A_FSTR_EXT_DATA	Data used by extensions.
X$L_FSTR_FID	Identifier of the font.
X$L_FSTR_DIRECTION	Hint about the direction in which the font is painted. The direction can be either left to right, specified by the constant x$c_font_left_to_right; or right to left, specified by the constant x$c_font_right_to_left. The core protocol does not support vertical text.
X$L_FSTR_MIN_CHAR_OR_BYTE2	The first character in the font.
X$L_FSTR_MAX_CHAR_OR_BYTE2	The last character in the font.
X$L_FSTR_MIN_BYTE1	First row that exists.
X$L_FSTR_MAX_BYTE1	Last row that exists.
X$L_FSTR_ALL_CHARS_EXIST	If the value of this member is true, all characters in the array pointed to by X$A_FSTR_PER_CHAR have nonzero bounding boxes.
X$L_FSTR_DEFAULT_CHAR	The character used when an undefined or nonexistent character is specified. The default character is a 16-bit, not a 2-byte, character. For a multiple-row font, X$L_FSTR_DEFAULT_CHAR has byte 1 in the most signficant byte and byte 2 in the least significant byte. If X$L_FSTR_DEFAULT_CHAR specifies an undefined or nonexistent character, the server does not print an undefined or nonexistent character.

(continued on next page)

Table 13–2 (Cont.) Members of the Font Data Structure (VAX Binding)

Member Name	Contents
X$L_FSTR_N_PROPERTIES	The number of properties associated with the font.
X$A_FSTR_PROPERTIES	The address of an array of font property structures that define font properties.
X$R_FSTR_MIN_BOUNDS	The minimum metrics values of all the characters in the font. The metrics define the left and right bearings, ascent and descent, and width of characters. For a description of the use of X$R_FSTR_MIN_BOUNDS, see X$R_FSTR_MAX_BOUNDS.

X$R_FSTR_MAX_BOUNDS	The maximum metrics values of all the characters in the font.
	Using the values of X$R_FSTR_MIN_BOUNDS and X$R_FSTR_MAX_BOUNDS, clients can compute the bounding box of a font. The bounding box of the font is determined by first computing the minimum and maximum value of the left bearing, right bearing, ascent, descent, and width of all characters, and then subtracting minimum from maximum values. The upper left coordinate of the font bounding box (x,y) is defined as follows:
	`x + X$R_FSTR_MIN_BOUNDS.X$W_CHAR_LBEARING,`
	`y - X$R_FSTR_MAX_BOUNDS.X$W_CHAR_ASCENT`
	The width of the font bounding box is defined as follows:
	`X$R_FSTR_MAX_BOUNDS.X$W_CHAR_RBEARING -`
	`X$R_FSTR_MIN_BOUNDS.X$W_CHAR_LBEARING`
	Note that this is not the width of a font character.
	The height is defined as follows:
	`X$R_FSTR_MAX_BOUNDS.X$W_CHAR_ASCENT +`
	`X$R_FSTR_MAX_BOUNDS.X$W_CHAR_DESCENT`

X$A_FSTR_PER_CHAR	The address of an array of character structures that define each character in the font. For a fixed font the value of this member is null.
X$L_FSTR_ASCENT	The distance from the baseline to the top of the bounding box. With X$L_FSTR_DESCENT, X$L_FSTR_ASCENT is used to determine line spacing. Specific characters in the font may extend beyond the font ascent.
X$L_FSTR_DESCENT	The distance from the baseline to the bottom of the bounding box. With X$L_FSTR_ASCENT, X$L_FSTR_DESCENT is used to determine line spacing. Specific characters in the font may extend beyond the font descent.

The font data structure for the MIT C binding is shown in Figure 13–2, and members of the data structure are described in Table 13–3.

Figure 13–2 Font Data Structure (MIT C binding)

```
typedef struct {
    XExtData    *ext_data;
    Font        fid;
    unsigned    direction;
    unsigned    min_char_or_byte2;
    unsigned    max_char_or_byte2;
    unsigned    min_byte1;
    unsigned    max_byte1;
    Bool        all_chars_exist;
    unsigned    default_char;
    int         n_properties;
    XFontProp   *properties;
    XCharStruct min_bounds;
    XCharStruct max_bounds;
    XCharStruct *per_char;
    int         ascent;
    int         descent;
} XFontStruct;
```

Table 13–3 Members of the Font Data Structure (MIT C Binding)

Member Name	Contents
ext_data	Data used by extensions.
fid	Identifier of the font.
direction	Hint about the direction the font is painted. The direction can be either left to right, specified by the constant FontLeftToRight; or right to left, specified by the constant FontRightToLeft. The core protocol does not support vertical text.
min_char_or_byte2	The first character in the font.
max_char_or_byte2	The last character in the font.
min_byte1	First row that exists.
max_byte1	Last row that exists.
all_chars_exist	If the value of this member is true, all characters in the array pointed to by per_char have nonzero bounding boxes.
default_char	The character used when an undefined or nonexistent character is printed. The default character is a 16-bit, not a 2-byte, character. For a multiple-row font, default_char has byte 1 in the most signficant byte and byte 2 in the least significant byte. If default_char specifies an undefined or nonexistent character, the server does not print an undefined or nonexistent character.
n_properties	The number of properties associated with the font.
properties	The address of an array of additional font properties.

(continued on next page)

Table 13–3 (Cont.) Members of the Font Data Structure (MIT C Binding)

Member Name	Contents
min_bounds	The minimum bounding box value of all the elements in the array of character structures that define each character in the font. For a description of the use of min_bounds see max_bounds.

max_bounds	The maximum metrics values of all the characters in the font.
	Using the values of min_bounds and max_bounds, clients can compute the bounding box of the font. The bounding box of the font is determined by first computing the minimum and maximum values of the left bearing, right bearing, width, ascent, and descent of all characters, and then subtracting minimum from maximum values. The upper-left coordinate of the bounding box (x,y) is defined as follows:
	`x + min_bounds.lbearing, y - max_bounds.ascent`
	The width of the font bounding box is defined as follows:
	`max_bounds.rbearing - min_bounds.lbearing`
	Note that this is not the width of a character in the font.
	The height is defined as follows:
	`max_bounds.ascent + max_bounds.descent`

per_char	The address of an array of character structures that define each character in the font.
ascent	The distance from the baseline to the top of the bounding box. With descent, ascent is used to determine line spacing. Specific characters in the font may extend beyond the font ascent.
descent	The distance from the baseline to the bottom of the bounding box. With ascent, descent is used to determine line spacing. Specific characters in the font may extend beyond the font descent.

13.2 Character Data Structure

The character data structure for the VAX binding is shown in Figure 13–3, and members of the data structure are described in Table 13–4.

Figure 13–3 Character Data Structure (VAX Binding)

x$w_char_rbearing	x$w_char_lbearing	0
x$w_char_ascent	x$w_char_width	4
x$w_char_attributes	x$w_char_descent	8

Table 13–4 Members of the Character Data Structure (VAX Binding)

Member Name	Contents
X$W_CHAR_LBEARING	Distance from the origin to the left edge of the bounding box. When the value of this member is zero, the server draws only pixels whose x-coordinates are less than the value of the origin x-coordinate.
X$W_CHAR_RBEARING	Distance from the origin to the right edge of the bounding box.
X$W_CHAR_WIDTH	Distance from the current origin to the origin of the next character. Text written right to left, such as Arabic, uses a negative width to place the next character to the left of the current origin.
X$W_CHAR_ASCENT	Distance from the baseline to the top of the bounding box.
X$W_CHAR_DESCENT	Distance from the baseline to the bottom of the bounding box.
X$W_CHAR_ATTRIBUTES	Attributes defined in the bitmap distribution format file. A character is not guaranteed to have any attributes.

The character data structure for the MIT C binding is shown in Figure 13–4, and members of the data structure are described in Table 13–5.

Figure 13–4 Character Data Structure (MIT C Binding)

```
typedef struct {
    short       lbearing;
    short       rbearing;
    short       width;
    short       ascent;
    short       descent;
    unsigned short attributes;
} XCharStruct;
```

Table 13–5 Members of the Character Data Structure (MIT C Binding)

Member Name	Contents
lbearing	Distance from the origin to the left edge of the bounding box. When the value of this member is zero, the server draws only pixels whose x-coordinates are less than the value of the origin x-coordinate.
rbearing	Distance from the origin to the right edge of the bounding box.
width	Distance from the current origin to the origin of the next character. Text written right to left, such as Arabic, uses a negative width to place the next character to the left of the current origin.
ascent	Distance from the baseline to the top of the bounding box.
descent	Distance from the baseline to the bottom of the bounding box.
attributes	Attributes of the character defined in the bitmap distribution format (BDF) file. A character is not guaranteed to have any attributes.

13.3 Font Property Data Structure

Any number of properties or no properties at all may be associated with a font. If properties are associated with a font, they are defined in a font property data structure.

Property values can be obtained with the QUERY FONT, LOAD QUERY FONT, and GET FONT PROPERTY routines. Whether a property is signed or unsigned must be determined from prior knowledge of the property.

The font property data structure for the VAX binding is shown in Figure 13–5, and members of the data structure are described in Table 13–6.

Figure 13–5 Font Property Data Structure (VAX Binding)

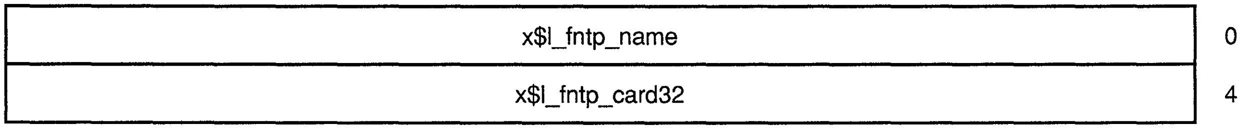


Table 13–6 Members of the Font Property Data Structure (VAX Binding)

Member Name	Contents
X$L_FNTP_NAME	The string of characters that names the property
X$L_FNTP_CARD32	A 32-bit value that defines the font property

The font property data structure for the MIT C binding is shown in Figure 13–6, and members of the data structure are described in Table 13–7.

