Section 1 - Motif Functions and Macros

This page describes the format and contents of each reference page in Section 1, which covers the Motif functions and macros.

Name

Function – a brief description of the function.

Synopsis

This section shows the signature of the function: the names and types of the arguments, and the type of the return value. If header file other than <*Xm/Xm.h*> is needed to declare the function, it is shown in this section as well.

Inputs

This subsection describes each of the function arguments that pass information to the function.

Outputs

This subsection describes any of the function arguments that are used to return information from the function. These arguments are always of some pointer type, so you should use the C address-of operator (&) to pass the address of the variable in which the function will store the return value. The names of these arguments are sometimes suffixed with *_return* to indicate that values are returned in them. Some arguments both supply and return a value; they will be listed in this section and in the "Inputs" section above. Finally, note that because the list of function arguments is broken into "Input" and "Output" sections, they do not always appear in the same order that they are passed to the function. See the function signature for the actual calling order.

Returns

This subsection explains the return value of the function, if any.

Availability

This section appears for functions that were added in Motif 2.0 and later, and also for functions that are now superseded by other, preferred, functions.

Description

This section explains what the function does and describes its arguments and return value. If you've used the function before and are just looking for a refresher, this section and the synopsis above should be all you need.

Usage

This section appears for most functions and provides less formal information about the function: when and how you might want to use it, things to watch out for, and related functions that you might want to consider.

Example

This section appears for some of the most commonly used Motif functions, and provides an example of their use.

Structures

This section shows the definition of any structures, enumerated types, typedefs, or symbolic constants used by the function.

Procedures

This section shows the syntax of any prototype procedures used by the function.

See Also

This section refers you to related functions, widget classes, and clients. The numbers in parentheses following each reference refer to the sections of this book in which they are found.

Name

XmActivateProtocol – activate a protocol.

Synopsis

#include <Xm/Protocols.h>

void XmActivateProtocol (Widget *shell*, Atom *property,* Atom *protocol*)

Inputs

shell - Specifies the widget associated with the protocol property.
property - Specifies the property that holds the protocol data.
protocol - Specifies the protocol atom.

Description

`XmActivateProtocol`() activates the specified protocol. If the shell is realized, `XmActivateProtocol`() updates its protocol handlers and the specified property. If the protocol is active, the protocol atom is stored in property; if the protocol is inactive, the protocol atom is not stored in property.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. `XmActivateProtocol`() makes the shell able to respond to ClientMessage events that contain the specified protocol. Before you can activate a protocol, the protocol must be added to the shell with `XmAddProtocols`(). Protocols are automatically activated when they are added. The inverse routine is `XmDeactivateProtocol`().

See Also

`XmActivateWMProtocol`(1), `XmAddProtocols`(1) `XmDeactivateProtocol`(1), `XmInternAtom`(1), `VendorShell`(2).

Name

XmActivateWMProtocol – activate the XA_WM_PROTOCOLS protocol.

Synopsis

#include <Xm/Protocols.h>

void XmActivateWMProtocol (Widget *shell*, Atom *protocol*)

Inputs

shell - Specifies the widget associated with the protocol property.
protocol - Specifies the protocol atom.

Description

`XmActivateWMProtocol`() is a convenience routine that calls `XmActivateProtocol`() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. Before you can activate the protocols, they must be added to the shell with `XmAddProtocols`() or `XmAddWMProtocols`(). Protocols are automatically activated when they are added. The inverse routine is `XmDeactivateWMProtocol`().

See Also

`XmActivateProtocol`(1), `XmAddProtocols`(1), `XmAddWMProtocols`(1), `XmDeactivateWMProtocol`(1), `XmInternAtom`(1), `VendorShell`(2).

Name

XmAddProtocolCallback – add client callbacks to a protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmAddProtocolCallback (  Widget          shell,
                              Atom            property,
                              Atom            protocol,
                              XtCallbackProc  callback,
                              XtPointer       closure)
```

Inputs

shell - Specifies the widget associated with the protocol property.
property - Specifies the property that holds the protocol data.
protocol - Specifies the protocol atom.
callback - Specifies the procedure to invoke when the protocol message is received.
closure - Specifies any client data that is passed to the callback.

Description

XmAddProtocolCallback() adds client callbacks to a protocol. The routine verifies that the protocol is registered, and if it is not, it calls XmAddProtocols(). XmAddProtocolCallback() adds the callback to the internal list of callbacks, so that it is called when the corresponding client message is received.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. To communicate using a protocol, a client sends a ClientMessage event containing a property and protocol, and the receiving client responds by calling the associated protocol callback routine. XmAddProtocolCallback() allows you to register these callback routines.

See Also

`XmAddProtocols`(1), `XmAddWMProtocolCallback`(1), `XmInternAtom`(1), `VendorShell`(2).

Name

XmAddProtocols – add protocols to the protocol manager.

Synopsis

#include <Xm/Protocols.h>

void XmAddProtocols (Widget *shell*, Atom *property*, Atom **protocols*, Cardinal *num_protocols*)

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocols	Specifies a list of protocol atoms.
num_protocols	Specifies the number of atoms in protocols.

Description

`XmAddProtocols`() registers a list of protocols to be stored in the specified property of the specified shell widget. The routine adds the protocols to the protocol manager and allocates the internal tables that are needed for the protocol.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. `XmAddProtocols`() allows you to add protocols that can be understood by your application. The inverse routine is `XmRemoveProtocols`(). To communicate using a protocol, a client sends a ClientMessage event containing a property and protocol, and the receiving client responds by calling the associated protocol callback routine. Use `XmAddProtocolCallback`() to add a callback function to be executed when a client message event containing the specified protocol atom is received.

See Also

`XmAddProtocolCallback`(1), `XmAddWMProtocols`(1), `XmInternAtom`(1), `XmRemoveProtocols`(1), `VendorShell`(2).

Name

XmAddTabGroup – add a widget to a list of tab groups.

Synopsis

void XmAddTabGroup (Widget *tab_group*)

Inputs

tab_group Specifies the widget to be added.

Availability

In Motif 1.1, `XmAddTabGroup`() is obsolete. It has been superseded by setting XmNnavigationType to XmEXCLUSIVE_TAB_GROUP.

Description

`XmAddTabGroup`() makes the specified widget a separate tab group. This routine is retained for compatibility with Motif 1.0 and should not be used in newer applications. If traversal behavior needs to be changed, this should be done directly by setting the XmNnavigationType resource, which is defined by Manager and Primitive.

Usage

A tab group is a group of widgets that can be traversed using the keyboard rather than the mouse. Users move from widget to widget within a single tab group by pressing the arrow keys. Users move between different tab groups by pressing the Tab or Shift-Tab keys. If the tab_group widget is a manager, its children are all members of the tab group (unless they are made into separate tab groups). If the widget is a primitive, it is its own tab group. Certain widgets must not be included with other widgets within a tab group. For example, each List, Scrollbar, OptionMenu, or multi-line Text widget must be placed in a tab group by itself, since these widgets define special behavior for the arrow or Tab keys, which prevents the use of these keys for widget traversal. The inverse routine is XmRemoveTabGroup().

See Also

`XmGetTabGroup`(1), `XmRemoveTabGroup`(1), `XmManager`(2), `XmPrimitive`(2).

Name

XmAddToPostFromList – make a menu accessible from a widget.

Synopsis

#include <Xm/RowColumn.h>

void XmAddToPostFromList (Widget *menu*, Widget *widget*)

Inputs

menu	Specifies a menu widget
widget	Specifies the widget from which to make menu accessible

Availability

In Motif 2.0 and later, the function prototype is removed from RowColumn.h, although there is otherwise no indication that the procedure is obsolete.

Description

`XmAddToPostFromList`() is a convenience function which makes menu accessible from widget. There is no limit to how many widgets may share the same menu. The event sequence required to popup the menu is the same in each widget context.

Usage

Rather than creating a new and identical hierarchy for each context in which a pulldown or popup menu is required, a single menu can be created and shared. If the type of the menu is XmMENU_PULLDOWN, the value of the XmNsubMenuId resource of widget is set to menu. If the type of the menu is XmMENU_POPUP, button and key press event handlers are added to widget in order to post the menu.

There are implicit assumptions that widget is a CascadeButton or CascadeButtonGadget when menu is XmMENU_PULLDOWN, and that widget is not a Gadget when menu is XmMENU_POPUP. These are not checked by the procedure.

See Also

`XmGetPostedFromWidget`(1), `XmRemoveFromPostFromList`(1), `XmCascadeButton`(2), `XmCascadeButtonGadget`(2), `XmGadget`(2), `XmPopupMenu`(2), `XmPulldownMenu`(2), `XmRowColumn`(2).

Name

XmAddWMProtocolCallback – add client callbacks to an XA_WM_PROTOCOLS protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmAddWMProtocolCallback (  Widget          shell,
                                Atom            protocol,
                                XtCallbackProc  callback,
                                XtPointer       closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
protocol	Specifies the protocol atom.
callback	Specifies the procedure to invoke when the protocol message is received.
closure	Specifies any client data that is passed to the callback.

Description

`XmAddWMProtocolCallback`() is a convenience routine that calls `XmAddProtocolCallback`() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. To communicate using a protocol, a client sends a ClientMessage event containing a property and protocol, and the receiving client responds by calling the associated protocol callback routine. `XmAddWMProtocolCallback`() allows you to register these callback routines with the window manager protocol property. The inverse routine is `XmRemoveWMProtocolCallback`().

Example

The following code fragment shows the use of `XmAddWMProtocolCallback`() to save the state of an application using the WM_SAVE_YOURSELF protocol:

