# Xmu Library

X Version 11, Release 6.4

"*Don't ask.*"

Copyright © 1989 X Consortium

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the X Consortium.

*X Window System* is a trademark of X Consortium, Inc.

## 1. Introduction

The Xmu Library is a collection of miscellaneous (some might say random) utility functions that have been useful in building various applications and widgets. This library is required by the Athena Widgets.

## 2. Atom Functions

The use the functions and macros defined in this section, you should include the header file **<X11/Xmu/Atoms.h>**.

XA_ATOM_PAIR(*d*)
XA_CHARACTER_POSITION(*d*)
XA_CLASS(*d*)
XA_CLIENT_WINDOW(*d*)
XA_CLIPBOARD(*d*)
XA_COMPOUND_TEXT(*d*)
XA_DECNET_ADDRESS(*d*)
XA_DELETE(*d*)
XA_FILENAME(*d*)
XA_HOSTNAME(*d*)
XA_IP_ADDRESS(*d*)
XA_LENGTH(*d*)
XA_LIST_LENGTH(*d*)
XA_NAME(*d*)
XA_NET_ADDRESS(*d*)
XA_NULL(*d*)
XA_OWNER_OS(*d*)
XA_SPAN(*d*)
XA_TARGETS(*d*)
XA_TEXT(*d*)
XA_TIMESTAMP(*d*)
XA_USER(*d*)
XA_UTF8_STRING(*d*)

These macros take a display as argument and return an **Atom**. The name of the atom is obtained from the macro name by removing the leading characters "XA_". The **Atom** value is cached, such that subsequent requests do not cause another round-trip to the server.

AtomPtr XmuMakeAtom(*name*)
    char* *name*;

*name* specifies the atom name

This function creates and initializes an opaque object, an **AtomPtr**, for an **Atom** with the given name. **XmuInternAtom** can be used to cache the Atom value for one or more displays.

char *XmuNameOfAtom(*atom_ptr*)
    AtomPtr *atom_ptr*;

*atom_ptr* specifies the AtomPtr

The function returns the name of an AtomPtr.

Atom XmuInternAtom(*d*, *atom_ptr*)
    Display **d*;
    AtomPtr *atom_ptr*;

*d* specifies the connection to the X server

*atom_ptr* specifies the AtomPtr

This function returns the **Atom** for an **AtomPtr**. The **Atom** is cached, such that subsequent requests do not cause another round-trip to the server.

char *XmuGetAtomName(*d*, *atom*)
    Display **d*;
    Atom *atom*;

*d* specifies the connection to the X server

*atom* specifies the atom whose name is desired

This function returns the name of an **Atom**. The result is cached, such that subsequent requests do not cause another round-trip to the server.

void XmuInternStrings(*d*, *names*, *count*, *atoms*)
    Display **d*;
    String **names*;
    Cardinal *count*;
    Atom **atoms*;

*d* specifies the connection to the X server

*names* specifies the strings to intern

*count* specifies the number of strings

*atoms* returns the list of Atom values

This function converts a list of atom names into **Atom** values. The results are cached, such that subsequent requests do not cause further round-trips to the server. The caller is responsible for preallocating the array pointed at by atoms.

### 3. Error Handler Functions

To use the functions defined in this section, you should include the header file <**X11/Xmu/Error.h**>.

int XmuPrintDefaultErrorMessage(*dpy*, *event*, *fp*)
    Display **dpy*;
    XErrorEvent **event*;
    FILE **fp*;

*dpy* specifies the connection to the X server

*event* specifies the error

*fp* specifies where to print the error message

This function prints an error message, equivalent to Xlib's default error message for protocol errors. It returns a non-zero value if the caller should consider exiting, otherwise it returns 0. This function can be used when you need to write your own error handler, but need to print out an error from within that handler.

int XmuSimpleErrorHandler(*dpy*, *errorp*)
    Display **dpy*;
    XErrorEvent **errorp*;

*dpy* specifies the connection to the X server

*errorp* specifies the error

This function ignores errors for **BadWindow** errors for **XQueryTree** and **XGetWindowAttributes**, and ignores **BadDrawable** errors for **XGetGeometry**; it returns 0 in those cases. Otherwise, it prints the default error message, and returns a non-zero value if the caller should consider exiting, and 0 if the caller should not exit.

### 4. System Utility Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/SysUtil.h>**.

int XmuGetHostname(*buf*, *maxlen*)
    char **buf*;
    int *maxlen*;

| | |
|---|---|
| *buf* | returns the host name |
| *maxlen* | specifies the length of buf |

This function stores the null terminated name of the local host in buf, and returns length of the name. This function hides operating system differences, such as whether to call gethostname or uname.

### 5. Window Utility Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/WinUtil.h>**.

Screen *XmuScreenOfWindow(*dpy*, *w*)
    Display **dpy*;
    Window *w*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *w* | specifies the window |

This function returns the **Screen** on which the specified window was created.

Window XmuClientWindow(*dpy*, *win*)
    Display **dpy*;
    Window *win*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *win* | specifies the window |

This function finds a window, at or below the specified window, which has a WM_STATE property. If such a window is found, it is returned, otherwise the argument window is returned.

Bool XmuUpdateMapHints(*dpy*, *w*, *hints*)
    Display **dpy*;
    Window *w*;
    XSizeHints **hints*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *win* | specifies the window |
| *hints* | specifies the new hints, or NULL |

This function clears the **PPosition** and **PSize** flags and sets the **USPosition** and **USSize** flags in the hints structure, and then stores the hints for the window using **XSetWMNormalHints** and returns **True**. If NULL is passed for the hints structure, then the current hints are read back from the window using **XGetWMNormalHints** and are used instead, and **True** is returned; otherwise **False** is returned.

## 6. Cursor Utility Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/CurUtil.h>**.

int XmuCursorNameToIndex(*name*)
    char **name*;

*name* specifies the name of the cursor

This function takes the name of a standard cursor and returns its index in the standard cursor font. The cursor names are formed by removing the "XC_" prefix from the cursor defines listed in Appendix B of the Xlib manual.

## 7. Graphics Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/Drawing.h>**.

void XmuDrawRoundedRectangle(*dpy*, *draw*, *gc*, *x*, *y*, *w*, *h*, *ew*, *eh*)
    Display **dpy*;
    Drawable *draw*;
    GC *gc*;
    int *x*, *y*, *w*, *h*, *ew*, *eh*;

*dpy* specifies the connection to the X server

*draw* specifies the drawable

*gc* specifies the GC

*x* specifies the upper left x coordinate

*y* specifies the upper left y coordinate

*w* specifies the rectangle width

*h* specifies the rectangle height

*ew* specifies the corner width

*eh* specifies the corner height

This function draws a rounded rectangle, where x, y, w, h are the dimensions of the overall rectangle, and ew and eh are the sizes of a bounding box that the corners are drawn inside of; ew should be no more than half of w, and eh should be no more than half of h. The current GC line attributes control all attributes of the line.

void XmuFillRoundedRectangle(*dpy*, *draw*, *gc*, *x*, *y*, *w*, *h*, *ew*, *eh*)
    Display **dpy*;
    Drawable *draw*;
    GC *gc*;
    int *x*, *y*, *w*, *h*, *ew*, *eh*;

*dpy* specifies the connection to the X server

*draw* specifies the drawable

*gc* specifies the GC

*x* specifies the upper left x coordinate

*y* specifies the upper left y coordinate

*w* specifies the rectangle width

*h* specifies the rectangle height

*ew* specifies the corner width

*eh* specifies the corner height

This function draws a filled rounded rectangle, where x, y, w, h are the dimensions of the overall rectangle, and ew and eh are the sizes of a bounding box that the corners are drawn inside of; ew should be no more than half of w, and eh should be no more than half of h. The current GC fill settings control all attributes of the fill contents.