Figure 13–6 Font Property Data Structure (MIT C Binding)

```
typedef struct {
        Atom    name;
        unsigned long card32;
} XFontProp;
```

Table 13–7 Members of the Font Property Data Structure (MIT C Binding)

Member Name	Contents
name	The string of characters that names the property
card32	A 32-bit value that defines the font property

13.4 Font Routines

The following pages describe the Xlib font routines.

FREE FONT

Frees all storage associated with the specified font and closes the font.

VAX FORMAT **X$FREE_FONT** *(display, font_struct)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
font_struct	record	x$font_struct	read	reference

MIT C FORMAT **XFreeFont** *(display, font_struct)*

argument information

```
XFreeFont(display, font_struct)
      Display *display;
      XFontStruct *font_struct;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

font_struct

The address of a structure that holds font information. For information about the font data structure, see Section 13.3.

DESCRIPTION

FREE FONT frees all storage used by the font data structure, effectively closing the font. When a font is no longer needed by an application, the application should call FREE FONT. The associated font identifier is invalid after the storage is freed and should no longer be referenced.

X ERRORS

VAX	C	Description
X$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, a graphics context).

FREE FONT INFO

Frees storage created for font information returned by LIST FONTS WITH INFO. This routine is used only with the MIT C binding.

MIT C FORMAT

XFreeFontInfo
(font_names_ptr, info_addr, count)

argument information

```
XFreeFontInfo(font_names_ptr, info_addr, count)
      char **font_names_ptr;
      XFontStruct *info_addr;
      int count;
```

ARGUMENTS

font_names_ptr

A pointer to the list of font names whose storage is to be freed. The pointer is returned by LIST FONTS WITH INFO.

info_addr

The address of an array of font data structures containing information related to the fonts listed by **font_names_ptr**. The address is returned by LIST FONTS WITH INFO.

count

The actual number of font names. This value is returned by LIST FONTS WITH INFO.

DESCRIPTION

FREE FONT INFO frees memory used to store font information returned by LIST FONTS WITH INFO.

FREE FONT NAMES

Releases the storage occupied by the specified list of font names. This routine is used only with the MIT C binding.

MIT C FORMAT

XFreeFontNames *(list_addr)*

argument information

```
XFreeFontNames(list_addr)
      char *list_addr[];
```

ARGUMENTS

list_addr

A pointer to an array of null-terminated string pointers. The pointer is returned by LIST FONTS.

DESCRIPTION

FREE FONT NAMES releases the storage occupied by the specified list of font names.

FREE FONT PATH

Releases the storage occupied by the specified font path. This routine is used only with the MIT C binding.

MIT C FORMAT **XFreeFontPath** *(list_addr)*

argument information

```
XFreeFontPath(list_addr)
      char **list_addr;
```

ARGUMENTS

list_addr

A pointer to an array of null-terminated string pointers. The pointer is returned by GET FONT PATH.

DESCRIPTION

FREE FONT PATH releases the storage occupied by the specified font path.

GET CHAR STRUCT

Fetches character structure information from a font data structure. This routine is used only with the VAX binding.

VAX FORMAT

X$GET_CHAR_STRUCT
(font_info, char_code, char_struc)

argument information

Argument	Usage	Data Type	Access	Mechanism
font_info	record	x$font_struct	read	reference
char_code	longword	uns longword	read	reference
char_struc	record	x$char_struct	read	reference

ARGUMENTS

font_info

The address of the font data structure that holds the character information to be accessed. See Section 13.3 for a description of the font data structure.

char_code

The ASCII value of the character for which information is returned.

char_struc

The address of a structure that holds information about the specified character. The character data structure is described in Section 13.2.

DESCRIPTION

GET CHAR STRUCT returns information about a particular character in a font, given the address of the font data structure that contains the information and the ASCII value of the character.

GET FONT PATH

Returns the current directory path used by the server to locate fonts.

VAX FORMAT

status_return = **X$GET_FONT_PATH** *(display, num_paths_return, directories_return [,len_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	uns longword	write	value
display	identifier	uns longword	read	reference
num_paths_return	longword	uns longword	read	reference
directories_return	char string	char string	write	descriptor
len_return	word	uns word	write	reference

MIT C FORMAT

directories_return = **XGetFontPath** *(display, num_paths_return)*

argument information

```
char **XGetFontPath(display, num_paths_return)
      Display *display;
      int *num_paths_return;
```

RETURNS

status_return (VAX only)

Return value that specifies whether the routine completed successfully.

directories_return (MIT C only)

A pointer to a string array of null-terminated directory names that make up the current font directory path.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

num_paths_return

Number of strings that make up the directory path.

directories_return (VAX only)

Comma-separated list of directories that make up the current font directory path.

len_return (VAX only)

The length of the returned string. This argument is optional.

DESCRIPTION

GET FONT PATH returns the current directory path used by the server when it is locating a font. A directory path may be made up of one or more directory names.

You can change the font directory path using the SET FONT PATH routine.

Use the FREE FONT PATH routine to free the data in the font path when the data is no longer needed.

GET FONT PROPERTY

Returns the value of a specified font property, given the property's associated atom and the font data structure address.

VAX FORMAT

status_return = **X$GET_FONT_PROPERTY** *(font_struct, atom_id, value_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
font_struct	record	x$font_struct	read	reference
atom_id	identifier	uns longword	read	reference
value_return	longword	longword	write	value

MIT C FORMAT

status_return = **XGetFontProperty** *(font_struct, atom_id, value_return)*

argument information

```
Bool XGetFontProperty(font_struct, atom_id, value_return)
      XFontStruct *font_struct;
      Atom atom_id;
      unsigned long *value_return;
```

RETURNS

status_return
Return value that specifies whether the routine completed successfully.

ARGUMENTS

font_struct
Address of the font data structure, returned by the LOAD FONT and FONT routines. The font data structure contains a pointer to font property information.

atom_id
Identifier of the atom associated with the property you want returned.

value_return
The returned property value.

DESCRIPTION

GET FONT PROPERTY returns the value of a font property, given the associated atom and the address of the font data structure.

A set of predefined atoms exists for font properties in the <X11/Xatom.h> library. For a complete description of predefined atoms, see Chapter 4.

LIST FONT

Returns the name of a specified font, if the font exists. This routine is used only with the VAX binding.

VAX FORMAT

status_return = **X$LIST_FONT**
(display, pattern_name, context, name [,len_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
pattern_name	char string	char string	read	descriptor
context	context	uns longword	modify	reference
name	char string	char string	write	descriptor
len_return	word	uns word	write	reference

RETURNS

status_return

Return value that states whether or not the routine completed successfully. This argument returns one of the following values:

Value	Description
1	A font name matching the pattern has been returned.
X$_NOTFOUND	No fonts match the specified pattern.
X$_NOMORE	No more fonts match the pattern.
LIB$STRTRU	A matching font name was returned but the fixed length destination string could not contain all of the characters copied from the font name.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

pattern_name

The address of a descriptor that points to a string. The string specifies a pattern that the returned font name must match. Both wildcard characters are acceptable—use an asterisk (*) for any number of characters and use a question mark (?) for a single character.

context

The address of a longword that stores the state of the search. The argument should not be modified if repetitive searches are desired.

You must initialize **context** to 0 before starting a search.

name

The address of a descriptor that points to a character string. The character string contains the returned font name.

len_return

The length of the returned string.

DESCRIPTION

LIST FONT returns a single font name that matches the string specified by the **pattern_name** argument.

LIST FONT WITH INFO

Returns the name of a specified font, if the font exists, and information about that font. This routine is used only with the VAX binding.

VAX FORMAT

status_return = **X$LIST_FONT_WITH_INFO** *(display, pattern_name, context, font_name_return [,len_return], font_struct_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
pattern_name	char_string	character string	read	descriptor
context	context	uns longword	modify	reference
font_name_return	char string	char string	write	descriptor
len_return	word	uns word	write	reference
font_struct_return	record	x$font_struct	write	reference

RETURNS

status_return

Return value that states whether or not the routine completed successfully. This argument returns one of the following values:

Value	Description
1	A font name matching the pattern has been returned.
X$_NOTFOUND	No fonts match the specified pattern.
X$_NOMORE	No more fonts match the pattern.
LIB$STRTRU	A matching font name was returned but the fixed length destination string could not contain all of the characters copied from the font name.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

pattern_name

The address of a descriptor that points to a string. The string specifies a pattern that the returned font name must match. Both wildcard characters are acceptable—use an asterisk (*) for any number of characters and use a question mark (?) for a single character.

context

The address of a longword that stores the state of the search. The argument should not be modified if repetitive searches are desired.

You must initialize **context** to 0 before starting a search.

font_name_return

The address of a descriptor that points to a character string. The character string contains the returned font name.

len_return

The length of the returned string.

font_struct_return

The address of the font data structure associated with the font.

DESCRIPTION

LIST FONT WITH INFO returns a single font name that matches the string specified by the **pattern_name** argument, as well as information associated with that font.

LIST FONTS

Returns a list of font names that match the specified naming pattern.

VAX FORMAT

status_return = **X$LIST_FONTS**
(display, pattern_name, match_limit, actual_count_return, names_return [,len_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
pattern_name	char string	char string	read	descriptor
match_limit	longword	longword	read	reference
actual_count_return	longword	longword	write	value
names_return	char string	char string	write	descriptor
len_return	word	uns word	write	reference

MIT C FORMAT

font_ptr = **XListFonts**
(display, pattern_name, match_limit, actual_count_return)

argument information