```
Atom wm_save_yourself;

wm_save_yourself = XInternAtom[1] (XtDisplay
                                   (toplevel),
```

1.From Motif 2.0, XmInternAtom() is marked for deprecation.

```
                                        "WM_SAVE_YOURSELF"
                                        , False);
    XmAddWMProtocols (toplevel, &wm_save_yourself, 1);
    XmAddWMProtocolCallback (toplevel,
                             wm_save_yourself,
                             save_state, toplevel);
```

save_state is a callback routine that saves the state of the application.

See Also

XmAddProtocolCallback(1), XmInternAtom(1), XmRemoveWMProtocolCallback(1), VendorShell(2).

Name

XmAddWMProtocols – add the XA_WM_PROTOCOLS protocols to the protocol manager.

Synopsis

#include <Xm/Protocols.h>

void `XmAddWMProtocols` (Widget *shell*, Atom **protocols*, Cardinal *num_protocols*)

Inputs

shell	Specifies the widget associated with the protocol property.
protocols	Specifies a list of protocol atoms.
num_protocols	Specifies the number of atoms in protocols.

Description

XmAddWMProtocols() is a convenience routine that calls XmAddProtocols() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. `XmAddWMProtocols`() allows you to add this protocol so that it can be understood by your application. The inverse routine is `XmRemoveWMProtocols`(). To communicate using a protocol, a client sends a ClientMessage event containing a property and protocol, and the receiving client responds by calling the associated protocol callback routine. Use `XmAddWMProtocolCallback`() to add a callback function to be executed when a client message event containing the specified protocol atom is received.

Example

The following code fragment shows the use of `XmAddWMProtocols`() to add the window manager protocols, so that the state of an application can be saved using the WM_SAVE_YOURSELF protocol:

```
Atom wm_save_yourself;

wm_save_yourself = XmInternAtom (XtDisplay
                                 (toplevel),
                                 "WM_SAVE_YOURSELF"
                                 , False);

XmAddWMProtocols (toplevel, &wm_save_yourself, 1);
```

```
XmAddWMProtocolCallback (toplevel,
                         wm_save_yourself,
                         save_state, toplevel);
```

save_state is a callback routine that saves the state of the application.

See Also

XmAddProtocols(1), XmAddWMProtocolCallback(1), XmInternAtom(1), XmRemoveWMProtocols(1), VendorShell(2).

Name

XmCascadeButtonHighlight, XmCascadeButtonGadgetHighlight – set the highlight state of a CascadeButton.

Synopsis

#include <Xm/CascadeB.h>

void XmCascadeButtonHighlight (Widget *cascadeButton*, Boolean *highlight*)

#include <Xm/CascadeBG.h>

void XmCascadeButtonGadgetHighlight (Widget *cascadeButton*, Boolean *highlight*)

Inputs

cascadeButton	Specifies the CascadeButton or CascadeButtonGadget.
highlight	Specifies the highlight state.

Description

`XmCascadeButtonHighlight`() sets the state of the shadow highlight around the specified *cascadeButton*, which can be a CascadeButton or a CascadeButtonGadget.

`XmCascadeButtonGadgetHighlight`() sets the highlight state of the specified *cascadeButton*, which must be a CascadeButtonGadget.

Both routines draw the shadow if *highlight* is True and erase the shadow if *highlight* is False.

Usage

CascadeButtons do not normally display a shadow like other buttons, so the highlight shadow is often used to show that the button is armed. `XmCascadeButtonHighlight`() and `XmCascadeButtonGadgetHighlight`() provide a way for you to cause the shadow to be displayed.

See Also

`XmCascadeButton`(2), `XmCascadeButtonGadget`(2).

Name

XmChangeColor – update the colors for a widget.

Synopsis

void XmChangeColor (Widget widget, Pixel background)

Inputs

widget	Specifies the widget whose colors are to be changed.
background	Specifies the background color.

Description

`XmChangeColor`() changes all of the colors for the specified widget based on the new background color. The routine recalculates the foreground color, the select color, the arm color, the trough color, and the top and bottom shadow colors and updates the corresponding resources for the widget.

Usage

`XmChangeColor`() is a convenience routine for changing all of the colors for a widget, based on the background color. Without the routine, an application would have to call `XmGetColors`() to get the new colors and then set the XmNforeground, XmNtopShadowColor, XmNbottomShadowColor, XmNtroughColor, XmNarmColor, XmNselectColor resources for the widget with `XtSetValues`(). The XmNhighlightColor is set to the value of the XmNforeground.

`XmChangeColor`() calls `XmGetColors`() internally to allocate the required pixels. In Motif 1.2 and earlier, this uses the default color calculation procedure unless a customized color calculation procedure has been set with `XmSetColorCalculation`(). In Motif 2.0 and later, color calculation can be specified on a per-screen basis, and any specified XmNcolorCalculationProc procedure of the XmScreen object associated with the widget is used in preference.

See Also

`XmGetColorCalculation(1), XmGetColors(1), XmSetColorCalculation(1), XmScreen(2).`

Name

XmClipboardBeginCopy – set up storage for a clipboard copy operation.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardBeginCopy (  Display     *display,
                            Window      window,
                            XmString    clip_label,
                            Widget      widget,
                            VoidProc    callback,
                            long        *item_id)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies a window ID that identifies the client to the clipboard.
clip_label	Specifies a label that is associated with the data item.
widget	Specifies the widget that receives messages requesting data that has been passed by name.
callback	Specifies the callback function that is called when the clipboard needs data that has been passed by name.

Outputs

item_id	Returns the ID assigned to the data item.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardBeginCopy`() is a convenience routine that calls `XmClipboardStartCopy`() with identical arguments and with a timestamp of CurrentTime.

Usage

`XmClipboardBeginCopy`() can be used to start a normal copy operation or a copy-by-name operation. In order to pass data by name, the *widget* and *callback* arguments to `XmClipboardBeginCopy`() must be specified.

Procedures

The VoidProc has the following format:

typedef void (*VoidProc) (Widget *widget*, int **data_id*, int **private_id*, int **reason*)

The VoidProc takes four arguments. The first argument, *widget*, is the widget passed to the callback routine, which is the same widget as passed to `XmClipboardBeginCopy`(). The *data_id* argument is the ID of the data item that is returned by `XmClipboardCopy`() and *private_id* is the private data passed to XmClipboardCopy().

The *reason* argument takes the value XmCR_CLIPBOARD_DATA_REQUEST, which indicates that the data must be copied to the clipboard, or XmCR_CLIPBOARD_DATA_DELETE, which indicates that the client can delete the data from the clipboard. Although the last three parameters are pointers to integers, the values are read-only and changing them has no effect.

See Also

`XmClipboardCancelCopy`(1), `XmClipboardCopy`(1), `XmClipboardCopyByName`(1), `XmClipboardEndCopy`(1), `XmClipboardStartCopy`(1).

Name

XmClipboardCancelCopy – cancel a copy operation to the clipboard.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardCancelCopy (Display **display*, Window *window*, long *item_id*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window Specifies a window ID that identifies the client to the clipboard.
item_id Specifies the ID of the data item.

Returns

ClipboardSuccess on success, ClipboardLocked if the clipboard is locked by another application, or ClipboardFail on failure.

Description

`XmClipboardCancelCopy`() cancels the copy operation that is in progress and frees temporary storage that has been allocated for the operation. The function returns ClipboardFail if `XmClipboardStartCopy`() has not been called or if the data item has too many formats.

Usage

A call to `XmClipboardCancelCopy`() is valid only between calls to `XmClipboardStartCopy`() and `XmClipboardEndCopy`(). `XmClipboardCancelCopy`() can be called instead of `XmClipboardEndCopy`() when you need to terminate a copying operation before it completes. If you have previously locked the clipboard, `XmClipboardCancelCopy`() unlocks it, so you should not call `XmClipboardUnlock`().

See Also

`XmClipboardBeginCopy`(1), `XmClipboardCopy`(1), `XmClipboardEndCopy`(1), `XmClipboardStartCopy`(1).

Name

XmClipboardCopy – copy a data item to temporary storage for later copying to the clipboard.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardCopy ( Display        *display,
                      Window         window,
                      long           item_id,
                      char           *format_name,
                      XtPointer      buffer,
                      unsigned long  length,
                      long           private_id,
                      long           *data_id)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies a window ID that identifies the client to the clipboard.
item_id	Specifies the ID of the data item.
format_name	Specifies the name of the format of the data item.
buffer	Specifies the buffer from which data is copied to the clipboard.
length	Specifies the length of the data being copied to the clipboard.
private_id	Specifies the private data that is stored with the data item.

Outputs

data_id	Returns an ID for a data item that is passed by name.

Returns

ClipboardSuccess on success, ClipboardLocked if the clipboard is locked by another application, or ClipboardFail on failure.

Description

`XmClipboardCopy`() copies the data item specified by buffer to temporary storage. The data item is moved to the clipboard data structure when `XmClipboardEndCopy`() is called. The *item_id* is the ID of the data item returned by `XmClipboardStartCopy`() and *format_name* is a string that describes the type of the data.

Since the data item is not actually stored in the clipboard until XmClipboardEndCopy() is called, multiple calls to XmClipboardCopy() add data item formats to the same data item or will append data to an existing format. The function returns ClipboardFail if XmClipboardStartCopy() has not been called or if the data item has too many formats.

Usage

XmClipboardCopy() is called between calls to XmClipboardStartCopy() and XmClipboardEndCopy(). If you need to make multiple calls to XmClipboardCopy() to copy a large amount of data, you should call XmClipboardLock() to lock the clipboard for the duration of the copy operation.

When there is a large amount of clipboard data and the data is unlikely to be retrieved, it can be copied to the clipboard by name. Since the data itself is not copied to the clipboard until it is requested with a retrieval operation, copying by name can improve performance. To pass data by name, call XmClipboardCopy() with buffer specified as NULL. A unique number is returned in data_id that identifies the data item for later use. When another application requests data that has been passed by name, a callback requesting the actual data will be sent to the application that owns the data and the owner must then call XmClipboardCopyByName() to transfer the data to the clipboard. Once data that is passed by name has been deleted from the clipboard, a callback notifies the owner that the data is no longer needed.

Example

The following callback shows the sequence of calls needed to copy data to the clipboard:

```
void to_clipbd ( Widget    widget,
                 XtPointer client_data,
                 XtPointer call_data)
{
   long     item_id = 0;
   int      status;
   XmString clip_label;
   char     buffer[32];
   Display  *dpy   = XtDisplayOfObject (widget);
   Window   window = XtWindowOfObject (widget);
```

```
        char     *data  = (char *) client_data;
        (void) sprintf (buffer, "%s", data);
        clip_label = XmStringCreateLocalized ("Data");
        /* start a copy; retry until unlocked */
        do
           status = XmClipboardStartCopy (dpy, window,
                                          clip_label,
                                          CurrentTime,
                                          NULL, NULL,
                                          &item_id);
        while (status == ClipboardLocked);
        XmStringFree (clip_label);
        /* copy the data; retry until unlocked */
        do {
           status = XmClipboardCopy (dpy, window,
                                     item_id, "STRING",
                                     (XtPointer) buffer,
                                     (unsigned long) strlen
                                     (buffer) + 1,
                                     (long) 0, (long *) 0);
        } while (status == ClipboardLocked);
        /* end the copy; retry until unlocked */
        do
           status = XmClipboardEndCopy (dpy, window,
           item_id);
        while (status == ClipboardLocked);
      }
```

See Also

XmClipboardBeginCopy(1), XmClipboardCancelCopy(1), XmClipboardCopyByName(1), XmClipboardEndCopy(1), XmClipboardStartCopy(1).

Name

XmClipboardCopyByName – copy a data item passed by name.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardCopyByName (     Display         *display,
                                Window          window,
                                long            data_id,
                                XtPointer       buffer,
                                unsigned long   length,
                                long            private_id)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

data_id Specifies the ID number assigned to the data item by XmClipboardCopy().

buffer Specifies the buffer from which data is copied to the clipboard.

length Specifies the length of the data being copied to the clipboard.

private_id Specifies the private data that is stored with the data item.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardCopyByName`() copies the actual data to the clipboard for a data item that has been previously passed by name. The data that is copied is specified by buffer. The *data_id* is the ID assigned to the data item by `XmClipboardCopy`().

Usage

`XmClipboardCopyByName`() is typically used for incremental copying; new data is appended to existing data with each call to `XmClipboardCopyByName`(). If you need to make multiple calls to `XmClipboardCopyByName`() to copy a large amount of data, you should call `XmClipboardLock`() to lock the clipboard for the duration of the copy operation.

Copying by name improves performance when there is a large amount of clipboard data and when this data is likely never to be retrieved, since the data itself is not copied to the clipboard until it is requested with a retrieval operation. Data is passed by name when `XmClipboardCopy`() is called with a *buffer* value of NULL. When a client requests the data passed by name, the callback registered

by XmClipboardStartCopy() is invoked. See XmClipboardStartCopy() for more information about the format of the callback. This callback calls XmClipboardCopyByName() to copy the actual data to the clipboard.

Example

The following XmCutPasteProc callback shows the use of XmClipboardCopyByName() to copy data passed by name:

```
void copy_by_name ( Widget  widget,
                    long    *data_id,
                    long    *private_id;
                    int     *reason)
{
   Display *dpy    = XtDisplay (toplevel);
   Window   window = XtWindow (toplevel);
   int      status;
   char     buffer[32];

   if (*reason == XmCR_CLIPBOARD_DATA_REQUEST) {
      (void) sprintf (buffer, "stuff");

      do
         status = XmClipboardCopyByName (dpy, win-
                        dow, *data_id,
                                (XtPointer) buffer,
                                (unsigned long)
                                strlen (buffer)+1,
                                *private_id);
      while (status != ClipboardSuccess);
}
```

See Also

XmClipboardBeginCopy(1), XmClipboardCopy(1), XmClipboardEndCopy(1), XmClipboardStartCopy(1).

Name

XmClipboardEndCopy – end a copy operation to the clipboard.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardEndCopy (Display **display*, Window *window*, long *item_id*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window Specifies a window ID that identifies the client to the clipboard.
item_id Specifies the ID of the data item.

Returns

ClipboardSuccess on success, ClipboardLocked if the clipboard is locked by another application, or ClipboardFail on failure.

Description

`XmClipboardEndCopy`() locks the clipboard, places data that has been accumulated by calling `XmClipboardCopy`() into the clipboard data structure, and then unlocks the clipboard. The *item_id* is the ID of the data item returned by XmClipboardStartCopy(). The function returns ClipboardFail if `XmClipboardStartCopy`() has not been called previously.

Usage

`XmClipboardEndCopy`() frees temporary storage that was allocated by `XmClipboardStartCopy`(). `XmClipboardStartCopy`() must be called before `XmClipboardEndCopy`(), which does not need to be called if `XmClipboardCancelCopy`() has already been called.

Example

The following callback shows the sequence of calls needed to copy data to the clipboard:

```
static void to_clipbd ( Widget      widget,
                        XtPointer   client_data,
                        XtPointer   call_data)
{
   long      item_id = 0;
   int       status;
   XmString clip_label;
   char      buffer[32];
   Display  *dpy    = XtDisplayOfObject (widget);
   Window    window = XtWindowOfObject (widget);
```

```
    char     *data   = (char *) client_data;

    (void) sprintf (buffer, "%s", data);
    clip_label = XmStringCreateLocalized ("Data");

    /* start a copy; retry until unlocked */
    do
       status = XmClipboardStartCopy (dpy, window,
       clip_label,
                                      CurrentTime,
                                      NULL, NULL,
                                      &item_id);
    while (status == ClipboardLocked);
    XmStringFree (clip_label);

    /* copy the data; retry until unlocked */
    do
       status = XmClipboardCopy (dpy, window,
       item_id, "STRING",
                                 (XtPointer) buffer,
                                 (unsigned
                                 long)strlen(buffer)+1,
                                 0, NULL);
    while (status == ClipboardLocked);

    /* end the copy; retry until unlocked */
    do
       status = XmClipboardEndCopy (dpy, window,
       item_id);
    while (status == ClipboardLocked);
  }
```

See Also

XmClipboardBeginCopy(1), XmClipboardCancelCopy(1), XmClipboardCopy(1), XmClipboardCopyByName(1), XmClipboardStartCopy(1).

Name

XmClipboardEndRetrieve – end a copy operation from the clipboard.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardEndRetrieve (Display **display*, Window *window*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

XmClipboardEndRetrieve() ends the incremental copying of data from the clipboard.

Usage

A call to `XmClipboardEndRetrieve`() is preceded by a call to `XmClipboardStartRetrieve`(), which begins the incremental copy, and calls to `XmClipboardRetrieve`(), which incrementally retrieve the data items from clipboard storage. `XmClipboardStartRetrieve`() locks the clipboard and it remains locked until `XmClipboardEndRetrieve`() is called.

Example

The following code fragment shows the sequence of calls needed to perform an incremental retrieve. Note that this code does not store the data as it is retrieved:

```
int           status;
unsigned long received;
char          buffer[32];
Display      *dpy    = XtDisplayOfObject (widget);
Window        window = XtWindowOfObject (widget);

do
   status = XmClipboardStartRetrieve (dpy, window,
   CurrentTime);
while (status == ClipboardLocked);

do {
   /* retrieve data from clipboard */
   status = XmClipboardRetrieve (dpy, window,
```

```
                                        "STRING",
                                        (XtPointer) buffer,
                                        (unsigned long)
                                        sizeof (buffer),
                                        &received,
                                        (long *) 0);
        } while (status == ClipboardTruncate);

        status = XmClipboardEndRetrieve (dpy, window);
```

See Also

XmClipboardRetrieve(1), XmClipboardStartRetrieve(1).

Name

XmClipboardInquireCount – get the number of data item formats available on the clipboard.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardInquireCount (Display       *display,
                             Window        window,
                             int           *count,
                             unsigned long *max_length)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

Outputs

count Returns the number of data item formats available for the data on the clipboard.

max_length Returns the maximum length of data item format names.

Returns

ClipboardSuccess on success, ClipboardLocked if the clipboard is locked by another application, or ClipboardNoData if there is no data on the clipboard.

Description

`XmClipboardInquireCount`() returns the number of data formats available for the current clipboard data item and the length of its longest format name. The count includes the formats that were passed by name. If there are no formats available, count is 0 (zero).

Usage

To inquire about the formats of the data on the clipboard, you use `XmClipboardInquireCount`() and `XmClipboardInquireFormat`() in conjunction. `XmClipboardInquireCount`() returns the number of formats for the data item and `XmClipboardInquireFormat`() allows you to iterate through all of the formats.

See Also

`XmClipboardInquireFormat`(1).

Name

XmClipboardInquireFormat – get the specified clipboard data format name.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardInquireFormat (  Display        *display,
                                Window         window,
                                int            index,
                                XtPointer      format_name_buf,
                                unsigned long  buffer_len,
                                unsigned long  *copied_len)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

index Specifies the index of the format name to retrieve.

buffer_len Specifies the length of format_name_buf in bytes.

Outputs

format_name_buf Returns the format name.

copied_len Returns the length (in bytes) of the string copied to format_name_buf.

Returns

ClipboardSuccess on success, ClipboardLocked if the clipboard is locked by another application, ClipboardTruncate if format_name_buf is not long enough to hold the returned data, or ClipboardNoData if there is no data on the clipboard.

Description

`XmClipboardInquireFormat`() returns a format name for the current data item in the clipboard. The format name returned is specified by index, where 1 refers to the first format. If index exceeds the number of formats for the data item, then `XmClipboardInquireFormat`() returns a value of 0 (zero) in the copied_len argument. `XmClipboardInquireFormat`() returns the format name in the format_name_buf argument. This argument is a buffer of a fixed length that is allocated by the programmer. If the buffer is not large enough to hold the format name, the routine copies as much of the format name as will fit in the buffer and returns ClipboardTruncate.

Usage

To inquire about the formats of the data on the clipboard, you use `XmClipboardInquireCount`() and `XmClipboardInquireFormat`() in conjunction. `XmClipboardInquireCount`() returns the number of formats for the data item and `XmClipboardInquireFormat`() allows you to iterate through all of the formats.

See Also

`XmClipboardInquireCount`(1).

Name

XmClipboardInquireLength – get the length of the data item on the clipboard.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardInquireLength ( Display        *display,
                               Window         window,
                               char           *format_name,
                               unsigned long  *length)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

format_name Specifies the format name for the data.

Outputs

length Returns the length of the data item for the specified format.

Returns

ClipboardSuccess on success, ClipboardLocked if the clipboard is locked by another application, or ClipboardNoData if there is no data on the clipboard for the requested format.

Description

`XmClipboardInquireLength`() returns the length of the data stored under the specified *format_name* for the current clipboard data item. If no data is found corresponding to *format_name* or if there is no item on the clipboard, `XmClipboardInquireLength`() returns a length of 0 (zero). When a data item is passed by name, the length of the data is assumed to be passed in a call to `XmClipboardCopy`(), even though the data has not yet been transferred to the clipboard.

Usage

`XmClipboardInquireLength`() provides a way for an application to find out how much data is on the clipboard, so that it can allocate a buffer that is large enough to retrieve the data with one call to `XmClipboardRetrieve`().

Example

The following code fragment demonstrates how to use `XmClipboardInquireLength`() to retrieve all of the data on the clipboard:

```
int           status;
unsigned long recvd, length;
```

```
char         *data;
Display      *dpy    = XtDisplayOfObject (widget);
Window        window = XtWindowOfObject (widget);

do
   status = XmClipboardInquireLength (dpy, window,
                                      "STRING",
                                      &length);
while (status == ClipboardLocked);

if (length != 0) {
   data = XtMalloc ((unsigned) (length+1) * sizeof
   (char));

   do
      status = XmClipboardRetrieve (dpy, window,
                                    "STRING",
                                     (XtPointer)
                                     data,
                                     (unsigned long)
                                     length+1,
                                     &recvd, (long *)
                                     0);
   while (status == ClipboardLocked);

   if (status != ClipboardSuccess || recvd !=
   length) {
      XtWarning ("Failed to receive all clipboard
      data");
   }
}
```

See Also

XmClipboardRetrieve(1).

Name

XmClipboardInquirePendingItems – get a list of pending data ID/private ID pairs.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardInquirePendingItems (  Display                 *display,
                                      Window                  window,
                                      char
*format_name,
                                      XmClipboardPendingList  *item_list,
                                      unsigned long           *count)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

format_name Specifies the format name for the data.

Outputs

item_list Returns an array of data_id/private_id pairs for the specified format.

count Returns the number of items in the item_list array.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardInquirePendingItems`() returns for the specified *format_name* a list of pending data items, represented by *data_id*/*private_id* pairs. The *data_id* and *private_id* arguments are specified in the clipboard functions for copying and retrieving. A data item is considered pending under these conditions: the application that owns the data item originally passed it by name, the application has not yet copied the data, and the data item has not been deleted from the clipboard. If there are no pending items for the specified *format_name*, the routine returns a count of 0 (zero). The application is responsible for freeing the memory that is allocated by `XmClipboardInquirePendingItems`() to store the list. Use XtFree() to free the memory.

Usage

An application should call XmClipboardInquirePendingItems() before exiting, to determine whether data that has been passed by name should be copied to the clipboard.

Structures

The XmClipboardPendingList is defined as follows:

```
typedef struct {
    long DataId;
    long PrivateId;
} XmClipboardPendingRec, *XmClipboardPendingList;
```

See Also

`XmClipboardStartCopy`(1).

Name

XmClipboardLock – lock the clipboard.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardLock (Display **display*, Window *window*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardLock`() locks the clipboard on behalf of an application, which prevents access to the clipboard by other applications. If the clipboard has already been locked by another application, the routine returns ClipboardLocked. If the same application has already locked the clipboard, the lock level is increased.

Usage

An application uses `XmClipboardLock`() to ensure that clipboard data is not changed by calls to clipboard functions by other applications. An application does not need to lock the clipboard between calls to `XmClipboardStartRetrieve`() and `XmClipboardEndRetrieve`(), because the clipboard is locked automatically between these calls. `XmClipboardUnlock`() allows other applications to access the clipboard again.

See Also

`XmClipboardEndCopy`(1), `XmClipboardEndRetrieve`(1), `XmClipboardStartCopy`(1), `XmClipboardStartRetrieve`(1), `XmClipboardUnlock`(1).

Name

XmClipboardRegisterFormat – register a new format for clipboard data items.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardRegisterFormat (Display **display*, char **format_name*, int *format_length*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
format_name Specifies the string name for the format.
format_length Specifies the length of the format in bits (0, 8, 16, or 32).

Returns

ClipboardSuccess on success, ClipboardBadFormat if the format is not properly specified, ClipboardLocked if the clipboard is locked by another application, or ClipboardFail on failure.

Description

`XmClipboardRegisterFormat`() registers a new format having the specified *format_name* and *format_length*. `XmClipboardRegisterFormat`() returns ClipboardFail if the format is already registered with the specified length or ClipboardBadFormat if *format_name* is NULL or *format_length* is not 0, 8, 16, or 32 bits.

Usage

`XmClipboardRegisterFormat`() is used by applications that support cutting and pasting of arbitrary data types. Every format that is stored on the clipboard needs to have a length associated with it, so that clipboard operations between applications that run on platforms with different byte-swapping orders function properly. Format types that are defined by the ICCCM are preregistered. If *format_length* is 0, `XmClipboardRegisterFormat`() searches through the preregistered format types, and returns ClipboardSuccess if *format_name* is found, ClipboardFail otherwise.

If you are registering your own data structure as a format, you should choose an appropriate name, and use 32 as the format size.

See Also

`XmClipboardStartCopy`(1).

Name

XmClipboardRetrieve – retrieve a data item from the clipboard.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardRetrieve (Display        *display,
                         Window         window,
                         char           *format_name,
                         XtPointer      buffer,
                         unsigned long  length,
                         unsigned long  *num_bytes,
                         long           *private_id)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

format_name Specifies the format name for the data.

buffer Specifies the buffer to which the clipboard data is copied.

length Specifies the length of buffer.

Outputs

num_bytes Returns the number of bytes of data copied into buffer.

private_id Returns the private data that was stored with the data item.

Returns

ClipboardSuccess on success, ClipboardLocked if the clipboard is locked by another application, ClipboardTruncate if buffer is not long enough to hold the returned data, or ClipboardNoData if there is no data on the clipboard for the requested format.

Description

`XmClipboardRetrieve`() fetches the current data item from the clipboard and copies it to the specified buffer. The format_name specifies the type of data being retrieved. The num_bytes parameter returns the amount of data that is copied into buffer. The routine returns ClipboardTruncate when all of the data does not fit in the buffer, to indicate that more data remains to be copied.

Usage

`XmClipboardRetrieve`() can be used to retrieve data in one large piece or in multiple smaller pieces. To retrieve data in one chunk, call `XmClipboardInquireLength`() to determine the size of the data on the clipboard. Multiple calls to `XmClipboardRetrieve`() with the same format_name, between calls to `XmClipboardStartRetrieve`() and `XmClipboardEndRetrieve`(),

copy data incrementally. Since the clipboard is locked by a call to `XmClipboardStartRetrieve`(), it is suggested that your application call any clipboard inquiry routines between this call and the first call to `XmClipboardRetrieve`()[1].

Example

The following code fragment shows the sequence of calls needed to perform an incremental retrieve. Note that this code does not store the data as it is retrieved:

```
int           status;
unsigned long received;
char          buffer[32];
Display      *dpy    = XtDisplayOfObject (widget);
Window        window = XtWindowOfObject (widget);

do
   status = XmClipboardStartRetrieve (dpy, window,
   CurrentTime);
while (status == ClipboardLocked);

do {
   /* retrieve data from clipboard */
   status = XmClipboardRetrieve (dpy, window,
   "STRING",
                                 (XtPointer) buffer,
                                 (unsigned long)
                                 sizeof (buffer),
                                 &received,
                                 (long *) 0);
} while (status == ClipboardTruncate);

status = XmClipboardEndRetrieve (dpy, window);
```

See Also

`XmClipboardEndRetrieve`(1), `XmClipboardInquireLength`(1), `XmClipboardLock`(1), `XmClipboardStartRetrieve`(1), `XmClipboardUnlock`(1).

1.Erroneously given as ClipboardRetrieve() in 1st and 2nd editions.

Name

XmClipboardStartCopy – set up storage for a clipboard copy operation.

Synopsis

```
#include <Xm/CutPaste.h>

int XmClipboardStartCopy ( Display          *display,
                           Window           window,
                           XmString         clip_label,
                           Time             timestamp,
                           Widget           widget,
                           XmCutPasteProc   callback,
                           long             *item_id)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies a window ID that identifies the client to the clipboard.
clip_label	Specifies a label that is associated with the data item.
timestamp	Specifies the time of the event that triggered the copy operation.
widget	Specifies the widget that receives messages requesting data that has been passed by name.
callback	Specifies the callback function that is called when the clipboard needs data that has been passed by name.

Outputs

item_id	Returns the ID assigned to the data item.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardStartCopy`() creates the storage and data structures that receive clipboard data. During a cut or copy operation, an application calls this function to initiate the operation. The data that is copied to the structures becomes the next clipboard data item.