XmuDrawLogo(*dpy*, *drawable*, *gcFore*, *gcBack*, *x*, *y*, *width*, *height*)
Display **dpy*;
Drawable *drawable*;
GC *gcFore*, *gcBack*;
int *x*, *y*;
unsigned int *width*, *height*;

*dpy* specifies the connection to the X server

*drawable* specifies the drawable

*gcFore* specifies the foreground GC

*gcBack* specifies the background GC

*x* specifies the upper left x coordinate

*y* specifies the upper left y coordinate

*width* specifies the logo width

*height* specifies the logo height

This function draws the ''official'' X Window System logo. The bounding box of the logo in the drawable is given by x, y, width, and height. The logo itself is filled using gcFore, and the rest of the rectangle is filled using gcBack.

Pixmap XmuCreateStippledPixmap(*screen*, *fore*, *back*, *depth*)
Screen **screen*;
Pixel *fore*, *back*;
unsigned int *depth*;

*screen* specifies the screen the pixmap is created on

*fore* specifies the foreground pixel value

*back* specifies the background pixel value

*depth* specifies the depth of the pixmap

This function creates a two pixel by one pixel stippled pixmap of specified depth on the specified screen. The pixmap is cached so that multiple requests share the same pixmap. The pixmap should be freed with **XmuReleaseStippledPixmap** to maintain correct reference counts.

void XmuReleaseStippledPixmap(*screen*, *pixmap*)
Screen **screen*;
Pixmap *pixmap*;

*screen* specifies the screen the pixmap was created on

*pixmap* specifies the pixmap to free

This function frees a pixmap created with **XmuCreateStippledPixmap**.

int XmuReadBitmapData(*fstream*, *width*, *height*, *datap*, *x_hot*, *y_hot*)
FILE **fstream*;
unsigned int **width*, **height*;
unsigned char ***datap*;
int **x_hot*, **y_hot*;

| | |
|---|---|
| *stream* | specifies the stream to read from |
| *width* | returns the width of the bitmap |
| *height* | returns the height of the bitmap |
| *datap* | returns the parsed bitmap data |
| *x_hot* | returns the x coordinate of the hotspot |
| *y_hot* | returns the y coordinate of the hotspot |

This function reads a standard bitmap file description from the specified stream, and returns the parsed data in a format suitable for passing to **XCreateBitmapFromData**. The return value of the function has the same interpretation as the return value for **XReadBitmapFile**.

int XmuReadBitmapDataFromFile(*filename*, *width*, *height*, *datap*, *x_hot*, *y_hot*)
char **filename*;
unsigned int **width*, **height*;
unsigned char ***datap*;
int **x_hot*, **y_hot*;

| | |
|---|---|
| *filename* | specifies the file to read from |
| *width* | returns the width of the bitmap |
| *height* | returns the height of the bitmap |
| *datap* | returns the parsed bitmap data |
| *x_hot* | returns the x coordinate of the hotspot |
| *y_hot* | returns the y coordinate of the hotspot |

This function reads a standard bitmap file description from the specified file, and returns the parsed data in a format suitable for passing to **XCreateBitmapFromData**. The return value of the function has the same interpretation as the return value for **XReadBitmapFile**.

Pixmap XmuLocateBitmapFile(*screen*, *name*, *srcname*, *srcnamelen*, *widthp*, *heightp*, *xhotp*, *yhotp*)
Screen **screen*;
char **name*;
char **srcname*;
int *srcnamelen*;
int **widthp*, **heightp*, **xhotp*, **yhotp*;

| | |
|---|---|
| *screen* | specifies the screen the pixmap is created on |
| *name* | specifies the file to read from |
| *srcname* | returns the full filename of the bitmap |
| *srcnamelen* | specifies the length of the srcname buffer |
| *width* | returns the width of the bitmap |
| *height* | returns the height of the bitmap |
| *xhotp* | returns the x coordinate of the hotspot |
| *yhotp* | returns the y coordinate of the hotspot |

This function reads a file in standard bitmap file format, using **XReadBitmapFile**, and returns the created bitmap. The filename may be absolute, or relative to the global resource named bitmapFilePath with class

BitmapFilePath. If the resource is not defined, the default value is the build symbol BITMAPDIR, which is typically "/usr/include/X11/bitmaps". If srcnamelen is greater than zero and srcname is not NULL, the null terminated filename will be copied into srcname. The size and hotspot of the bitmap are also returned.

Pixmap XmuCreatePixmapFromBitmap(*dpy*, *d*, *bitmap*, *width*, *height*, *depth*, *fore*, *back*)
Display **dpy*;
Drawable *d*;
Pixmap *bitmap*;
unsigned int *width*, *height*;
unsigned int *depth*;
unsigned long *fore*, *back*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *d* | specifies the screen the pixmap is created on |
| *bitmap* | specifies the bitmap source |
| *width* | specifies the width of the pixmap |
| *height* | specifies the height of the pixmap |
| *depth* | specifies the depth of the pixmap |
| *fore* | specifies the foreground pixel value |
| *back* | specifies the background pixel value |

This function creates a pixmap of the specified width, height, and depth, on the same screen as the specified drawable, and then performs an **XCopyPlane** from the specified bitmap to the pixmap, using the specified foreground and background pixel values. The created pixmap is returned.

### 8. Selection Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/StdSel.h>**.

Boolean XmuConvertStandardSelection(*w*, *time*, *selection*, *target*, *type*, *value*, *length*, *format*)
Widget *w*;
Time *time*;
Atom **selection*, **target*, **type*;
caddr_t **value*;
unsigned long **length*;
int **format*;

| | |
|---|---|
| *w* | specifies the widget which currently owns the selection |
| *time* | specifies the time at which the selection was established |
| *selection* | this argument is ignored |
| *target* | specifies the target type of the selection |
| *type* | returns the property type of the converted value |
| *value* | returns the converted value |
| *length* | returns the number of elements in the converted value |
| *format* | returns the size in bits of the elements |

This function converts the following standard selections: CLASS, CLIENT_WINDOW, DEC-NET_ADDRESS, HOSTNAME, IP_ADDRESS, NAME, OWNER_OS, TARGETS, TIMESTAMP, and USER. It returns **True** if the conversion was successful, else it returns **False.**