```
char **XListFonts(display, pattern_name, match_limit,
                  actual_count_return)
      Display *display;
      char *pattern_name;
      int match_limit;
      int *actual_count_return;
```

RETURNS

status_return (VAX only)
Return value that specifies whether the routine completed successfully.

font_ptr (MIT C only)
A pointer to an array of available font names.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

pattern_name

A character string specifying a pattern that the returned font names must match. Use ISO Latin-1 encoding to specify the string. Both wildcard characters are acceptable—use an asterisk (*) for any number of characters and use a question mark (?) for a single character.

match_limit

The number of font names in the requested list.

actual_count_return

The actual number of font names returned.

names_return (VAX only)

A character string containing all the returned font names separated by commas.

len_return (VAX only)

The length of the returned string of font names. This argument is optional.

DESCRIPTION

LIST FONTS returns a list of font names that match the string pattern defined by **pattern_name**. The number of font names returned is limited to the value specified by **actual_count_return**.

When finished with the font name list, a client should free server memory with FREE FONT NAMES.

LIST FONTS WITH INFO

Returns a list of names of loaded fonts and information about those fonts.

FORMAT

status_return = **X$LIST_FONTS_WITH_INFO** *(display, pattern_name, maxnames, count_return, font_names_return [,len_return] [,info_return] [,info_size] [,info_buff_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond value	uns longword	write	value
display	identifier	uns longword	read	reference
pattern_name	char string	char string	read	descriptor
maxnames	longword	longword	read	reference
count_return	longword	longword	write	reference
font_names_return	char string	char string	write	descriptor
len_return	word	uns word	write	reference
info_return	address	uns longword	read	reference
info_size	longword	longword	read	reference
info_buff_return	v uns longword	uns longword	write	reference

MIT C FORMAT

font_names_ptr = **XListFontsWithInfo** *(display, pattern_name, maxnames, count_return, font_names_return)*

argument information

```
char **XListFontsWithInfo(display, pattern_name, maxnames,
                          count_return, font_names_return)
      Display *display;
      char *pattern_name;
      int maxnames;
      int *count_return;
      XFontStruct **font_names_return;
```

RETURNS

status_return (VAX only)

Return value that specifies whether the routine completed successfully.

font_names_ptr (MIT C only)

A pointer to the address of a list of font names.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

pattern_name

A null-terminated character string specifying a pattern that the returned font names must match. Both wildcard characters are acceptable—use an asterisk (*) for any number of characters and use a question mark (?) for a single character.

maxnames

The maximum number of font names to be returned.

count_return

The actual number of matched font names.

font_names_return

The virtual address of a pointer to an array of font data, returned by the routine and residing in space reserved by Xlib.

len_return (VAX only)

The length of the string of font names returned in **font_names_return**.

info_return (VAX only)

The virtual address of a pointer to an array of font information data, returned by the routine and residing in space reserved by Xlib.

info_size (VAX only)

The size of the buffer specified in **info_buff_return**.

info_buff_return (VAX only)

A pointer to a data buffer, residing in space you have reserved, where each entry is one font information element. The length of the buffer is specified by **info_size**. The property data is returned by the routine.

DESCRIPTION

LIST FONTS WITH INFO returns a list of font names that match the pattern given in the **pattern_name** argument. The routine also returns information associated with each font that matches the pattern.

The list of names returned is limited to the number defined with **maxnames**.

To specify arguments that describe the font information data returned by the routine, use **info_return** to access data owned by Xlib, or **info_size** and **info_buff_return** to obtain a private copy of the data.

The information returned for each font is identical to what LOAD QUERY FONT returns, except that the character metrics are not returned.

LOAD FONT

Loads the specified font into server memory.

VAX FORMAT

font_id = **X$LOAD_FONT** *(display, font_name)*

argument information

Argument	Usage	Data Type	Access	Mechanism
font_id	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
font_name	char string	char string	read	descriptor

MIT C FORMAT

font_id = **XLoadFont** *(display, font_name)*

argument information

```
font_id XLoadFont(display, font_name)
     Display *display;
     char *name;
```

RETURNS

font_id

The font identifier returned by the server after the specified font is loaded. This identifier is used in subsequent routines that manipulate the font.

If the specified font cannot be loaded, the server returns a zero in place of the font identifier. Before attempting to use an identifier, an application should test the validity of the identifier.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

font_name

The name of the font to be loaded into server memory.

DESCRIPTION

LOAD FONT loads the specified font into server memory and returns an identifier for the font. A font must be loaded in server memory before it can be used by any subsequent routines.

When a font is no longer needed, remove it from server memory using the CLOSE FONT routine.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_NAME	BadName	The font or color that you specified does not exist.

LOAD QUERY FONT

Loads a specified font and returns information about it in a font data structure.

VAX FORMAT

status_return = **X$LOAD_QUERY_FONT** *(display, font_name, font_struct_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
font_name	char string	char string	read	descriptor
font_struct_return	record	x$font_struct	write	reference

MIT C FORMAT

font_struct_return = **XLoadQueryFont** *(display, font_name)*

argument information

```
XFontStruct *XLoadQueryFont(display, font_name)
      Display *display;
      char *font_name;
```

RETURNS

status_return (VAX only)

Return value that specifies whether the routine completed successfully.

font_struct_return (MIT C only)

A pointer to the font data structure associated with the specified font. This pointer can be used to access any of the information in the font data structure, as shown in Section 13.3.

If the information cannot be returned, the server returns a null value.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

font_name

The name of the font to be accessed for information.

font_struct_return (VAX only)

The address of the font data structure associated with the font.

DESCRIPTION

LOAD QUERY FONT loads a specified font into server memory and returns information about the font in the font data structure shown in Section 13.3. If the font does not exist, LOAD QUERY FONT returns a null value.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.

QUERY FONT

Returns information about an available font.

VAX FORMAT

status_return = **X$QUERY_FONT**
(display, font_id, font_struct_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
font_id	identifier	uns longword	read	reference
font_struct_return	record	x$font_struct	write	reference

MIT C FORMAT

font_struct_return = **XQueryFont**
(display, font_id)

argument information

```
XFontStruct *XQueryFont(display, font_id)
      Display *display;
      XID font_id;
```

RETURNS

status_return (VAX only)

Return value that specifies whether the routine completed successfully.

font_struct_return (MIT C only)

A pointer to the font data structure associated with the specified font. This pointer can be used to access any of the information in the font data structure, as shown in Section 13.3.

If the information cannot be returned, the server returns a null value.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

font_id

The identifier that specifies the font being queried. A font identifier is returned by LOAD FONT or LOAD QUERY FONT. Additionally, graphics contexts identify fonts. If you want to use the font specified in the graphics context, use the relevant identifier returned by CREATE GC.

font_struct_return (VAX only)

The address of the font data structure associated with the font. To obtain character structure information from the font data structure, use the GET CHAR STRUCT routine after calling LOAD QUERY FONT.

DESCRIPTION

QUERY FONT returns information associated with a specified font that has been loaded into server memory with LOAD FONT. Access to font information varies according to the type of binding used.

SET FONT PATH

Defines the directory path used by the server to locate fonts.

VAX FORMAT

X$SET_FONT_PATH
(display, directory_names)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
directory_names	char string	char string	read	descriptor

MIT C FORMAT

XSetFontPath
(display, directory_names, num_dirs)

argument information

```
XSetFontPath(display, directory_names, num_dirs)
      Display *display;
      char **directory_names;
      int num_dirs;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

directory_names

A pointer to a character string that specifies the directory path used to look for the font.

Specifying no directory path restores the server's default path.

num_dirs (MIT C only)

Number of directories that make up the directory path.

DESCRIPTION

SET FONT PATH defines the directory path when it is locating a font, for example, during a LOAD FONT routine.

Call SET FONT PATH before loading a font into server memory. Note that SET FONT PATH defines the directory font path for all clients. A directory path consists of one or more directory names. If you place fonts in a directory other than the default, specify the directory path using SET FONT PATH before you try to access the font.

When executed, SET FONT PATH flushes all cached information about fonts for which no explicit resource identifiers are allocated.

X ERRORS

VAX	C	Description
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

UNLOAD FONT

Closes the specified font and, if no other processes are referencing the font, unloads it from server memory.

FORMAT

X$UNLOAD_FONT *(display, font_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
font_id	identifier	uns longword	read	reference

MIT C FORMAT

XUnloadFont *(display, font_id)*

argument information

```
XUnloadFont(display, font_id)
      Display *display;
      Font font_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

font_id

The identifier of the font to be closed. The font identifier is returned by the GET FONT or LOAD QUERY FONT routines when the font is loaded.

DESCRIPTION

UNLOAD FONT closes the specified font and, if no other processes are referencing the font, unloads it from server memory.

Use UNLOAD FONT when your application no longer needs the font. Once unloaded, the font should not be referenced.

X ERRORS

VAX	C	Description
X$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, a graphics context).

14 Cursor Routines

This chapter describes routines that perform the following functions:

- Creating a cursor
- Changing a cursor
- Destroying a cursor
- Associating a cursor with a window

For concepts related to cursor routines and information on how to use cursor routines, see the *VMS DECwindows Xlib Programming Volume.*

The routines described in this chapter are listed in Table 14–1.

Table 14–1 Window and Session Cursor Routines

Routine Name	Description
CREATE FONT CURSOR	Creates a cursor from a library of standard fonts.
CREATE GLYPH CURSOR	Creates a cursor using font glyphs.
CREATE PIXMAP CURSOR	Creates a cursor from two pixmaps. One pixmap defines the shape of the cursor. Another pixmap specifies the mask that determines how the cursor is displayed on the screen.
DEFINE CURSOR	Defines the cursor to be displayed when the mouse is mapped to a window.
FREE CURSOR	Deletes the cursor identifier specified by the user and releases storage allocated for the cursor.
QUERY BEST CURSOR	Determines the largest size cursor, supported by display hardware, that most closely matches the cursor specified by a user.
RECOLOR CURSOR	Changes the color of a cursor specified by the user. If the cursor is currently displayed, the change is immediately visible.
UNDEFINE CURSOR	Removes the association of a cursor with a window.

14.1 Cursor Routines

The following pages describe the Xlib cursor routines.

CREATE FONT CURSOR

Creates a cursor from a library of standard fonts.

VAX FORMAT

cursor_id_return = **X$CREATE_FONT_CURSOR** *(display, cursor_shape)*

argument information

Argument	Usage	Data Type	Access	Mechanism
cursor_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
cursor_shape	longword	uns longword	read	reference

MIT C FORMAT

cursor_id_return = **XCreateFontCursor** *(display, cursor_shape)*

argument information