Several arguments to `XmClipboardStartCopy`() provide identifying information. The *window* argument specifies the window that identifies the application to the clipboard; an application should pass the same window ID to each clipboard routine that it calls. *clip_label* assigns a text string to the data item that could be used as the label for a clipboard viewing window. The *timestamp* passed

to the routine must be a valid timestamp. The *item_id* argument returns a number that identifies the data item. An application uses this number to specify the data item in other clipboard calls.

Usage

Since copying a large piece of data to the clipboard can take a long time and it is possible that the data will never be requested by another application, the clipboard copy routines provide a mechanism to copy data by name. When a clipboard data item is passed by name, the application does not need to copy the data to the clipboard until it has been requested by another application. In order to pass data by name, the widget and callback arguments to `XmClipboardStartCopy`() must be specified. *widget* specifies the ID of the widget that receives messages requesting that data be passed by name. All of the message handling is done by the clipboard operations, so any valid widget ID can be used. *callback* specifies the procedure that is invoked when the clipboard needs the data that was passed by name and when the data item is removed from the clipboard. The *callback* function copies the actual data to the clipboard using `XmClipboardCopyByName`().

Example

The following routines show the sequence of calls needed to copy data by name. The to_clipbd callback shows the copying of data and copy_by_name shows the callback that actually copies the data:

```
void copy_by_name ( Widget  widget,
                    long    *data_id,
                    long    *private_id,
                    int     *reason)
{
   Display      *dpy    = XtDisplay (toplevel);
   Window        window = XtWindow (toplevel);
   int           status;
   char          buffer[32];

   if (*reason == XmCR_CLIPBOARD_DATA_REQUEST) {
      (void) sprintf (buffer, "stuff");

      do
         status = XmClipboardCopyByName (dpy, win-
         dow, *data_id,
                                (XtPointer) buffer,
                                (unsigned long)
                                strlen (buffer)+1,
                                *private_id);
```

```
      while (status != ClipboardSuccess);
   }
}

void to_clipbd ( Widget      widget,
                 XtPointer   client_data,
                 XtPointer   call_data)
{
   unsigned long item_id = 0;
   int            status;
   XmString       clip_label;
   Display       *dpy    = XtDisplayOfObject
   (widget);
   Window         window = XtWindowOfObject
   (widget);
   unsigned long size = DATA_SIZE;
   char          *data = (char *) client_data;

   clip_label = XmStringCreateLocalized ("Data");

   /* start a copy; retry until unlocked */
   do
      status = XmClipboardStartCopy (dpy, window,
      clip_label,
                                     CurrentTime,
                                     widget,
                                     copy_by_name,
                                     &item_id);
   while (status == ClipboardLocked);
   XmStringFree (clip_label);

   /* copy the data; retry until unlocked */
   do
      status = XmClipboardCopy (dpy, window,
      item_id,
                                "STRING", NULL,
                                size, 0, NULL);
   while (status == ClipboardLocked);

   /* end the copy; retry until unlocked */
   do
      status = XmClipboardEndCopy (dpy, window,
      item_id);
   while (status == ClipboardLocked);
}
```

Procedures

The XmCutPasteProc has the following format:

typedef void (*XmCutPasteProc) (Widget *widget*, long **data_id*, long **private_id*, int **reason*)

An XmCutPasteProc takes four arguments. The first argument, *widget*, is the widget passed to the callback routine, which is the same widget as passed to `XmClipboardBeginCopy`(). The *data_id* argument is the ID of the data item that is returned by `XmClipboardCopy`() and *private_id* is the private data passed to `XmClipboardCopy`().

The *reason* argument takes the value XmCR_CLIPBOARD_DATA_REQUEST, which indicates that the data must be copied to the clipboard, or XmCR_CLIPBOARD_DATA_DELETE, which indicates that the client can delete the data from the clipboard. Although the last three parameters are pointers to integers, the values are read-only and changing them has no effect.

See Also

`XmClipboardBeginCopy`(1), `XmClipboardCancelCopy`(1), `XmClipboardCopy(1)`, `XmClipboardCopyByName`(1), `XmClipboardEndCopy`(1), `XmClipboardLock`(1), `XmClipboardRegisterFormat`(1), `XmClipboardUndoCopy`(1), `XmClipboardUnlock`(1), `XmClipboardWithdrawFormat`(1).

Name

XmClipboardStartRetrieve – start a clipboard retrieval operation.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardStartRetrieve (Display **display*, Window *window*, Time *timestamp*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

timestamp Specifies the time of the event that triggered the retrieval operation.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardStartRetrieve`() starts a clipboard retrieval operation by telling the clipboard that an application is ready to start copying data from the clipboard. `XmClipboardStartRetrieve`() locks the clipboard until `XmClipboardEndRetrieve`() is called. The *window* argument specifies the window that identifies the application to the clipboard; an application should pass the same window ID to each clipboard routine that it calls. The *timestamp* passed to the routine must be a valid timestamp.

Usage

Multiple calls to `XmClipboardRetrieve`() with the same *format_name*, between calls to `XmClipboardStartRetrieve`() and `XmClipboardEndRetrieve`(), copy data incrementally.

Example

The following code fragment shows the sequence of calls needed to perform an incremental retrieve. Note that this code does not store the data as it is retrieved:

```
int           status;
unsigned long received;
char          buffer[32];
Display      *dpy = XtDisplayOfObject (widget);
Window        window = XtWindowOfObject (widget);

do
```

```
   status = XmClipboardStartRetrieve (dpy, window,
   CurrentTime);
while (status == ClipboardLocked);

do {
   /* retrieve data from clipboard */
   status = XmClipboardRetrieve (dpy, window,
   "STRING",
                                 (XtPointer) buffer,
                                 (unsigned long)
                                 sizeof (buffer),
                                 &received,
                                 (long *) 0);
} while (status == ClipboardTruncate);

status = XmClipboardEndRetrieve (dpy, window);
```

See Also

XmClipboardEndRetrieve(1), XmClipboardInquireCount(1), XmClipboardInquireFormat(1), XmClipboardInquireLength(1), XmClipboardInquirePendingItems(1), XmClipboardLock(1), XmClipboardRetrieve(1), XmClipboardUnlock(1).

Name

XmClipboardUndoCopy – remove the last item copied to the clipboard.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardUndoCopy (Display **display*, Window *window*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardUndoCopy`() deletes the item most recently placed on the clipboard, provided that the application that originally placed the item has matching values for display and window. If the values do not match, no action is taken. The routine also restores any data item that was deleted from the clipboard by the call to `XmClipboardCopy`().

Usage

Motif maintains a two-deep stack of items that have been placed on the clipboard. Once an item has been copied to the clipboard, the copy can be undone by calling `XmClipboardUndoCopy`(). Calling this routine twice undoes the last undo operation.

See Also

`XmClipboardBeginCopy`(1), `XmClipboardCopy`(1), `XmClipboardCopyByName`(1), `XmClipboardEndCopy`(1), `XmClipboardStartCopy`(1).

Name

XmClipboardUnlock – unlock the clipboard.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardUnlock (Display **display*, Window *window*, Boolean *remove_all_locks*)

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies a window ID that identifies the client to the clipboard.
`remove_all_locks`	Specifies whether nested locks should be removed.

Returns

ClipboardSuccess on success or ClipboardFail if the clipboard is not locked or if it is locked by another application.

Description

`XmClipboardUnlock`() unlocks the clipboard, which allows other applications to access it. If remove_all_locks is True, all nested locks are removed. If it is False, only one level of lock is removed.

Usage

Multiple calls to `XmClipboardLock`() can increase the lock level, and normally, each `XmClipboardLock`() call requires a corresponding call to `XmClipboardUnlock`(). However, by setting remove_all_locks to True, nested locks can be removed with a single call.

See Also

`XmClipboardBeginCopy`(1), `XmClipboardCancelCopy`(1), `XmClipboardEndCopy`(1), `XmClipboardEndRetrieve`\(1) `XmClipboardLock`(1), `XmClipboardStartCopy`(1), `XmClipboardStartRetrieve`(1).