## 9. Type Converter Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/Converters.h>**.

void XmuCvtFunctionToCallback(*args*, *num_args*, *fromVal*, *toVal*)
    XrmValue **args*;
    Cardinal **num_args*;
    XrmValuePtr *fromVal*;
    XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | this argument is ignored |
| *num_args* | this argument is ignored |
| *fromVal* | the function to convert |
| *toVal* | the place to store the converted value |

This function converts a callback procedure to a callback list containing that procedure, with NULL closure data. To use this converter, include the following in your widget's ClassInitialize procedure:

XtAddConverter(XtRCallProc, XtRCallback, XmuCvtFunctionToCallback, NULL, 0);

void XmuCvtStringToBackingStore(*args*, *num_args*, *fromVal*, *toVal*)
    XrmValue **args*;
    Cardinal **num_args*;
    XrmValuePtr *fromVal*;
    XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | this argument is ignored |
| *num_args* | this argument must be a pointer to a Cardinal containing the value 0 |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |

This function converts a string to a backing-store integer as defined in **<X11/X.h>**. The string "notUseful" converts to **NotUseful**, "whenMapped" converts to **WhenMapped**, and "always" converts to **Always**. The string "default" converts to the value **Always+ WhenMapped+ NotUseful**. The case of the string does not matter. To use this converter, include the following in your widget's ClassInitialize procedure:

XtAddConverter(XtRString, XtRBackingStore, XmuCvtStringToBackingStore, NULL, 0);

void XmuCvtStringToBitmap(*args*, *num_args*, *fromVal*, *toVal*)
    XrmValuePtr *args*;
    Cardinal **num_args*;
    XrmValuePtr *fromVal*;
    XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | the sole argument specifies the Screen on which to create the bitmap |
| *num_args* | must be the value 1 |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |

This function creates a bitmap (a Pixmap of depth one) suitable for window manager icons. The string argument is the name of a file in standard bitmap file format. For the possible filename specifications, see **XmuLocateBitmapFile**. To use this converter, include the following in your widget's ClassInitialize procedure:

```
static XtConvertArgRec screenConvertArg[] = {
  {XtBaseOffset, (XtPointer)XtOffset(Widget, core.screen), sizeof(Screen *)}
};
```

```
XtAddConverter(XtRString, XtRBitmap, XmuCvtStringToBitmap,
        screenConvertArg, XtNumber(screenConvertArg));
```

Boolean XmuCvtStringToColorCursor(*dpy*, *args*, *num_args*, *fromVal*, *toVal*, *data*)
    Display * *dpy*;
    XrmValuePtr *args*;
    Cardinal **num_args*;
    XrmValuePtr *fromVal*;
    XrmValuePtr *toVal*;
    XtPointer * *data*;

| | |
|---|---|
| *dpy* | specifies the display to use for conversion warnings |
| *args* | specifies the required conversion arguments |
| *num_args* | specifies the number of required conversion arguments, which is 4 |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |
| *data* | this argument is ignored |

This function converts a string to a **Cursor** with the foreground and background pixels specified by the conversion arguments. The string can either be a standard cursor name formed by removing the ‘‘XC_’’ prefix from any of the cursor defines listed in Appendix B of the Xlib Manual, a font name and glyph index in decimal of the form "FONT fontname index [[font] index]", or a bitmap filename acceptable to **XmuLocateBitmapFile**. To use this converter, include the following in the widget ClassInitialize procedure:

```
static XtConvertArgRec colorCursorConvertArgs[] = {
  {XtWidgetBaseOffset, (XtPointer) XtOffsetOf(WidgetRec, core.screen),
   sizeof(Screen *)},
  {XtResourceString, (XtPointer) XtNpointerColor, sizeof(Pixel)},
  {XtResourceString, (XtPointer) XtNpointerColorBackground, sizeof(Pixel)},
  {XtWidgetBaseOffset, (XtPointer) XtOffsetOf(WidgetRec, core.colormap),
   sizeof(Colormap)}
};
XtSetTypeConverter(XtRString, XtRColorCursor, XmuCvtStringToColorCursor,
        colorCursorConvertArgs, XtNumber(colorCursorConvertArgs),
      XtCacheByDisplay, NULL); The widget must recognize XtNpointerColor and XtNpointerColor-
```

Background as resources, or specify other appropriate foreground and background resources. The widget’s Realize and SetValues methods must cause the converter to be invoked with the appropriate arguments when one of the foreground, background, or cursor resources has changed, or when the window is created, and must assign the cursor to the window of the widget.

void XmuCvtStringToCursor(*args*, *num_args*, *fromVal*, *toVal*)
    XrmValuePtr *args*;
    Cardinal **num_args*;
    XrmValuePtr *fromVal*;
    XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | specifies the required conversion argument, the screen |
| *num_args* | specifies the number of required conversion arguments, which is 1 |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |

This function converts a string to a **Cursor**. The string can either be a standard cursor name formed by removing the ‘‘XC_’’ prefix from any of the cursor defines listed in Appendix B of the Xlib Manual, a font name and glyph index in decimal of the form "FONT fontname index [[font] index]", or a bitmap filename

acceptable to **XmuLocateBitmapFile**. To use this converter, include the following in your widget's ClassInitialize procedure:

```
static XtConvertArgRec screenConvertArg[] = {
  {XtBaseOffset, (XtPointer)XtOffsetOf(WidgetRec, core.screen), sizeof(Screen *)}
};
XtAddConverter(XtRString, XtRCursor, XmuCvtStringToCursor,
               screenConvertArg, XtNumber(screenConvertArg));
```

void XmuCvtStringToGravity(*args*, *num_args*, *fromVal*, *toVal*)
XrmValuePtr **args*;
Cardinal **num_args*;
XrmValuePtr *fromVal*;
XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | this argument is ignored |
| *num_args* | this argument must be a pointer to a Cardinal containing the value 0 |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |

This function converts a string to an **XtGravity** enumeration value. The string "forget" and a NULL value convert to **ForgetGravity**, "NorthWestGravity" converts to **NorthWestGravity**, the strings "NorthGravity" and "top" convert to **NorthGravity**, "NorthEastGravity" converts to **NorthEastGravity**, the strings "West" and "left" convert to **WestGravity**, "CenterGravity" converts to **CenterGravity**, "EastGravity" and "right" convert to **EastGravity**, "SouthWestGravity" converts to **SouthWestGravity**, "SouthGravity" and "bottom" convert to **SouthGravity**, "SouthEastGravity" converts to **SouthEastGravity**, "StaticGravity" converts to **StaticGravity**, and "UnmapGravity" converts to **UnmapGravity**. The case of the string does not matter. To use this converter, include the following in your widget's class initialize procedure:

```
XtAddConverter(XtRString, XtRGravity, XmuCvtStringToGravity, NULL, 0);
```

void XmuCvtStringToJustify(*args*, *num_args*, *fromVal*, *toVal*)
XrmValuePtr **args*;
Cardinal **num_args*;
XrmValuePtr *fromVal*;
XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | this argument is ignored |
| *num_args* | this argument is ignored |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |

This function converts a string to an **XtJustify** enumeration value. The string "left" converts to **XtJustifyLeft**, "center" converts to **XtJustifyCenter**, and "right" converts to **XtJustifyRight**. The case of the string does not matter. To use this converter, include the following in your widget's ClassInitialize procedure:

```
XtAddConverter(XtRString, XtRJustify, XmuCvtStringToJustify, NULL, 0);
```

void XmuCvtStringToLong(*args*, *num_args*, *fromVal*, *toVal*)
XrmValuePtr *args*;
Cardinal *num_args*;
XrmValuePtr *fromVal*;
XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | this argument is ignored |
| *num_args* | this argument must be a pointer to a Cardinal containing 0 |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |

This function converts a string to an integer of type long. It parses the string using **sscanf** with a format of "%ld". To use this converter, include the following in your widget's ClassInitialize procedure:

XtAddConverter(XtRString, XtRLong, XmuCvtStringToLong, NULL, 0);

void XmuCvtStringToOrientation(*args*, *num_args*, *fromVal*, *toVal*)
    XrmValuePtr **args*;
    Cardinal **num_args*;
    XrmValuePtr *fromVal*;
    XrmValuePtr *toVal*;

| | |
|---|---|
| *args* | this argument is ignored |
| *num_args* | this argument is ignored |
| *fromVal* | specifies the string to convert |
| *toVal* | returns the converted value |

This function converts a string to an **XtOrientation** enumeration value. The string "horizontal" converts to **XtorientHorizontal** and "vertical" converts to **XtorientVertical**. The case of the string does not matter. To use this converter, include the following in your widget's ClassInitialize procedure:

XtAddConverter(XtRString, XtROrientation, XmuCvtStringToOrientation, NULL, 0);

Boolean XmuCvtStringToShapeStyle(*dpy*, *args*, *num_args*, *from*, *toVal*, data)
    Display **dpy*;
    XrmValue **args*;
    Cardinal **num_args*;
    XrmValue **from*;
    XrmValue **toVal*;
    XtPointer **data*;

| | |
|---|---|
| *dpy* | the display to use for conversion warnings |
| *args* | this argument is ignored |
| *num_args* | this argument is ignored |
| *fromVal* | the value to convert from |
| *toVal* | the place to store the converted value |
| *data* | this argument is ignored |

This function converts a string to an integer shape style. The string "rectangle" converts to **XmuShapeRectangle**, "oval" converts to **XmuShapeOval**, "ellipse" converts to **XmuShapeEllipse**, and "roundedRectangle" converts to **XmuShapeRoundedRectangle**. The case of the string does not matter. To use this converter, include the following in your widget's ClassInitialize procedure:

XtSetTypeConverter(XtRString, XtRShapeStyle, XmuCvtStringToShapeStyle,
        NULL, 0, XtCacheNone, NULL);

Boolean XmuReshapeWidget(w, *shape_style*, *corner_width*, *corner_height*)
    Widget *w*;
    int *shape_style*;
    int *corner_width*, *corner_height*;

*w* specifies the widget to reshape

*shape_style* specifies the new shape

*corner_width* specifies the width of the rounded rectangle corner

*corner_height* specified the height of the rounded rectangle corner

This function reshapes the specified widget, using the Shape extension, to a rectangle, oval, ellipse, or rounded rectangle, as specified by shape_style ( **XmuShapeRectangle**, **XmuShapeOval**, **XmuShapeEllipse**, and **XmuShapeRoundedRectangle**, respectively). The shape is bounded by the outside edges of the rectangular extents of the widget. If the shape is a rounded rectangle, corner_width and corner_height specify the size of the bounding box that the corners are drawn inside of (see **XmuFillRoundedRectangle**); otherwise, corner_width and corner_height are ignored. The origin of the widget within its parent remains unchanged.

void XmuCvtStringToWidget(*args*, *num_args*, *fromVal*, *toVal*)
XrmValuePtr *args*;
Cardinal **num_args*;
XrmValuePtr *fromVal*;
XrmValuePtr *toVal*;

*args* this sole argument is the parent Widget

*num_args* this argument must be 1

*fromVal* specifies the string to convert

*toVal* returns the converted value

This function converts a string to an immediate child widget of the parent widget passed as an argument. Note that this converter only works for child widgets that have already been created; there is no lazy evaluation. The string is first compared against the names of the normal and popup children, and if a match is found the corresponding child is returned. If no match is found, the string is compared against the classes of the normal and popup children, and if a match is found the corresponding child is returned. The case of the string is significant. To use this converter, include the following in your widget's ClassInitialize procedure:

```
static XtConvertArgRec parentCvtArg[] = {
  {XtBaseOffset, (XtPointer)XtOffset(Widget, core.parent), sizeof(Widget)},
};
XtAddConverter(XtRString, XtRWidget, XmuCvtStringToWidget,
               parentCvtArg, XtNumber(parentCvtArg));
```

Boolean XmuNewCvtStringToWidget(*dpy*, *args*, *num_args*, *fromVal*, *toVal*, *data*)
Display **dpy*;
XrmValue * *args*;
Cardinal **num_args*;
XrmValue * *fromVal*;
XrmValue * *toVal*;
XtPointer **data*;

*dpy* the display to use for conversion warnings

*args* this sole argument is the parent Widget

*num_args* this argument must be a pointer to a Cardinal containing the value 1

*fromVal* specifies the string to convert

*toVal* returns the converted value

*data* this argument is ignored

This converter is identical in functionality to XmuCvtStringToWidget, except that it is a new-style converter, allowing the specification of a cache type at the time of registration. Most widgets will not cache the conversion results, as the application may dynamically create and destroy widgets, which would cause cached values to become illegal. To use this converter, include the following in the widget's class initialize procedure:

```
static XtConvertArgRec parentCvtArg[] = {
  {XtWidgetBaseOffset, (XtPointer)XtOffsetOf(WidgetRec, core.parent),
   sizeof(Widget)}
};
XtSetTypeConverter(XtRString, XtRWidget, XmuNewCvtStringToWidget,
                   parentCvtArg, XtNumber(parentCvtArg), XtCacheNone, NULL);
```

## 10. Character Set Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/CharSet.h>**.

The functions in this section are **deprecated** because they don't work in most locales now supported by X11; the function **XmbLookupString** provides a better alternative.

void XmuCopyISOLatin1Lowered(*dst*, *src*)
    char **dst*, **src*;

*dst* returns the string copy

*src* specifies the string to copy

This function copies a null terminated string from src to dst (including the null), changing all Latin-1 uppercase letters to lowercase. The string is assumed to be encoded using ISO 8859-1.

void XmuCopyISOLatin1Uppered(*dst*, *src*)
    char **dst*, **src*;

*dst* returns the string copy

*src* specifies the string to copy

This function copies a null terminated string from src to dst (including the null), changing all Latin-1 lowercase letters to uppercase. The string is assumed to be encoded using ISO 8859-1.

int XmuCompareISOLatin1(*first*, *second*)
    char **first*, **second*;

*dst* specifies a string to compare

*src* specifies a string to compare

This function compares two null terminated Latin-1 strings, ignoring case differences, and returns an integer greater than, equal to, or less than 0, according to whether first is lexicographically greater than, equal to, or less than second. The two strings are assumed to be encoded using ISO 8859-1.