```
Cursor XCreateFontCursor(display, cursor_shape)
      Display *display;
      unsigned int cursor_shape;
```

RETURNS

cursor_id_return

The cursor identifier returned by the server after the cursor is created. This identifier is used in subsequent routines that manipulate the cursor.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

cursor_shape

The glyph used to create the cursor.

DESCRIPTION

CREATE FONT CURSOR creates a cursor from a library of standard fonts, X11\cursorfont.h, and returns a cursor identifier. For a description of the library of standard fonts, see the *VMS DECwindows Xlib Programming Volume*.

Specify the cursor shape with the **cursor_shape** argument, which uniquely identifies a glyph within the standard font library.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

CREATE GLYPH CURSOR

Creates a cursor using font glyphs.

VAX FORMAT

cursor_id_return = **X$CREATE_GLYPH_CURSOR** *(display, src_font_id, mask_font_id, src_char, mask_char, foreground_color, background_color)*

argument information

Argument	Usage	Data Type	Access	Mechanism
cursor_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
src_font_id	identifier	uns longword	read	reference
mask_font_id	identifier	uns longword	read	reference
src_char	longword	uns longword	read	reference
mask_char	longword	uns longword	read	reference
foreground_color	record	x$color	read	reference
background_color	record	x$color	read	reference

MIT C FORMAT

cursor_id_return = **XCreateGlyphCursor** *(display, src_font_id, mask_font_id, src_char, mask_char, foreground_color, background_color)*

argument information

```
Cursor XCreateGlyphCursor(display, src_font_id, mask_font_id,
                          src_char, mask_char, foreground_color,
                          background_color)
      Display *display;
      Font src_font_id, mask_font_id;
      unsigned int src_char, mask_char;
      XColor *foreground_color;
      XColor *background_color;
```

RETURNS

cursor_id_return
The cursor identifier returned by the server after the cursor is created. This identifier is used in subsequent routines that manipulate the cursor.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

src_font_id

Identifier of the source font that includes the glyph used to create the cursor. The identifier is returned by LOAD FONT.

mask_font_id

Identifier of the font containing masks that control how the cursor is displayed on the screen. The identifier is returned by LOAD FONT, or None.

src_char

The glyph that defines the cursor.

mask_char

The mask used to control how the glyph is displayed on the screen. The set bits of the mask determine which pixels of the glyph are displayed.

foreground_color

Red, green, and blue values of the cursor foreground.

See Chapter 12 for an illustration of the color definition data structure.

background_color

Red, green, and blue values of the cursor background.

See Chapter 12 for an illustration of the color definition data structure.

DESCRIPTION

CREATE GLYPH CURSOR creates a cursor from a font glyph and returns a unique cursor identifier.

Specify the character to be used to create the cursor with the **src_font_id** and **src_char** arguments. The **src_font_id** argument identifies a font data structure of which the character is a member; the **src_char** argument identifies the character within the font data structure that you want to use as a cursor. The **src_char** must be a defined glyph in **src_font_id**.

To control how the glyph is displayed on the screen, specify a mask using the **mask_font_id** and **mask_char** arguments. The **mask_font_id** argument identifies a font data structure that contains masks. The **mask_char** argument identifies the mask you want to use to modify the character. The **mask_char** must be a defined glyph in the **mask_font_id**. The **mask_font_id** could be None, and all pixels of the source would be displayed.

Red, green, and blue values must be specified for the foreground and the background, even if the server has only a monochrome screen. The set bits of the source define the foreground; the zero bits define the background.

The hotspot of the displayed cursor is predefined. The x-coordinate is the left bearing of the displayed character; the y-coordinate is the ascent of the displayed character.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, a graphics context).
X$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

CREATE PIXMAP CURSOR

Creates a cursor from two pixmaps. One pixmap defines the source cursor. Another pixmap specifies the mask that determines how the cursor is displayed on the screen.

VAX FORMAT

cursor_id_return = **X$CREATE_PIXMAP_CURSOR** *(display, src, mask, foreground_color, background_color, x_hot_coord, y_hot_coord)*

argument information

Argument	Usage	Data Type	Access	Mechanism
cursor_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
src	identifier	uns longword	read	reference
mask	identifier	uns longword	read	reference
foreground_color	record	x$color	read	reference
background_color	record	x$color	read	reference
x_hot_coord	longword	longword	read	reference
y_hot_coord	longword	longword	read	reference

MIT C FORMAT

cursor_id_return = **XCreatePixmapCursor** *(display, src, mask, foreground_color, background_color, x_hot_coord, y_hot_coord)*

argument information

```
Cursor XCreatePixmapCursor(display, src, mask, foreground_color,
background_color, x_hot_coord, y_hot_coord)
      Display *display;
      Pixmap src;
      Pixmap mask;
      XColor *foreground_color;
      XColor *background_color;
      unsigned int x_hot_coord, y_hot_coord;
```

RETURNS

cursor_id_return

The cursor identifier returned by the server after the cursor is created. This identifier is used in subsequent routines that manipulate the cursor.

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

src

Identifier of the source pixmap used to create the cursor. The pixmap identifier is returned by CREATE PIXMAP.

mask

Identifier of the mask that controls how the cursor is displayed on the screen. The mask identifier is returned by CREATE PIXMAP.

foreground_color

The red, green, and blue values of the cursor foreground.

background_color

The red, green, and blue values of the cursor background.

x_hot_coord

Cursor hotspot x-coordinate. The **x_hot_coord** and **y_hot_coord** argument match the displayed cursor's position with the movements of the mouse when the mouse is mapped to a window.

y_hot_coord

Cursor hotspot y-coordinate. The **x_hot_coord** and **y_hot_coord** argument match the displayed cursor's position with the movements of the mouse when the mouse is mapped to a window.

DESCRIPTION

CREATE PIXMAP CURSOR creates a cursor from two pixmaps and returns a unique cursor identifier.

Specify the pixmap to be used to create a new cursor with the **src** argument.

The source cursor and mask can originate from any drawable pixmaps.

Both the source and mask must have a depth of one.

To control how the cursor is displayed on the screen, specify a mask with the **mask** argument. The **mask** argument specifies the shape of the cursor. The server performs a logical AND operation on the source and the mask. The resulting set bits determine which pixels are displayed when the cursor is visible. The mask pixmap must be the same size as the source pixmap.

Red, green, and blue values must be specified for the foreground and the background, even if the server has only a monochrome screen. The set bits of the source define the foreground; the zero bits define the background.

The **x_hot_coord** and **y_hot_coord** arguments define the cursor's hotspot, coordinates that reflect the location of a mouse when it is mapped to a window. The hotspot must be a point within the source cursor pixmap.

X ERRORS

VAX	C	Description
X$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.

DEFINE CURSOR

Defines the cursor to be displayed when the mouse is mapped to a window.

VAX FORMAT

X$DEFINE_CURSOR
(display, window_id, cursor_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
cursor_id	identifier	uns longword	read	reference

MIT C FORMAT

XDefineCursor
(display, window_id, cursor_id)

argument information

```
XDefineCursor(display, window_id, cursor_id)
      Display *display;
      Window window_id;
      Cursor cursor_id;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

window_id
Identifier of the window with which the cursor is associated. The window identifier is returned by CREATE WINDOW or WINDOW.

cursor_id
Identifier of the cursor associated with a window. The cursor identifier is returned by CREATE PIXMAP CURSOR, CREATE FONT CURSOR, or CREATE GLYPH CURSOR.

DESCRIPTION

DEFINE CURSOR associates a cursor with a window. After it is defined, the cursor is displayed whenever the mouse is associated with the specified window and the window is visible.

X ERRORS

VAX	C	Description
X$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

FREE CURSOR

Deletes the cursor specified by the user and releases storage allocated for the cursor.

VAX FORMAT

X$FREE_CURSOR *(display, cursor_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
cursor_id	identifier	uns longword	read	reference

MIT C FORMAT

XFreeCursor *(display, cursor_id)*

argument information

```
XFreeCursor(display, cursor_id)
      Display *display;
      Cursor cursor_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

cursor_id

Identifier of the cursor to be deleted. The cursor identifier is returned by CREATE FONT CURSOR, CREATE GLYPH CURSOR, or CREATE PIXMAP CURSOR.

DESCRIPTION

FREE CURSOR deletes a cursor specified by the **cursor_id** argument and releases server memory that has been allocated for the cursor.

You cannot refer to the cursor after it is deleted.

X ERRORS

VAX	C	Description
X$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.

QUERY BEST CURSOR

Determines the largest cursor, supported by display hardware, that most closely matches the cursor specified by a user.

VAX FORMAT

X$QUERY_BEST_CURSOR
(display, drawable_id, width, height, width_return, height_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
width_return	longword	uns longword	write	reference
height_return	longword	uns longword	write	reference

MIT C FORMAT

XQueryBestCursor
(display, drawable_id, width, height, width_return, height_return)

argument information

```
Status XQueryBestCursor(display, drawable_id, width, height,
width_return, height_return)
      Display *display;
      Drawable drawable_id;
      unsigned int width, height;
      unsigned int *width_return, *height_return;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

drawable_id
The identifier of the window or pixmap with which the cursor is associated. The drawable identifier can be either a window identifier or a pixmap identifier. If the drawable is a window, the identifier is returned by CREATE WINDOW or WINDOW. If the drawable is a pixmap, the identifier is returned by CREATE PIXMAP.

width
The width of a cursor specified by the user.

height

The height of a cursor specified by the user.

width_return

The width of an actual cursor supported by display hardware that most closely matches the cursor specified by the user.

height_return

The height of the actual cursor, supported by display hardware that most closely matches the cursor specified by the user.