Name

XmClipboardWithdrawFormat – indicate that an application does not want to supply a data item any longer.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardWithdrawFormat (Display **display*, Window *window*, long *data_id*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

window Specifies a window ID that identifies the client to the clipboard.

data_id Specifies the ID for the passed-by-name data item.

Returns

ClipboardSuccess on success or ClipboardLocked if the clipboard is locked by another application.

Description

`XmClipboardWithdrawFormat`() withdraws a data item that has been passed by name from the clipboard. The *data_id* is the ID that was assigned to the item when it was passed by `XmClipboardCopy`().

Usage

Despite its name, `XmClipboardWithdrawFormat`() does not remove a format specification from the clipboard. The routine provides an application with a way to withdraw data of a particular format from the clipboard.

See Also

`XmClipboardBeginCopy`(1), `XmClipboardCopy`(1), `XmClipboardCopyByName`(1), `XmClipboardStartCopy`(1).

Name

XmComboBoxAddItem – add a compound string to the ComboBox list.

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxAddItem (Widget *widget*, XmString *item*, int *position*, Boolean *unique*)

Inputs

widget	Specifies the ComboBox widget.
item	Specifies the compound string that is added to the ComboBox list.
position	Specifies the position at which to add the new item.
unique	Specifies whether the item must be unique in the list.

Availability

Motif 2.1 and later.

Description

`XmComboBoxAddItem`() is a convenience routine that adds an item into a ComboBox list. `XmComboBoxAddItem`() inserts the specified *item* into the list component of the ComboBox *widget* at the specified *position*. A *position* value of 1 indicates the first location in the list, a *position* value of 2 indicates the second location, and so forth. A value of 0 (zero) specifies the last location in the list. If the value exceeds the current number of items in the list, the *item* is silently appended. If *unique* is true, the item is only added if it does not already appear in the list.

Usage

In order to use this routine, a compound string must be created for the item. The routine calls `XmListAddItemUnselected`() to insert the item into the list component. The ComboBox list takes a copy of the supplied item. It is the responsibility of the programmer to reclaim the space by calling XmStringFree() at an appropriate point.

See Also

`XmComboBoxSelectItem`(1), `XmComboBoxSetItem`(1), `XmComboBoxDeletePos`(1), `XmComboBoxUpdate`(1), `XmComboBox`(2).

Name

XmComboBoxDeletePos – delete an item at the specified position from a ComboBox list.

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxDeletePos (Widget *widget*, int *position*)

Inputs

widget	Specifies the ComboBox widget.
position	Specifies the position from which to delete an item.

Availability

Motif 2.1 and later.

Description

`XmComboBoxDeletePos`() removes the item at the specified *position* from the ComboBox list. The first location within the list is at position 1, the second list item is at position 2, and so forth. A *position* value of 0 (zero) specifies the last location in the list. If the ComboBox list does not have an item at the specified *position*, a warning message is displayed.

Usage

`XmComboBoxDeletePos`() is a convenience routine that allows you to remove an item from a ComboBox list. The routine calls `XmListDeletePos`() on the list component of the ComboBox.

See Also

`XmComboBoxAddItem`(1), `XmComboBoxSelectItem`(1), `XmComboBoxSetItem`(1), `XmComboBoxUpdate`(1), `XmComboBox`(2).

Name

XmComboBoxSelectItem – select an item from a ComboBox list.

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxSelectItem (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the ComboBox widget.
item	Specifies the item that is to be selected.

Availability

Motif 2.1 and later.

Description

`XmComboBoxSelectItem`() selects the first occurrence of the specified *item* in the ComboBox list. If the *item* is found within the list, the value is also inserted into the ComboBox text field. Otherwise, a warning message is displayed.

Usage

`XmComboBoxSelectItem`() is a convenience routine that allows you to select an *item* in the ComboBox list. In order to use this routine, a compound string must be created for the *item*. No ComboBox selection callbacks are invoked as a result of calling this procedure. The routine internally calls `XmListSelect-Pos`() on the list component of the ComboBox, after performing a linear search through the XmNitems of the list: the *item* parameter is used only for the search and is not directly used as the newly selected item. It is the responsibility of the programmer to reclaim any allocated memory for the compound string item by calling `XmStringFree`() at an appropriate time.

See Also

`XmComboBoxAddItem`(1), `XmComboBoxDeletePos`(1), `XmComboBoxSetItem`(1), `XmComboBoxUpdate`(1), `XmComboBox`(2).

Name

XmComboBoxSetItem – select and make visible an item from a ComboBox list.

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxSetItem (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the ComboBox widget.
item	Specifies the item that is to be selected.

Availability

Motif 2.1 and later.

Description

`XmComboBoxSetItem`() selects the first occurrence of the specified *item* in the ComboBox list, and makes the selection the first visible item in the list. If the *item* is found within the list, the value is also inserted into the ComboBox text field. Otherwise, a warning message is displayed.

Usage

`XmComboBoxSetItem`() is a convenience routine that allows you to select an *item* in the ComboBox. In order to use this routine, a compound string must be created for the *item*. No ComboBox selection callbacks are invoked as a result of calling this procedure. The routine internally calls `XmListSelectPos`() on the list component of the ComboBox, after performing a linear search through the XmNitems of the list: the *item* parameter is used only for the search and is not directly used as the newly selected item. It is the responsibility of the programmer to reclaim any allocated memory for the compound string item by calling `XmStringFree`() at an appropriate time.

See Also

`XmComboBoxAddItem`(1), `XmComboBoxDeletePos`(1), `XmComboBoxSelectItem`(1), `XmComboBoxUpdate`(1), `XmComboBox`(2).

Name

XmComboBoxUpdate – update the ComboBox list after changes to component widgets.

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxUpdate (Widget *widget*)

Inputs

widget Specifies the ComboBox widget.

Availability

Motif 2.0 and later.

Description

`XmComboBoxUpdate`() updates the ComboBox to reflect the state of component child widgets. This may be required where the programmer has directly modified the contents or resources of the ComboBox list component rather than through resources and functions of the ComboBox itself.

Usage

`XmComboBoxUpdate`() is a convenience routine that synchronizes the internal state of the ComboBox with that of the component list and text field. In particular, the value of XmNselectedPosition is reset to the value taken from the internal list. In addition, if the text field is unchanged, the XmNitems and XmNitemCount resources of the list are queried and used in conjunction with the recalculated XmNselectedPosition to reset the ComboBox selected item.

This routine should be called, for example, when the component list is directly manipulated to change the selected item without notifying the ComboBox directly.

See Also

`XmComboBoxAddItem`(1), `XmComboBoxSelectItem`(1), `XmComboBoxSetItem`(1), `XmComboBoxDeletePos`(1), `XmComboBox`(2).

Name

XmCommandAppendValue – append a compound string to the command.

Synopsis

#include <Xm/Command.h>

void XmCommandAppendValue (Widget *widget*, XmString *command*)

Inputs

widget	Specifies the Command widget.
command	Specifies the string that is appended.

Description

`XmCommandAppendValue`() appends the specified *command* to the end of the string that is displayed on the command line of the specified Command widget.

Usage

`XmCommandAppendValue`() is a convenience routine that changes the value of the XmNcommand resource of the Command widget. In order to use this routine, a compound string must be created for the *command*. The widget internally copies *command*, and it is the responsibility of the programmer to reclaim any allocated memory for the compound string at an appropriate time.

See Also

`XmCommandSetValue`(1), `XmCommand`(2).

Name

XmCommandError – display an error message in a Command widget.

Synopsis

#include <Xm/Command.h>

void XmCommandError (Widget *widget*, XmString *error*)

Inputs

widget	Specifies the Command widget.
error	Specifies the error message to be displayed.

Description

`XmCommandError`() displays an error message in the history region of the specified Command *widget*. The *error* string remains displayed until the next command takes effect.