int XmuLookupLatin1(*event*, *buffer*, *nbytes*, *keysym*, *status*)
    XKeyEvent **event*;
    char **buffer*;
    int *nbytes*;
    KeySym **keysym*;
    XComposeStatus **status*;

*event* specifies the key event

*buffer* returns the translated characters

*nbytes* specifies the length of the buffer

*keysym* returns the computed KeySym, or None

*status* specifies or returns the compose state

This function is identical to **XLookupString**, and exists only for naming symmetry with other functions.

int XmuLookupLatin2(*event*, *buffer*, *nbytes*, *keysym*, *status*)
XKeyEvent **event*;
char **buffer*;
int *nbytes*;
KeySym **keysym*;
XComposeStatus **status*;

*event* specifies the key event

*buffer* returns the translated characters

*nbytes* specifies the length of the buffer

*keysym* returns the computed KeySym, or None

*status* specifies or returns the compose state

This function is similar to **XLookupString**, except that it maps a key event to an Latin-2 (ISO 8859-2) string, or to an ASCII control string.

int XmuLookupLatin3(*event*, *buffer*, *nbytes*, *keysym*, *status*)
XKeyEvent **event*;
char **buffer*;
int *nbytes*;
KeySym **keysym*;
XComposeStatus **status*;

*event* specifies the key event

*buffer* returns the translated characters

*nbytes* specifies the length of the buffer

*keysym* returns the computed KeySym, or None

*status* specifies or returns the compose state

This function is similar to **XLookupString**, except that it maps a key event to an Latin-3 (ISO 8859-3) string, or to an ASCII control string.

int XmuLookupLatin4(*event*, *buffer*, *nbytes*, *keysym*, *status*)
XKeyEvent **event*;
char **buffer*;
int *nbytes*;
KeySym **keysym*;
XComposeStatus **status*;

*event* specifies the key event

*buffer* returns the translated characters

*nbytes* specifies the length of the buffer

*keysym* returns the computed KeySym, or None

*status* specifies or returns the compose state

This function is similar to **XLookupString**, except that it maps a key event to an Latin-4 (ISO 8859-4) string, or to an ASCII control string.

int XmuLookupKana(*event*, *buffer*, *nbytes*, *keysym*, *status*)
    XKeyEvent **event*;
    char **buffer*;
    int *nbytes*;
    KeySym **keysym*;
    XComposeStatus **status*;

| | |
|---|---|
| *event* | specifies the key event |
| *buffer* | returns the translated characters |
| *nbytes* | specifies the length of the buffer |
| *keysym* | returns the computed KeySym, or None |
| *status* | specifies or returns the compose state |

This function is similar to **XLookupString**, except that it maps a key event to a string in an encoding consisting of Latin-1 (ISO 8859-1) and ASCII control in the Graphics Left half (values 0 to 127), and Katakana in the Graphics Right half (values 128 to 255), using the values from JIS X201-1976.

int XmuLookupJISX0201(*event*, *buffer*, *nbytes*, *keysym*, *status*)
    XKeyEvent **event*;
    char **buffer*;
    int *nbytes*;
    KeySym **keysym*;
    XComposeStatus **status*;

| | |
|---|---|
| *event* | specifies the key event |
| *buffer* | returns the translated characters |
| *nbytes* | specifies the length of the buffer |
| *keysym* | returns the computed KeySym, or None |
| *status* | specifies or returns the compose state |

This function is similar to **XLookupString**, except that it maps a key event to a string in the JIS X0201-1976 encoding, including ASCII control.

int XmuLookupArabic(*event*, *buffer*, *nbytes*, *keysym*, *status*)
    XKeyEvent **event*;
    char **buffer*;
    int *nbytes*;
    KeySym **keysym*;
    XComposeStatus **status*;

| | |
|---|---|
| *event* | specifies the key event |
| *buffer* | returns the translated characters |
| *nbytes* | specifies the length of the buffer |
| *keysym* | returns the computed KeySym, or None |
| *status* | specifies or returns the compose state |

This function is similar to **XLookupString**, except that it maps a key event to a Latin/Arabic (ISO 8859-6) string, or to an ASCII control string.

int XmuLookupCyrillic(*event*, *buffer*, *nbytes*, *keysym*, *status*)
XKeyEvent **event*;
char **buffer*;
int *nbytes*;
KeySym **keysym*;
XComposeStatus **status*;

| | |
|---|---|
| *event* | specifies the key event |
| *buffer* | returns the translated characters |
| *nbytes* | specifies the length of the buffer |
| *keysym* | returns the computed KeySym, or None |
| *status* | specifies or returns the compose state |

This function is similar to **XLookupString**, except that it maps a key event to a Latin/Cyrillic (ISO 8859-5) string, or to an ASCII control string.

int XmuLookupGreek(*event*, *buffer*, *nbytes*, *keysym*, *status*)
XKeyEvent **event*;
char **buffer*;
int *nbytes*;
KeySym **keysym*;
XComposeStatus **status*;

| | |
|---|---|
| *event* | specifies the key event |
| *buffer* | returns the translated characters |
| *nbytes* | specifies the length of the buffer |
| *keysym* | returns the computed KeySym, or None |
| *status* | specifies or returns the compose state |

This function is similar to **XLookupString**, except that it maps a key event to a Latin/Greek (ISO 8859-7) string, or to an ASCII control string.

int XmuLookupHebrew(*event*, *buffer*, *nbytes*, *keysym*, *status*)
XKeyEvent **event*;
char **buffer*;
int *nbytes*;
KeySym **keysym*;
XComposeStatus **status*;

| | |
|---|---|
| *event* | specifies the key event |
| *buffer* | returns the translated characters |
| *nbytes* | specifies the length of the buffer |
| *keysym* | returns the computed KeySym, or None |
| *status* | specifies or returns the compose state |

This function is similar to **XLookupString**, except that it maps a key event to a Latin/Hebrew (ISO 8859-8) string, or to an ASCII control string.

int XmuLookupAPL(*event*, *buffer*, *nbytes*, *keysym*, *status*)
    XKeyEvent **event*;
    char **buffer*;
    int *nbytes*;
    KeySym **keysym*;
    XComposeStatus **status*;

| | |
|---|---|
| *event* | specifies the key event |
| *buffer* | returns the translated characters |
| *nbytes* | specifies the length of the buffer |
| *keysym* | returns the computed KeySym, or None |
| *status* | specifies or returns the compose state |

This function is similar to **XLookupString**, except that it maps a key event to an APL string.

### 11. Compound Text Functions

The functions defined in this section are for parsing Compound Text strings, decomposing them into individual segments. Definitions needed to use these routines are in the include file **<X11/Xmu/Xct.h>**.

The functions in this section are **deprecated** because they shift the burden for recently introduced locale encodings to the application. The use of the **UTF8_STRING** text encoding provides a better alternative.