DESCRIPTION

QUERY BEST CURSOR determines the size of the cursor, supported by hardware, that most closely matches a cursor specified by a user.

Specify the size of a cursor using the **width** and **height** arguments. QUERY BEST CURSOR returns the size in the **width_return** and **height_return** arguments of the largest cursor supported by display hardware.

X ERRORS

VAX	C	Description
X$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.

RECOLOR CURSOR

Changes the color of a cursor specified by the user. If the cursor is currently displayed, the change is immediately visible.

VAX FORMAT

X$RECOLOR_CURSOR
(display, cursor_id, foreground_color, background_color)

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
cursor_id	identifier	uns longword	read	reference
foreground_color	record	x$color	read	reference
background_color	record	x$color	read	reference

MIT C FORMAT

XRecolorCursor
(display, cursor_id, foreground_color, background_color)

argument information

```
XRecolorCursor(display, cursor_id, foreground_color,
              background_color)
      Display *display;
      Cursor cursor_id;
      XColor *foreground_color, *background_color;
```

ARGUMENTS

display
The display information originally returned by OPEN DISPLAY.

cursor_id
The identifier of the cursor whose color is to be changed. The cursor identifier is returned by CREATE FONT CURSOR, CREATE GLYPH CURSOR, or CREATE PIXMAP CURSOR.

foreground_color
Red, green, and blue values of the cursor foreground.

See Chapter 12 for an illustration of the color definition data structure.

background_color
Red, green, and blue values of the cursor background.

See Chapter 12 for an illustration of the color definition data structure.

DESCRIPTION

RECOLOR CURSOR changes the color of a cursor identified by the **cursor_id** argument.

Specify the new foreground color with the **foreground_color** argument and the new background color with the **background_color** argument.

Red, green, and blue values must be specified for the foreground and the background, even if the server has only a monochrome screen. The set bits of the source define the foreground; the zero bits define the background.

X ERRORS

VAX	C	Description
X$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.

UNDEFINE CURSOR

Removes the association of a cursor with a window.

VAX FORMAT X$UNDEFINE_CURSOR *(display, window_id)*

argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

MIT C FORMAT XUndefineCursor *(display, window_id)*

argument information

```
XUndefineCursor(display, window_id)
      Display *display;
      Window window_id;
```

ARGUMENTS

display

The display information originally returned by OPEN DISPLAY.

window_id

The identifier of the window with which the cursor is associated. The identifier is returned by CREATE WINDOW or WINDOW.

DESCRIPTION

UNDEFINE CURSOR removes the association of a cursor with a window. After the user defined cursor has been dissociated from the window, the cursor of the window's parent is displayed whenever the mouse is associated with the specified window and the window is visible.

When no cursor is specified in a root window after completion of the routine, the default cursor is restored.

X ERRORS

VAX	C	Description
X$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

15 Resource Manager Routines

The resource manager is essentially a database manager. The resources managed by the resource manager include various attributes of the DECwindows environment. For example, an application button might require resources such as a title string, font, foreground color, and background color. Each of these resources is an entry in a resource database.

DECwindows provides a set of routines that allow users to manipulate the resource manager. These functions provide for

- Storing and retrieving resources
- Retrieving database levels
- Converting resource values
- Merging two resource databases
- Retrieving and storing databases

The resource manager routines described in this chapter are listed in Table 15–1.

Table 15–1 Resource Manager Routines

Routine Name	Description
PERMALLOC	Allocates memory for permanently allocated storage.
RM GET FILE DATABASE	Retrieves a database in text file format.
RM GET RESOURCE	Retrieves a resource from the database.
RM GET STRING DATABASE	Creates a database from a string.
RM INITIALIZE	Initializes the resource manager.
RM MERGE DATA BASES	Merges the contents of one database into another.
RM PUT FILE DATABASE	Stores a copy of the application's current database in nonvolatile storage.
RM PUT LINE RESOURCE	Adds a single resource entry to a specified database.
RM PUT RESOURCE	Stores a resource in the database.
RM PUT STRING RESOURCE	Adds a resource that is specified as a string.
RM Q GET RESOURCE	Retrieves a resource from the database.
RM Q GET SEARCH LIST	Returns a list of database levels.

(continued on next page)

Table 15–1 (Cont.) Resource Manager Routines

Routine Name	Description
RM Q GET SEARCH RESOURCE	Searches for resource database levels for a given resource.
RM Q PUT RESOURCE	Stores a resource in the database.
RM Q PUT STRING RESOURCE	Adds a string resource, using quarks as a specification.
RM QUARK TO STRING	Converts a quark to a string.
RM STRING TO BIND QUARK LIST	Converts a string with one or more components to a binding list and a quark list.
RM STRING TO QUARK	Converts a string to a quark.
RM STRING TO QUARK LIST	Converts a string with one or more components to a quark list.
RM UNIQUE QUARK	Allocates a new quark.

15.1 The Resource Manager

The resource manager is a database manager but with a difference. In most database systems, you perform a query using an imprecise specification and get back a set of records. The resource manager, however, allows you to specify a large set of values with an imprecise specification, to query the database with a precise specification, and to get back only a single value. This should be used by applications that need to know what the user prefers for colors, fonts, and other resources.

For example, a user of your application may want to specify that all windows should have a blue background but that all mail reading windows should have a red background. Presuming that all applications use the resource manager, a user can define this information using only two lines of specification. Your personal resource database usually is stored in a file and is loaded onto a server property when you log in. This database is retrieved automatically by Xlib when a connection is opened.

As an example of how the resource manager works, consider a mail reading application called *xmh*. Assume that it is designed in such a manner that it uses a complex window hierarchy all the way down to individual command buttons, which may be actual small subwindows in some toolkits. These are often called *objects*. In such toolkit systems, user interface objects (called *widgets* in the X toolkit) can be composed of other objects. Each user interface *object* can be assigned a name and a class. Fully qualified names or classes can have arbitrary numbers of component names, but a fully qualified name always has the same number of component names as a fully qualified class. This generally reflects the structure of the application as composed of these objects, starting with the application itself.

For example, the xmh mail program has a name, *xmh*, and is one of a class of *Mail* programs. By convention, the first character of a class component is capitalized, while the first letter of a name component is in lowercase. Each name and class finally have an attribute (for example *foreground* or *font*). If each window is properly assigned a name and class, it becomes easy for the user to specify attributes of any portion of the application.

At the top level, the application might consist of a paned window (that is, a window divided into several sections) named *toc*. One pane of the paned window is a button box window named *buttons* filled with command buttons. One of these command buttons is used to retrieve (*include*) new mail and has the name *include*. This window has a fully qualified name, *xmh.toc.buttons.include*, and a fully qualified class, *Xmh.VPaned.Box.Command*. Its fully qualified name is the name of its parent, *xmh.toc.buttons*, followed by its name, *include*. Its class is the class of its parent, *Xmh.VPaned.Box*, followed by its particular class, *Command*. The fully qualified name of a resource is the attribute's name appended to the object's fully qualified name, and the fully qualified class is its class appended to the object's class.

This include button needs the following resources:

- Title string
- Font
- Foreground color for its inactive state
- Background color for its inactive state
- Foreground color for its active state
- Background color for its active state

Each of the resources that this button needs is considered to be an attribute of the button and, as such, has a name and a class. For example, the foreground color for the button in its active state might be named *activeForeground*, and its class would be *Foreground.*

When an application looks up a resource (for example, a color), it passes the complete name and complete class of the resource to a lookup routine. After lookup, the resource manager returns the resource value and the representation type.

The resource manager allows applications to store resources by an incomplete specification of name, class, and representation type, as well as to retrieve them given a fully qualified name and class.

15.2 Resource Manager Matching Rules

The algorithm for determining which resource name or names match a given query is the heart of the database. Resources are stored with only partially specified names and classes, using pattern matching constructs. An asterisk is used to represent any number of intervening components (including none). A dot or period is used to separate immediately adjacent components. All queries fully specify the name and class of the resource needed. The lookup algorithm then searches the database for the name that most closely matches (is most specific) to this full name and class. In order of precedence, the rules for a match are as follows:

1 The attributes of the name and class must match. For example, the following queries will not match the database entry *xterm.scrollbar:on.*

```
xterm.scrollbar.background      (name)
XTerm.Scrollbar.Background      (class)
```

2 Database entries with name or class prefixed by a period are more specific than those prefixed by an asterisk. For example, the entry *xterm.geometry* is more specific than the entry *xterm*geometry.*

3 Names are more specific than classes. For example, the entry **scrollbar.background* is more specific than the entry **Scrollbar.Background.*

4 A name or class is more specific than omission. For example, the entry *Scrollbar*Background* is more specific than the entry **Background.*

5 Left components are more specific than right components. For example, **vt100*background* is more specific than the entry **scrollbar*background*, for the query *.vt100.scrollbar.background.*

6 If neither a period nor an asterisk is specified at the beginning, a period is implicit. For example, *xterm.background* is identical to *.xterm.background.*

Names and classes can be mixed. As an example of these rules, assume the following user preference specification:

```
xmh*background: red
*command.font: x
*command.background: blue
*Command.Foreground: green
xmh.toc*Command.activeForeground:black
```

A query for the name *xmh.toc.messagefunctions.include.activeForeground* and class *Xmh.VPaned.Box.Command.Foreground* would match *xmh.toc*Command.activeForeground* and return *black.* However, it also matches **Command.Foreground.* Using the precedence algorithm described above, the resource manager would return the value specified by *xmh.toc*Command.activeForeground.*

15.3 Quarks

Most uses of the resource manager involve defining names, classes, and representation types as string constants. However, always referring to strings in the resource manager can slow performance. To improve performance, the resource manager uses a shorthand name for a string during many resource manager functions. Simple comparisons can then be performed, rather than string comparisons. The shorthand name for a string is called a *quark,* and is the type X$RM_QUARK or XrmQuark. (Quarks can also be called representations.) On some occasions, you may want to allocate a quark that has no string equivalent. A quark is to a string what an atom is to a property name in the server, but its use is entirely local to your application.