A Compound Text string is represented as the following type:

```
typedef unsigned char *XctString;
```

XctData XctCreate(*string*, *length*, *flags*)
    XctString *string*;
    int *length*;
    XctFlags *flags*;

| | |
|---|---|
| *string* | the Compound Text string |
| *length* | the number of bytes in string |
| *flags* | parsing control flags |

This function creates an **XctData** structure for parsing a Compound Text string. The string need not be null terminated. The following flags are defined to control parsing of the string:

**XctSingleSetSegments** -- This means that returned segments should contain characters from only one set (C0, C1, GL, GR). When this is requested, **XctSegment** is never returned by **XctNextItem**, instead **XctC0Segment**, **XctC1Segment**, **XctGlSegment**, and **XctGRSegment** are returned. C0 and C1 segments are always returned as singleton characters.

**XctProvideExtensions** -- This means that if the Compound Text string is from a higher version than this code is implemented to, then syntactically correct but unknown control sequences should be returned as **XctExtension** items by **XctNextItem**. If this flag is not set, and the Compound Text string version indicates that extensions cannot be ignored, then each unknown control sequence will be reported as an **XctError**.

**XctAcceptC0Extensions** -- This means that if the Compound Text string is from a higher version than this code is implemented to, then unknown C0 characters should be treated as if they were legal, and returned as C0 characters (regardless of how **XctProvideExtensions** is set) by **XctNextItem**. If this flag is not set, then all unknown C0 characters are treated according to **XctProvideExtensions**.

**XctAcceptC1Extensions** -- This means that if the Compound Text string is from a higher version than this code is implemented to, then unknown C1 characters should be treated as if they were legal, and returned as C1 characters (regardless of how **XctProvideExtensions** is set) by **XctNextItem**. If this flag is not set, then all unknown C1 characters are treated according to **XctProvideExtensions**.

**XctHideDirection** -- This means that horizontal direction changes should be reported as **XctHorizontal** items by **XctNextItem**. then direction changes are not returned as items, but the current direction is still maintained and reported for other items. The current direction is given as an enumeration, with the values **XctUnspecified**, **XctLeftToRight**, and **XctRightToLeft**.

**XctFreeString** -- This means that **XctFree** should free the Compound Text string that is passed to **XctCreate**. If this flag is not set, the string is not freed.

**XctShiftMultiGRToGL** -- This means that **XctNextItem** should translate GR segments on-the-fly into GL segments for the GR sets: GB2312.1980-1, JISX0208.1983-1, and KSC5601.1987-1.

void XctReset(*data*)
XctData *data*;

*data* specifies the Compound Text structure

This function resets the **XctData** structure to reparse the Compound Text string from the beginning.

XctResult XctNextItem(*data*)
XctData *data*;

*data* specifies the Compound Text structure

This function parses the next ''item'' from the Compound Text string. The return value indicates what kind of item is returned. The item itself, it's length, and the current contextual state, are reported as components of the **XctData** structure. **XctResult** is an enumeration, with the following values:

**XctSegment** -- the item contains some mixture of C0, GL, GR, and C1 characters.

**XctC0Segment** -- the item contains only C0 characters.

**XctGLSegment** -- the item contains only GL characters.

**XctC1Segment** -- the item contains only C1 characters.

**XctGRSegment** -- the item contains only GR characters.

**XctExtendedSegment** -- the item contains an extended segment.

**XctExtension** -- the item is an unknown extension control sequence.

**XctHorizontal** -- the item indicates a change in horizontal direction or depth. The new direction and depth are recorded in the **XctData** structure.

**XctEndOfText** -- The end of the Compound Text string has been reached.

**XctError** -- the string contains a syntactic or semantic error; no further parsing should be performed.

The following state values are stored in the **XctData** structure:

```
XctString item;                 /* the action item */
int item_length;                /* the length of item in bytes */
int char_size;                  /* the number of bytes per character in
                                 * item, with zero meaning variable */
char *encoding;                 /* the XLFD encoding name for item */
XctHDirection horizontal;       /* the direction of item */
int horz_depth;                 /* the current direction nesting depth */
char *GL;                       /* the "{I} F" string for the current GL */
char *GL_encoding;              /* the XLFD encoding name for the current GL */
int GL_set_size;                /* 94 or 96 */
int GL_char_size;               /* the number of bytes per GL character */
char *GR;                       /* the "{I} F" string for the current GR */
char *GR_encoding;              /* the XLFD encoding name the for current GR */
int GR_set_size;                /* 94 or 96 */
int GR_char_size;               /* the number of bytes per GR character */
char *GLGR_encoding;            /* the XLFD encoding name for the current
```

```
                                        * GL+GR, if known */
```

void XctFree(*data*)
    XctData *data*;

*data* specifies the Compound Text structure

This function frees all data associated with the **XctData** structure.

### 12. CloseDisplay Hook Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/CloseHook.h>**.

CloseHook XmuAddCloseDisplayHook(*dpy*, *func*, *arg*)
    Display **dpy*;
    int (**func*)();
    caddr_t *arg*;

*dpy* specifies the connection to the X server

*func* specifies the function to call at display close

*arg* specifies arbitrary data to pass to func

This function adds a callback for the given display. When the display is closed, the given function will be called with the given display and argument as:

    (*func)(dpy, arg)

The function is declared to return an int even though the value is ignored, because some compilers have problems with functions returning void.

This routine returns NULL if it was unable to add the callback, otherwise it returns an opaque handle that can be used to remove or lookup the callback.

Bool XmuRemoveCloseDisplayHook(*dpy*, *handle*, *func*, *arg*)
    Display **dpy*;
    CloseHook *handle*;
    int (**func*)();
    caddr_t *arg*;

*dpy* specifies the connection to the X server

*handle* specifies the callback by id, or NULL

*func* specifies the callback by function

*arg* specifies the function data to match

This function deletes a callback that has been added with **XmuAddCloseDisplayHook**. If handle is not NULL, it specifies the callback to remove, and the func and arg parameters are ignored. If handle is NULL, the first callback found to match the specified func and arg will be removed. Returns **True** if a callback was removed, else returns **False**.

Bool XmuLookupCloseDisplayHook(*dpy*, *handle*, *func*, *arg*)
    Display **dpy*;
    CloseHook *handle*;
    int (**func*)();
    caddr_t *arg*;

*dpy* specifies the connection to the X server

*handle* specifies the callback by id, or NULL

*func* specifies the callback by function

*arg* specifies the function data to match

This function determines if a callback is installed. If handle is not NULL, it specifies the callback to look for, and the func and arg parameters are ignored. If handle is NULL, the function will look for any callback for the specified func and arg. Returns **True** if a matching callback exists, else returns **False**.

### 13. Display Queue Functions

To use the functions and types defined in this section, you should include the header file **<X11/Xmu/DisplayQue.h>**. It defines the following types:

```
typedef struct _XmuDisplayQueueEntry {
        struct _XmuDisplayQueueEntry *prev, *next;
        Display *display;
        CloseHook closehook;
        caddr_t data;
} XmuDisplayQueueEntry;

typedef struct _XmuDisplayQueue {
        int nentries;
        XmuDisplayQueueEntry *head, *tail;
        int (*closefunc)();
        int (*freefunc)();
        caddr_t data;
} XmuDisplayQueue;
```

XmuDisplayQueue *XmuDQCreate(*closefunc*, *freefunc*, *data*)
    int (**closefunc*)();
    int (**freefunc*)();
    caddr_t *data*;

*closefunc* specifies the close function

*freefunc* specifies the free function

*data* specifies private data for the functions

This function creates and returns an empty **XmuDisplayQueue** (which is really just a set of displays, but is called a queue for historical reasons). The queue is initially empty, but displays can be added using **XmuAddDisplay**. The data value is simply stored in the queue for use by the closefunc and freefunc callbacks. Whenever a display in the queue is closed using **XCloseDisplay**, the closefunc (if non-NULL) is called with the queue and the display's **XmuDisplayQueueEntry** as follows:

```
(*closefunc)(queue, entry)
```

The freeproc (if non-NULL) is called whenever the last display in the queue is closed, as follows:

```
(*freefunc)(queue)
```

The application is responsible for actually freeing the queue, by calling **XmuDQDestroy**.

XmuDisplayQueueEntry *XmuDQAddDisplay(*q*, *dpy*, *data*)
    XmuDisplayQueue **q*;
    Display **dpy*;
    caddr_t *data*;

*q* specifies the queue

*dpy* specifies the display to add

*data* specifies private data for the free function

This function adds the specified display to the queue. If successful, the queue entry is returned, otherwise NULL is returned. The data value is simply stored in the queue entry for use by the queue's freefunc callback. This function does not attempt to prevent duplicate entries in the queue; the caller should use **XmuDQLookupDisplay** to determine if a display has already been added to a queue.

XmuDisplayQueueEntry *XmuDQLookupDisplay(*q*, *dpy*)
    XmuDisplayQueue **q*;
    Display **dpy*;

*q* specifies the queue

*dpy* specifies the display to lookup

This function returns the queue entry for the specified display, or NULL if the display is not in the queue.

XmuDQNDisplays(*q*)

This macro returns the number of displays in the specified queue.

Bool XmuDQRemoveDisplay(*q*, *dpy*)
    XmuDisplayQueue **q*;
    Display **dpy*;

*q* specifies the queue

*dpy* specifies the display to remove

This function removes the specified display from the specified queue. No callbacks are performed. If the display is not found in the queue, **False** is returned, otherwise **True** is returned.

Bool XmuDQDestroy(*q*, *docallbacks*)
    XmuDisplayQueue **q*;
    Bool *docallbacks*;

*q* specifies the queue to destroy

*docallbacks* specifies whether close functions should be called

This function releases all memory associated with the specified queue. If docallbacks is **True**, then the queue's closefunc callback (if non-NULL) is first called for each display in the queue, even though **XCloseDisplay** is not called on the display.

### 14. Toolkit Convenience Functions

To use the functions defined in this section, you should include the header file **<X11/Xmu/Initer.h>**.

void XmuAddInitializer(*func*, *data*)
    void (**func*)();
    caddr_t *data*;

*func* specifies the procedure to register

*data* specifies private data for the procedure

This function registers a procedure, to be invoked the first time **XmuCallInitializers** is called on a given application context. The procedure is called with the application context and the specified data:

```
(*func)(app_con, data)
```

void XmuCallInitializers(*app_con*)
    XtAppContext *app_con*;

*app_con*  specifies the application context to initialize

This function calls each of the procedures that have been registered with **XmuAddInitializer**, if this is the first time the application context has been passed to **XmuCallInitializers**. Otherwise, this function does nothing.

### 15. Standard Colormap Functions

To use the functions defined in this section, you should include the header file <**X11/Xmu/StdCmap.h**>.

Status XmuAllStandardColormaps(*dpy*)
    Display **dpy*;

*dpy*  specifies the connection to the X server

To create all of the appropriate standard colormaps for every visual of every screen on a given display, use **XmuAllStandardColormaps**.

This function defines and retains as permanent resources all standard colormaps which are meaningful for the visuals of each screen of the display. It returns 0 on failure, non-zero on success. If the property of any standard colormap is already defined, this function will redefine it.

This function is intended to be used by window managers or a special client at the start of a session.

The standard colormaps of a screen are defined by properties associated with the screen's root window. The property names of standard colormaps are predefined, and each property name except RGB_DEFAULT_MAP may describe at most one colormap.

The standard colormaps are: RGB_BEST_MAP, RGB_RED_MAP, RGB_GREEN_MAP, RGB_BLUE_MAP, RGB_DEFAULT_MAP, and RGB_GRAY_MAP. Therefore a screen may have at most 6 standard colormap properties defined.

A standard colormap is associated with a particular visual of the screen. A screen may have multiple visuals defined, including visuals of the same class at different depths. Note that a visual id might be repeated for more than one depth, so the visual id and the depth of a visual identify the visual. The characteristics of the visual will determine which standard colormaps are meaningful under that visual, and will determine how the standard colormap is defined. Because a standard colormap is associated with a specific visual, there must be a method of determining which visuals take precedence in defining standard colormaps.

The method used here is: for the visual of greatest depth, define all standard colormaps meaningful to that visual class, according to this order of (descending) precedence: **DirectColor**; **PseudoColor**; **TrueColor** and **GrayScale**; and finally **StaticColor** and **StaticGray**.

This function allows success, on a per screen basis. For example, if a map on screen 1 fails, the maps on screen 0, created earlier, will remain. However, none on screen 1 will remain. If a map on screen 0 fails, none will remain.

See **XmuVisualStandardColormaps** for which standard colormaps are meaningful under these classes of visuals.

Status XmuVisualStandardColormaps(*dpy*, *screen*, *visualid*, *depth*, *replace*, *retain*)
    Display **dpy*;
    int *screen*;
    VisualID *visualid*;
    unsigned int *depth*;
    Bool *replace*;
    Bool *retain*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *screen* | specifies the screen of the display |
| *visualid* | specifies the visual type |
| *depth* | specifies the visual depth |
| *replace* | specifies whether or not to replace |
| *retain* | specifies whether or not to retain |

To create all of the appropriate standard colormaps for a given visual on a given screen, use **XmuVisualStandardColormaps**.

This function defines all appropriate standard colormap properties for the given visual. If replace is **True**, any previous definition will be removed. If retain is **True**, new properties will be retained for the duration of the server session. This function returns 0 on failure, non-zero on success. On failure, no new properties will be defined, but old ones may have been removed if replace was True.

Not all standard colormaps are meaningful to all visual classes. This routine will check and define the following properties for the following classes, provided that the size of the colormap is not too small. For **DirectColor** and **PseudoColor**: RGB_DEFAULT_MAP, RGB_BEST_MAP, RGB_RED_MAP, RGB_GREEN_MAP, RGB_BLUE_MAP, and RGB_GRAY_MAP. For **TrueColor** and **StaticColor**: RGB_BEST_MAP. For **GrayScale** and **StaticGray**: RGB_GRAY_MAP.