15.4 The Resource Manager Value Data Structure

The definitions for the resource manager's use are contained in the *Xresource.h* header file. Xlib also uses the resource manager internally to allow for non-English-language error messages.

The resource manager value data structure defines database values. Database values consist of a size, an address, and a representation type. The size is specified in bytes. The representation type is a way for you to store data tagged by some application-defined type (for example, font or color). It has nothing to do with the MIT C data type or with its class.

The resource manager value data structure for the VAX binding is shown in Figure 15–1, and members of the data structure are described in Table 15–2.

Figure 15–1 Resource Manager Value Data Structure (VAX Binding)

x$l_rval_size	0
x$a_rval_addr	4

Table 15–2 Members of the Resource Manager Value Data Structure (VAX Binding)

Member Name	Contents
X$L_RVAL_SIZE	Size of the database
X$L_RVAL_ADDR	Address of the database

The resource manager value data structure for the MIT C binding is shown in Figure 15–2, and members of the data structure are described in Table 15–3.

Figure 15–2 Resource Manager Value Data Structure (MIT C Binding)

```
typedef struct {
        unsigned int size;
        caddr_t addr;
} XrmValue, *XrmValuePtr:
```

Table 15–3 Members of the Resource Manager Value Data Structure (MIT C Binding)

Member Name	Contents
size	Size of the database
address	Address of the database

15.5 Resource Manager Routines

The following pages describe the Xlib resource manager routines.

PERMALLOC

Allocates memory for permanently allocated storage.

VAX FORMAT

location_return = **X$PERM_ALLOC** *(size)*

argument information

Argument	Usage	Data Type	Access	Mechanism
location_return	longword	longword	write	value
size	longword	longword	read	reference

MIT C FORMAT

location_return = **Xpermalloc** *(size)*

argument information

```
char *Xpermalloc(size)
        unsigned int size;
```

RETURNS

location_return
The location of the allocated storage.

ARGUMENTS

size
The size, in bytes, of the storage.

DESCRIPTION

PERMALLOC allocates memory space for permanently allocated storage.

RM GET FILE DATABASE

Retrieves a database from nonvolatile storage.

VAX FORMAT

database_id_return = **X$RM_GET_FILE_DATABASE** *(file_name)*

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id_return	identifier	uns longword	write	value
file_name	char string	char string	read	descriptor

MIT C FORMAT

database_id_return = **XrmGetFileDatabase** *(file_name)*

argument information

```
XrmDatabase XrmGetFileDatabase(file_name)
        char *file_name;
```

RETURNS

database_id_return
The identifier of the returned file database.

ARGUMENTS

file_name
The resource database file name.

DESCRIPTION

RM GET FILE DATABASE retrieves a database from nonvolatile storage. This function opens the file specified by **file_name**, creates a new resource database, and loads it with the specifications read in from the specified file. The specified file must contain lines in the format accepted by RM PUT LINE RESOURCE. If it cannot open the specified file, RM GET FILE DATABASE returns a null value.

RM GET RESOURCE

Retrieves a resource from the database.

VAX FORMAT

status_return = **X$RM_GET_RESOURCE** *(database_id, name_list_string, class_list_string, repr_type_return [,repr_value_return] [,buf_len] [,val_buf_return] [,len_return])*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
database_id	identifier	uns longword	read	reference
name_list_string	char string	char string	read	descriptor
class_list_string	char string	char string	read	descriptor
repr_type_return	char string	char string	write	descriptor
repr_value_return	record	x$rm_value	write	reference
buf_len	longword	longword	read	reference
val_buf_return	vector longword	uns longword	write	reference
len_return	longword	longword	write	reference

MIT C FORMAT

XrmGetResource *(database_id, name_list_string, class_list_string, repr_type_return, repr_value_return)*

argument information

```
XrmGetResource(database_id, name_list_string, class_list_string,
repr_type_return, repr_value_return)
      XrmDatabase  database_id;
      char * name_list_string;
      char * class_list_string;
      char **repr_type_return;
      XrmValue  *repr_value_return;
```

RETURNS

status_return
Return value that specifies whether or not the routine completed successfully.

ARGUMENTS

database_id

The descriptor of the resource database.

name_list_string

The full inheritance name of the value being retrieved.

class_list_string

The full inheritance class of the value being retrieved.

repr_type_return

The representation type of the destination.

repr_value_return

The descriptor into which the value is returned, representing the address of the database. This argument is optional in the VAX binding only.

buf_len (VAX only)

The length of the buffer in which the value is returned. This argument is optional.

val_buf_return (VAX only)

The address of the buffer containing the returned value. This argument is optional.

len_return (VAX only)

The length of the returned value contained in the return value buffer. This argument is optional.

DESCRIPTION

RM GET RESOURCE retrieves a resource from the specified database. The value returned points to database memory.

RM GET STRING DATABASE

Creates a database from a string.

VAX FORMAT

database_id_return =
X$RM_GET_STRING_DATABASE
(contents_name)

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id_return	identifier	uns longword	write	value
contents_name	char string	char string	read	descriptor

MIT C FORMAT

database_id_return = **XrmGetStringDatabase**
(contents_name)

argument information

```
XrmDatabase XrmGetStringDatabase (contents_name)
        char *contents_name;
```

RETURNS

database_id_return
The identifier of the created database.

ARGUMENTS

contents_name
The string that specifies the contents of the database.

DESCRIPTION

RM GET STRING DATABASE creates a new database and fills it with the resources in the specified null-terminated string. RM GET STRING DATABASE is similar to RM GET FILE DATABASE, except that it reads the information out of a string instead of a file. Each line is separated by a new-line character in the format accepted by RM PUT LINE RESOURCE.

RM INITIALIZE

Initializes the resource manager.

VAX FORMAT

status_return = **X$RM_INITIALIZE**

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value

MIT C FORMAT

XrmInitialize

argument information

```
void XrmInitialize( )
```

RETURNS

status_return (VAX only)
Return value that states whether or not the routine completed successfully.

DESCRIPTION

RM INITIALIZE initializes the resource manager.

RM MERGE DATABASES

Merges the contents of one database into another.

VAX FORMAT

X$RM_MERGE_DATABASES
(src_database_id, dst_database_id)

argument information

Argument	Usage	Data Type	Access	Mechanism
src_database_id	longword	uns longword	read	reference
dst_database_id	longword	uns longword	modify	reference

MIT C FORMAT

XrmMergeDatabases
(src_database_id, dst_database_id)

argument information

```
XrmMergeDatabases(src_database_id,dst_database_id)
     XrmDatabase src_database_id, *dst_database_id;
```

ARGUMENTS

src_database_id

The descriptor of the resource database to be merged into the existing database.

dst_database_id

The descriptor of the resource database into which the new database will be merged.

DESCRIPTION

RM MERGE DATABASES merges the contents of one database into another. The function may overwrite entries in the destination database. This procedure is used to combine databases, for example, an application-specific database of defaults and a database of user preferences.

RM PARSE COMMAND

Loads a resource database from a command line.

VAX FORMAT

X$PARSE_COMMAND
(database_id, options, num_options, prefix_name, argc, argv)

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	identifier	uns longword	read	reference
options	any	v uns longword	read	reference
num_options	longword	longword	read	reference
prefix_name	char string	char string	read	descriptor
argc	longword	longword	read	modify
argv	any	byte	read	modify

MIT C FORMAT

XrmParseCommand
(database_id, options, num_options, prefix_name, argc, argv)

argument information

```
void XrmParseCommand (database_id, options, num_options,
                      prefix_name, argc, argv)
        XrmDatabase *database_id;
        XrmOptionList options;
        int num_options;
        char *prefix_name;
        int *argc;
        char **argv;
```

ARGUMENTS

database_id

A pointer to the resource database. If the database contains a null value, a new resource database is created and a pointer to it is returned in the database.

options

The table containing a list of command line arguments to be parsed.

num_options

The number of entries in the table specified in **options**.

prefix_name

The prefix to be appended to the resources.

argc

Argument that specifies the number of arguments in the command line and returns the number of remaining arguments.

argv

Argument that specifies a pointer to the command line arguments and returns the remaining arguments.

DESCRIPTION

RM PARSE COMMAND loads a resource database from a command line. RM PARSE COMMAND parses an (**argc**, **argv**) pair according to the specified option table, loads recognized options into the specified database, and modifies the (**argc**, **argv**) pair to remove all recognized options.

The specified table is used to parse the command line. Recognized entries in the table are removed from **argv**, and entries are made in the specified resource database. The table entries contain information on the option string, the option name, which style of option, and a value to provide if the option kind is XrmoptionNoArg. The **argc** argument specifies the number of arguments in **argv** and is set to the remaining number of arguments that were not parsed. The **name** argument should be the name of your application for use in building the database entry. The **name** argument is prefixed to the resource name in the option table before the resource manager stores the specification. No separating (binding) character is inserted. The table must contain either a period or an asterisk as the first character in the resource name entry. To specify a more completely qualified resource name, the resource name entry can contain multiple components.

RM PUT FILE DATABASE

Stores a copy of the application's current database in nonvolatile storage.

VAX FORMAT

X$RM_PUT_FILE_DATABASE
(database_id, file_name)

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	identifier	uns longword	read	reference
file_name	char string	char string	read	descriptor

MIT C FORMAT

XRmPutFileDatabase
(database_id, file_name)

argument information

```
XRmPutFileDatabase (database_id, file_name)
        XrmDatabase database_id;
        char *file_name;
```

ARGUMENTS

database_id
The identifier of the application's current database.

file_name
The file name for the stored database.

DESCRIPTION

RM PUT FILE DATABASE stores a copy of the application's current database in the file specified by **file_name**. This file is an ASCII text file. The file contains lines in the format that is accepted by RM PUT LINE RESOURCE.

RM PUT LINE RESOURCE

Adds a single resource entry to a specified database.

VAX FORMAT

X$RM_PUT_LINE_RESOURCE
(database_id, resource_line)

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	record	uns longword	modify	reference
resource_line	char string	char string	read	descriptor

MIT C FORMAT

XrmPutLineResource
(database_id, resource_line)

argument information

```
XrmPutLineResource(database_id, resource_line)
        XrmDatabase *database_id;
        char *resource_line;
```

ARGUMENTS

database_id

A pointer to the resource database. If the database contains a null value, a new resource database is created and a pointer to it is returned in the database.

resource_line

The resource/value pair as a single string. A colon separates the name from the value.

DESCRIPTION

RM PUT LINE RESOURCE adds a resource entry to the specified database. Any space before or after the name or colon in the **resource_line** argument is ignored. The value is terminated by a new-line or a null character.

To allow values to contain embedded new-line characters, a line-feed character (\n) is recognized and replaced by a new-line character. For example, a line might have the value

```
xterm*background:green\n
```

This adds an extra byte to the length of the return value. Null-terminated strings without a new line are also permitted.

RM PUT RESOURCE

Stores a resource in the database.

VAX FORMAT

X$RM_PUT_RESOURCE
(database_id, specifier_name, type_name [,resource_value] [,buf_len] [,val_buf])

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	identifier	uns longword	modify	descriptor
specifier_name	char string	char string	read	descriptor
type_name	char string	char string	read	descriptor
resource_value	record	x$rm_value	read	reference
buf_len	longword	longword	read	reference
val_buf	uns longword	uns longword	read	reference

MIT C FORMAT

XrmPutResource
(database_id, specifier_name, type_name, resource_value)

argument information

```
XrmPutResource(database_id,specifier_name,type_name,
               resource_value)
      XrmDatabase  *database_id;
      char *specifier_name;
      char *type_name;
      XrmValue  *resource_value;
```

ARGUMENTS

database_id

The resource database.

specifier_name

The partial name or class list of the resource to be stored.

type_name

The data representation of the resource to be stored.

resource_value

The descriptor for the resource entry. This argument is optional in the VAX binding only.

buf_len (VAX only)

Length of the value buffer. This argument is optional.

val_buf (VAX only)

Address of the value buffer. This argument is optional.

DESCRIPTION

RM PUT RESOURCE stores a resource database, specified by descriptor, in a file on nonvolatile storage. The value is copied into the specified database.

RM PUT STRING RESOURCE

Adds a resource that is specified as a string.

VAX FORMAT

X$RM_PUT_STRING_RESOURCE
(database_id, resource_name, value_name)

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	identifier	uns longword	modify	reference
resource_name	char string	char string	read	descriptor
value_name	char string	char string	read	descriptor

MIT C FORMAT

XrmPutStringResource
(database_id, resource_name, value_name)

argument information

```
XrmPutStringResource(database_id,resource_name,value_name)
      XrmDatabase  *database_id;
      char *resource_name;
      char  *value_name;
```

ARGUMENTS

database_id

A pointer to the resource database. If the database contains a null value, a new resource database is created and a pointer to it is returned in the database.

resource_name

A character string that specifies the name of the resource.

value_name

A character string that specifies the value of the resource.

DESCRIPTION

RM PUT STRING RESOURCE adds a resource with a specified value to a database. RM PUT STRING RESOURCE takes both **resource_name** and **value_name** as null-terminated strings, converts them to quarks, and then calls RM Q PUT RESOURCE, using a string representation type.

RM Q GET RESOURCE

Retrieves a resource from the database.

VAX FORMAT

status_return = **X$RM_Q_GET_RESOURCE** *(database_id, name_list_id, class_list_id, repr_type_id_return, repr_value_id_return, buf_len, val_buf_return, len_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
database_id	identifier	uns longword	read	reference
name_list_id	identifier	uns longword	read	reference
class_list_id	identifier	uns longword	read	reference
repr_type_id_return	identifier	uns longword	read	reference
repr_value_id_return	record	x$rm_value	write	reference
buf_len	longword	longword	read	reference
val_buf_return	record	byte	write	reference
len_return	longword	longword	write	reference

MIT C FORMAT

XrmQGetResource *(database_id, name_list_id, class_list_id, repr_type_id_return, repr_value_id_return)*

argument information

```
XrmQGetResource(database_id, name_list_id, class_list_id,
repr_type_id_return, repr_value_id_return)
      XrmDatabase  *database_id;
      XrmNameList  name_list_id;
      XrmClassList  class_list_id;
      XrmRepresentation  *repr_type_id_return;
      XrmValue  *repr_value_id_return;
```

ARGUMENTS

database_id
The descriptor of the resource database.

name_list_id
The full inheritance name of the value being retrieved.

class_list_id

The full inheritance class of the value being retrieved.

repr_type_id_return

The representation type of the destination.

repr_value_id_return

The descriptor into which the value is returned, representing the address of the database.

buf_len (VAX only)

The length of the buffer in which the value is returned.

val_buf_return (VAX only)

The address of the buffer containing the returned value.

len_return (VAX only)

The length of the returned value contained in the return value buffer.

DESCRIPTION

RM Q GET RESOURCE retrieves a resource from the specified database. The value returned points to database memory.

RM Q GET SEARCH LIST

Returns a list of database levels.

VAX FORMAT

X$RM_Q_GET_SEARCH_LIST
(database_id, name_list_id, class_list_id, search_list_id_return, list_len)

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	identifier	uns longword	read	reference
name_list_id	identifier	uns longword	read	reference
class_list_id	identifier	uns longword	read	reference
search_list_id_return	identifier	uns longword	write	reference
list_len	uns longword	uns longword	read	reference

MIT C FORMAT

XrmQGetSearchList
(database_id, name_list_id, class_list_id, search_list_id_return, list_len)

argument information

```
Bool XrmQGetSearchList (database_id, name_list_id, class_list_id,
                        search_list_id_return, list_len)
      XrmDatabase database_id;
      XrmNameList name_list_id;
      XrmClassList class_list_id;
      XrmSearchList search_list_id_return;
      int list_len;
```

RETURNS

Bool (MIT C only)
Boolean argument that specifies whether the size of **search_list_id_return** is large enough. RM Q GET SEARCH LIST returns true to this argument if **search_list_id_return** is large enough, and returns false if it is not.

ARGUMENTS

database_id
The identifier of the database to be used.

name_list_id
A list of resource names.

class_list_id

A list of resource classes.

search_list_id_return

The search list of database levels that is returned.

list_len

The number of entries allocated for **search_list_id_return**.

DESCRIPTION

RM Q GET SEARCH LIST takes a list of resource names and resource classes and returns a list of database levels where a match may occur. The list uses the same algorithm as RM GET RESOURCE for determining precedence.

The size required for **search_list_id_return** is dependent on the number of levels and wildcards in the resource specifiers that are stored in the database.

When you use RM Q GET SEARCH LIST before multiple searches for resources with a common name and class prefix, you should specify only the common prefix in the **name_list_id** and **class_list_id** arguments.

RM Q GET SEARCH RESOURCE

Searches for resource database levels for a given resource.

VAX FORMAT

X$RM_Q_GET_SEARCH_RESOURCE
(search_list_id, name_id, class_id, repr_type_id_return [,repr_value_return] [,buf_len] [,val_buf_return] [,ret_len_return])

argument information

Argument	Usage	Data Type	Access	Mechanism
search_list_id_return	identifier	uns longword	read	reference
name_id	identifier	uns longword	read	reference
class_id	identifier	uns longword	read	reference
repr_type_id_return	identifier	uns longword	write	reference
repr_value_return	record	xrm$value	write	reference
buf_len	longword	longword	read	reference
val_buf_return	record	byte	write	descriptor
ret_len_return	longword	longword	write	reference

MIT C FORMAT

XrmQGetSearchResource
(search_list_id, name_id, class_id, repr_type_id_return [,repr_value_return])

argument information

```
Bool XrmQGetSearchResource (search_list_id, name_id, class_id,
repr_type_id_return [,repr_value_return])
        XrmSearchList search_list_id;
        XrmName name_id;
        XrmClass class_id;
        XrmRepresentation *repr_type_id_return;
        XrmValue *repr_value_return;
```

RETURNS

Bool (MIT C only)
An argument that specifies whether the resource was found. RM Q GET SEARCH RESOURCE returns true if the resource is found, and false if the resource is not found.

ARGUMENTS

search_list_id
The search list returned from GET SEARCH LIST.

name_id
The resource name.

class_id
The resource class.

repr_type_id_return
The returned data representation type.

repr_value_return
The returned value descriptor. This argument is optional in the VAX binding only.

buf_len (VAX only)
The length of the following buffer. This argument is optional.

val_buf_return (VAX only)
The returned buffer containing the value in the database. This argument is optional.

ret_len_return (VAX only)
The length of the data written to the buffer. This argument is optional.