Status XmuLookupStandardColormap(*dpy*, *screen*, *visualid*, *depth*, *property*, *replace*, *retain*)
Display **dpy*;
int *screen*;
VisualID *visualid*;
unsigned int *depth*;
Atom *property*;
Bool *replace*;
Bool *retain*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *screen* | specifies the screen of the display |
| *visualid* | specifies the visual type |
| *depth* | specifies the visual depth |
| *property* | specifies the standard colormap property |
| *replace* | specifies whether or not to replace |
| *retain* | specifies whether or not to retain |

To create a standard colormap if one does not currently exist, or replace the currently existing standard colormap, use **XmuLookupStandardColormap**.

Given a screen, a visual, and a property, this function will determine the best allocation for the property under the specified visual, and determine the whether to create a new colormap or to use the default colormap of the screen.

If replace is True, any previous definition of the property will be replaced. If retain is True, the property and the colormap will be made permanent for the duration of the server session. However, pre-existing property definitions which are not replaced cannot be made permanent by a call to this function; a request to retain resources pertains to newly created resources.

This function returns 0 on failure, non-zero on success. A request to create a standard colormap upon a visual which cannot support such a map is considered a failure. An example of this would be requesting any standard colormap property on a monochrome visual, or, requesting an RGB_BEST_MAP on a display whose colormap size is 16.

Status XmuGetColormapAllocation(*vinfo*, *property*, *red_max*, *green_max*, *blue_max*)
XVisualInfo **vinfo*;
Atom *property*;
unsigned long **red_max*, **green_max*, **blue_max*;

| | |
|---|---|
| *vinfo* | specifies visual information for a chosen visual |
| *property* | specifies one of the standard colormap property names |
| *red_max* | returns maximum red value |
| *green_max* | returns maximum green value |
| *blue_max* | returns maximum blue value |

To determine the best allocation of reds, greens, and blues in a standard colormap, use **XmuGetColormapAllocation**.

**XmuGetColormapAllocation** returns 0 on failure, non-zero on success. It is assumed that the visual is appropriate for the colormap property.

XStandardColormap *XmuStandardColormap(*dpy*, *screen*, *visualid*, *depth*, *property*,
*cmap*, *red_max*, *green_max*, *blue_max*)
Display *dpy*;
int *screen*;
VisualID *visualid*;
unsigned int *depth*;
Atom *property*;
Colormap *cmap*;
unsigned long *red_max*, *green_max*, *blue_max*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *screen* | specifies the screen of the display |
| *visualid* | specifies the visual type |
| *depth* | specifies the visual depth |
| *property* | specifies the standard colormap property |
| *cmap* | specifies the colormap ID, or None |
| *red_max* | specifies the red allocation |
| *green_max* | specifies the green allocation |
| *blue_max* | specifies the blue allocation |

To create any one standard colormap, use **XmuStandardColormap**.

This function creates a standard colormap for the given screen, visualid, and visual depth, with the given red, green, and blue maximum values, with the given standard property name. Upon success, it returns a pointer to an **XStandardColormap** structure which describes the newly created colormap. Upon failure, it returns NULL. If cmap is the default colormap of the screen, the standard colormap will be defined on the default colormap; otherwise a new colormap is created.

Resources created by this function are not made permanent; that is the caller's responsibility.

Status XmuCreateColormap(*dpy*, *colormap*)
Display **dpy*;
XStandardColormap **colormap*;

| | |
|---|---|
| *dpy* | specifies the connection under which the map is created |
| *colormap* | specifies the map to be created |

To create any one colormap which is described by an **XStandardColormap** structure, use **XmuCreateColormap**.

This function returns 0 on failure, and non-zero on success. The base_pixel of the colormap is set on success. Resources created by this function are not made permanent. No argument error checking is provided; use at your own risk.

All colormaps are created with read-only allocations, with the exception of read-only allocations of colors which fail to return the expected pixel value, and these are individually defined as read/write allocations. This is done so that all the cells defined in the colormap are contiguous, for use in image processing. This typically happens with White and Black in the default map.

Colormaps of static visuals are considered to be successfully created if the map of the static visual matches the definition given in the standard colormap structure.

void XmuDeleteStandardColormap(*dpy*, *screen*, *property*)
    Display **dpy*;
    int *screen*;
    Atom *property*;

| | |
|---|---|
| *dpy* | specifies the connection to the X server |
| *screen* | specifies the screen of the display |
| *property* | specifies the standard colormap property |

To remove any standard colormap property, use **XmuDeleteStandardColormap**. This function will remove the specified property from the specified screen, releasing any resources used by the colormap(s) of the property, if possible.

### 16. Widget Description Functions

The functions defined in this section are for building a description of the structure of and resources associated with a hierarchy of widget classes. This package is typically used by applications that wish to manipulate the widget set itself.

The definitions needed to use these interfaces are in the header file **<X11/Xmu/WidgetNode.h>**. The following function must be called before any of the others described below:

void XmuWnInitializeNodes(*node_array*, *num_nodes*)
    XmuWidgetNode **node_array*;
    int *num_nodes*;

| | |
|---|---|
| *node_array* | specifies a list of widget classes, in alphabetical order |
| *num_nodes* | specfies the number of widget classes in the node array |

To determine the resources provided by a widget class or classes, use

void XmuWnFetchResources(*node*, *toplevel*, *top_node*)
    XmuWidgetNode **node*;
    Widget *toplevel*;
    XmuWidgetNode **top_node*;

| | |
|---|---|
| *node* | specifies the widget class for which resources should be obtained. |
| *toplevel* | specifies the widget that should be used for creating an instance of *node* from which resources are extracted. This is typically the value returned by **XtAppInitialize**. |
| *top_node* | specifies the ancestor of *node* that should be treated as the root of the widget inheritance tree (used in determining which ancestor contributed which resources). |

Each widget class inherits the resources of its parent. To count the number of resources contributed by a particular widget class, use:

int XmuWnCountOwnedResources(*node*, *owner_node*, *constraints*)
    XmuWidgetNode **node*;
    XmuWidgetNode **owner_node*;
    Bool *constraints*;

*node* specifies the widget class whose resources are being examined.

*owner_node* specifies the widget class of the ancestor of *node* whose contributions are being counted.

*constraints* specifies whether or not to count constraint resources or normal resources.

This routine returns the number of resources contributed (or ‘‘owned’’) by the specified widget class.

XmuWidgetNode *XmuWnNameToNode(*node_list*, *num_nodes*, *name*)
    XmuWidgetNode **node_list*;
    int *num_nodes*;
    char **name*;

*node_list* specifies a list of widget nodes

*num_nodes* specifies the number of nodes in the list

*name* specifies the name of the widget class in the node list to search for

This function returns the WidgetNode in the list that matches the given widget name or widget class name. If no match is found, it returns NULL.

### 17. Participation in the Editres Protocol

To participate in the editres protocol, applications which are not based on the Athena widget set should include the header file **<X11/Xmu/Editres.h>**.

To participate in the editres protocol, Xt applications which do not rely on the Athena widget set should register the editres protocol handler on each shell widget in the application, specifying an event mask of 0, nonmaskable events, and client data as NULL:

XtAddEventHandler(shell, (EventMask) 0, True, _XEditResCheckMessages, NULL);