DESCRIPTION

RM Q GET SEARCH RESOURCE searches the specified database levels for the resource that is identified by **name_id** and **class_id**. The search stops when a match is found.

A call to RM Q GET SEARCH LIST with a name and class list containing all but the last component of a resource name, followed by a call to RM Q GET SEARCH RESOURCE with the last component name and class, returns the same database entry as RM GET RESOURCE or RM Q GET RESOURCE with the fully qualified name and class.

RM Q PUT RESOURCE

Stores a resource in the database.

VAX FORMAT

X$RM_Q_PUT_RESOURCE
(database_id, binding_list_id, repr_list_id, repr_type_id, repr_value, val_len, val_buf)

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	identifier	uns longword	modify	reference
binding_list_id	identifier	uns longword	read	reference
repr_type_id	identifier	uns longword	read	reference
repr_value	record	x$rm_value	read	reference
val_len	longword	longword	read	reference
val_buf	any	vector uns longword	read	reference

MIT C FORMAT

XrmQPutResource
(database_id, binding_list_id, repr_type_id, repr_value)

argument information

```
XrmQPutResource(database_id,binding_list_id,repr_type_id,
                repr_value)
      XrmDatabase  *database_id;
      XrmBindingList  binding_list_id;
      XrmRepresentation  repr_type_id;
      XrmValue  repr_value;
```

ARGUMENTS

database_id
Identifier of the resource database.

binding_list_id
A list of bindings defining the resource.

repr_type_id
Identifier of the data representation of the resource to be stored.

repr_value
The descriptor for the resource entry. This argument is optional in the VAX binding only.

val_len (VAX only)

Length of the value buffer. This argument is optional.

val_buf (VAX only)

Address of the value buffer. This argument is optional.

DESCRIPTION

RM Q PUT RESOURCE stores a resource database, specified by descriptor, in a file on nonvolatile storage. The value is copied into the specified database.

RM Q PUT STRING RESOURCE

Adds a string resource, using quarks as a specification.

VAX FORMAT

X$RM_Q_PUT_STRING_RESOURCE *(database_id, binding_list_id, repr_list_id, value_name)*

argument information

Argument	Usage	Data Type	Access	Mechanism
database_id	identifier	uns longword	modify	reference
binding_list_id	identifier	uns longword	read	reference
repr_list_id	identifier	uns longword	read	reference
value_name	char string	char string	read	descriptor

MIT C FORMAT

XrmQPutStringResource *(database_id, binding_list_id, repr_list_id, value_name)*

argument information

```
XrmQPutStringResource (database_id, binding_list_id,
                       repr_list_id, value_name)
        XrmDatabase  *database_id;
        XrmBindingList  binding_list_id;
        XrmQuarkList  repr_list_id;
        char  *value_name;
```

ARGUMENTS

database_id

A pointer to the resource database. If the database contains a null value, a new resource database is created and a pointer to it is returned in the database. If the resource database is null, a new database is created.

binding_list_id

A list of bindings defining the resource.

repr_list_id

A list of quarks defining the resource.

value_name

A character string that specifies the value of the resource.

DESCRIPTION

RM Q PUT STRING RESOURCE adds a string resource, using quarks as a specification. This routine constructs a resource manager value data structure for the value string, and then calls RM Q PUT RESOURCE, using a string representation type.

RM QUARK TO STRING

Converts a quark to a string.

VAX FORMAT

X$RM_QUARK_TO_STRING
(repr_id, repr_name)

argument information

Argument	Usage	Data Type	Access	Mechanism
repr_id	identifier	uns longword	read	reference
repr_name	char string	char string	write	descriptor

MIT C FORMAT

repr_id_return = **XrmQuarkToString**
(repr_name)

argument information

```
char XrmQuarkToString(repr_id)
     XrmQuark repr_id;
```

RETURNS

repr_id_return (MIT C only)

The string returned for the quark specified in **repr_name**.

ARGUMENTS

repr_id

The quark for which you want to obtain an equivalent string.

repr_name (VAX only)

The string returned for the quark specified in **repr_id**.

DESCRIPTION

RM QUARK TO STRING converts a quark to a string. The string returned in **repr_name** must not be modified or freed. If no equivalent string exists for **repr_id**, RM QUARK TO STRING returns a null value.

RM STRING TO BIND QUARK LIST

Converts a string with one or more components to a binding list and a quark list.

VAX FORMAT

X$RM_STRING_TO_BIND_QUARK_LIST *(value_name, binding_list_id_return, repr_list_id_return)*

argument information

Argument	Usage	Data Type	Access	Mechanism
value_name	char string	char string	read	descriptor
binding_list_id_return	identifier	uns longword	write	reference
repr_list_id_return	identifier	uns longword	write	reference

MIT C FORMAT

XrmStringToBindingQuarkList *(value_name, binding_list_id_return, repr_list_id_return)*

argument information

```
XrmStringToBindingQuarkList(value_name, binding_list_id_return,
                            repr_list_id_return)
        char  *value_name;
        XrmBindingList binding_list_id_return;
        XrmQuarkList repr_list_id_return;
```

ARGUMENTS

value_name

The name of the character string for which a quark is to be allocated.

binding_list_id_return

The returned binding list. The caller must allocate sufficient space for the binding list before calling RM STRING TO BIND QUARK LIST.

repr_list_id_return

The returned quarks list. The caller must allocate sufficient space for the quarks list before calling RM STRING TO BIND QUARK LIST.

DESCRIPTION

RM STRING TO BIND QUARK LIST converts a string with one or more components to a binding list and a quark list. Component names in the list must be separated by either a period or an asterisk.

RM STRING TO QUARK

Converts a string to a quark.

VAX FORMAT

repr_id_return = **X$RM_STRING_TO_QUARK** *(repr_name)*

argument information

Argument	Usage	Data Type	Access	Mechanism
repr_id_return	identifier	uns longword	write	value
repr_name	char string	char string	read	descriptor

MIT C FORMAT

repr_id_return = **XrmStringToQuark** *(repr_name)*

argument information

```
XrmQuark XrmStringToQuark (repr_name)
        char *repr_name
```

RETURNS

repr_id_return
The identifier of the quark allocated for the specified string.

ARGUMENTS

repr_name
The string for which a quark is to be allocated.

DESCRIPTION

RM STRING TO QUARK converts a string to a quark.

RM STRING TO QUARK LIST

Converts a string with one or more components to a quark list.

VAX FORMAT

X$RM_STRING_TO_QUARK_LIST
(repr_name, repr_list_id_return)

argument information

Argument	Usage	Data Type	Access	Mechanism
repr_name	char string	char string	read	descriptor
repr_list_id_return	identifier	uns longword	write	reference

MIT C FORMAT

XrmStringToQuarkList
(repr_name, repr_list_id_return)

argument information

```
XrmStringToQuarkList(repr_name, repr_list_id_return)
        char  *repr_name;
        XrmQuarkList  repr_list_id_return;
```

ARGUMENTS

repr_name
The string for which a quark is to be allocated.

repr_list_id_return
The returned quark list.

DESCRIPTION

RM STRING TO QUARK LIST converts a string with one or more components to a quark list. The component names in the list are separated by either a period or an asterisk.

RM UNIQUE QUARK

Allocates a new quark.

VAX FORMAT

repr_id_return = **X$RM_UNIQUE_QUARK** *()*

argument information

Argument	Usage	Data Type	Access	Mechanism
repr_id_return	identifier	uns longword	write	value

MIT C FORMAT

repr_id_return = **XrmUniqueQuark** *()*

argument information

```
XrmQuark XrmUniqueQuark( )
```

RETURNS

repr_id_return
The identifier of the new quark.

DESCRIPTION

RM UNIQUE QUARK allocates a new quark. This quark does not represent any representation type that is known to the resource manager.

Index

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local DIGITAL subsidiary or approved distributor
Internal[1]	————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1] For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VMS DECwindows
Xlib Routines
Reference Manual
Part I: AA–MG26A–TE
Part II: AA–MG27A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	□	□	□	□
Completeness (enough information)	□	□	□	□
Clarity (easy to understand)	□	□	□	□
Organization (structure of subject matter)	□	□	□	□
Figures (useful)	□	□	□	□
Examples (useful)	□	□	□	□
Index (ability to find topic)	□	□	□	□
Page layout (easy to find information)	□	□	□	□

I would like to see more/less ______________________________

____

____

What I like best about this manual is ______________________________

____

____

What I like least about this manual is ______________________________

____

____

I found the following errors in this manual:

Page Description

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

Additional comments or suggestions to improve this manual:

____

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ____________

Company ______________________________ Date ____________

Mailing Address ______________________________

______________________________ Phone ____________

— Do Not Tear - Fold Here and Tape ——————————————————————

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here ——————————————————————

Reader's Comments

VMS DECwindows
Xlib Routines
Reference Manual
Part I: AA–MG26A–TE
Part II: AA–MG27A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	□	□	□	□
Completeness (enough information)	□	□	□	□
Clarity (easy to understand)	□	□	□	□
Organization (structure of subject matter)	□	□	□	□
Figures (useful)	□	□	□	□
Examples (useful)	□	□	□	□
Index (ability to find topic)	□	□	□	□
Page layout (easy to find information)	□	□	□	□

I would like to see more/less ______________________________

____

____

What I like best about this manual is ______________________________

____

____

What I like least about this manual is ______________________________

____

____

I found the following errors in this manual:

Page Description

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

Additional comments or suggestions to improve this manual:

____

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ______________

Company ______________________________ Date ______________

Mailing Address ______________________________

______________________________ Phone ______________

— Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here