Usage

`XmCommandError`() displays the *error* message as one of the items in the XmNhistoryItems list. When the next command is entered, the *error* message is deleted from the list. In order to use this routine, a compound string must be created for the *error* item. The *widget* internally copies *error*, and it is the responsibility of the programmer to reclaim any allocated memory for the compound string at an appropriate time.

See Also

`XmCommand`(2).

Name

XmCommandGetChild – get the specified child of a Command widget.

Synopsis

#include <Xm/Command.h>

Widget XmCommandGetChild (Widget *widget*, unsigned char *child*)

Inputs

widget Specifies the Command widget.
child Specifies a type of child of the Command widget.

Returns

The widget ID of the specified child of the Command widget.

Availability

As of Motif 2.0, the abstract child fetch routines in the toolkit are generally considered deprecated. Although `XmCommandGetChild`() continues to work, you should prefer `XtNameToWidget`() to access children of the XmCommand component.

Description

XmCommandGetChild() returns the widget ID of the specified child of the Command widget.

Usage

The *child* XmDIALOG_COMMAND_TEXT specifies the command text entry area, XmDIALOG_PROMPT_LABEL specifies the prompt label for the command line, XmDIALOG_HISTORY_LIST specifies the command history list, and XmDIALOG_WORK_AREA specifies any work area child that has been added to the Command widget. For more information on the different children of the Command widget, see the manual page in Section 2, *Motif and Xt Widget Classes*.

Structures

The possible values for child are:

XmDIALOG_COMMAND_TEXT	XmDIALOG_HISTORY_LIST
XmDIALOG_PROMPT_LABEL	XmDIALOG_WORK_AREA

Widget Hierarchy

The following names are associated with the Command children:

"Selection"	XmDIALOG_PROMPT_LABEL
"Text"	XmDIALOG_COMMAND_TEXT
"ItemsList"[1]	XmDIALOG_HISTORY_LIST

See Also

XmCommand(2).

1.The List is not a direct descendant of the Command widget, but of an intermediary ScrolledList. Therefore if fetching the widget via XtNameToWidget(), you should use the value "*ItemsList".

Name

XmCommandSetValue – replace the command string.

Synopsis

#include <Xm/Command.h>

void XmCommandSetValue (Widget *widget*, XmString *command*)

Inputs

widget	Specifies the Command widget.
command	Specifies the string that is displayed.

Description

`XmCommandSetValue`() replaces the currently displayed command-line text of the specified *Command* widget with the string specified by *command*. Specifying a zero-length string clears the command line.

Usage

`XmCommandSetValue`() is a convenience routine that changes the value of the XmNcommand resource of the Command widget. In order to use this routine, a compound string must be created for the *command*. The *widget* internally copies command, and it is the responsibility of the programmer to reclaim any allocated memory for the compound string at an appropriate time.

See Also

`XmCommandAppendValue`(1), `XmCommand`(2).

Name

XmContainerCopy – copy the Container primary selection onto the clipboard.

Synopsis

#include <Xm/Container.h>

Boolean XmContainerCopy (Widget *container*, Time *timestamp*)

Inputs

container	Specifies a Container widget.
timestamp	Specifies the server time at which to modify the selection.

Returns

True if the Container selection is transferable to the clipboard, False otherwise.

Availability

Motif 2.0 and later.

Description

`XmContainerCopy`() copies the primary selection from a Container widget to the clipboard. The primary selection of a Container widget consists of a set of selected Container items.

If there are no selected Container items within container, or if the container widget does not own the primary selection, or if container cannot gain ownership of the clipboard selection, the function returns False.

Usage

`XmContainerCopy`() is a convenience routine that copies a Container primary selection to the clipboard. The procedures identified by the XmNconvertCallback list of the Container are called to transfer the selection: the selection member of the XmConvertCallbackStruct passed to callbacks has the value CLIPBOARD, and the parm member is set to XmCOPY. See `XmTransfer`(1) for specific details of the XmConvertCallbackStruct, and of the Uniform Transfer Model (UTM) in general.

See Also

`XmContainerCut`(1), `XmContainerCopyLink`(1), `XmContainerGetItemChildren`(1), `XmContainerPaste`(1), `XmContainerPasteLink`(1), `XmContainerRelayout`(1), `XmContainerReorder`(1), `XmTransfer`(1), `XmContainer`(2).

Name

XmContainerCopyLink – copy links to the Container primary selection onto the clipboard.

Synopsis

#include <Xm/Container.h>

Boolean XmContainerCopyLink (Widget *container*, Time *timestamp*)

Inputs

container Specifies a Container widget.
timestamp Specifies a time stamp at which to modify the selection.

Returns

True if the Container selection is transferable to the clipboard, False otherwise.

Availability

Motif 2.0 and later.

Description

`XmContainerCopyLink`() copies links to the primary selection of a Container widget onto the clipboard. The primary selection of a Container widget consists of a set of selected Container items.

If there are no selected Container items within container, or if the container widget does not own the primary selection, or if container cannot gain ownership of the clipboard selection, the function returns False.

Usage

`XmContainerCopyLink`() is a convenience routine that copies links to a Container primary selection to the clipboard. The procedures identified by the XmNconvertCallback list of the Container are called, possibly many times: the selection member of the XmConvertCallbackStruct passed to callbacks has the value CLIPBOARD, and the parm member is set to XmLINK. See `XmTransfer`(1) for specific details of the XmConvertCallbackStruct, and of the Uniform Transfer Model (UTM) in general.

See Also

`XmContainerCut`(1), `XmContainerCopy`(1), `XmContainerGetItemChildren`(1), `XmContainerPaste`(1), `XmContainerPasteLink`(1), `XmContainerRelayout`(1), `XmContainerReorder`(1), `XmTransfer`(1), `XmContainer`(2).

Name

XmContainerCut – cuts the Container primary selection onto the clipboard.

Synopsis

#include <Xm/Container.h>

Boolean XmContainerCut (Widget *container*, Time *timestamp*)

Inputs

container Specifies a Container widget.
timestamp Specifies the time at which to modify the selection.

Returns

True if the Container selection is transferable to the clipboard, False otherwise.

Availability

Motif 2.0 and later.

Description

`XmContainerCut`() cuts the primary selection from a Container widget onto the clipboard. The primary selection of a Container widget consists of a set of selected Container items.

If there are no selected Container items within container, or if the container widget does not own the primary selection, or if container cannot gain ownership of the clipboard selection, the function returns False.

Usage

`XmContainerCut`() is a convenience routine that moves a Container primary selection onto the clipboard, then removes the primary selection. The procedures identified by the XmNconvertCallback list of the Container are invoked to move the selection to the clipboard: the selection member of the XmConvertCallback-Struct passed to callbacks has the value CLIPBOARD, and the parm member is set to XmMOVE. Thereafter, if the data was transferred, the convert callbacks are invoked again to delete the primary selection: the selection member is set to CLIPBOARD, and the target member is set to DELETE. See `XmTransfer`(1) for specific details of the XmConvertCallbackStruct, and of the Uniform Transfer Model (UTM) in general.

See Also

`XmContainerCopy`(1), `XmContainerCopyLink`(1), `XmContainerGetItemChildren`(1), `XmContainerPaste`(1), `XmContainerPasteLink`(1), `XmContainerRelayout`(1), `XmContainerReorder`(1), `XmTransfer`(1), `XmContainer`(2).

Name

XmContainerGetItemChildren – find the children of a Container item.

Synopsis

#include <Xm/Container.h>

int XmContainerGetItemChildren (Widget *container*, Widget *item*, WidgetList **item_children*)

Inputs

container	Specifies a Container widget.
item	A child of the Container which holds the XmQTcontainerItem trait.

Outputs

item_children	The list of logical children associated with the *item*.

Returns

The number of logical children within the *item_children* list.

Availability

Motif 2.0 and later.

Description

`XmContainerGetItemChildren`() constructs a list of Container items which have item as a logical parent. *item* must hold the XmQTcontainerItem trait: an IconGadget child of container, for example. A widget is a logical child of item if the value of its constraint resource XmNentryParent is equal to item. *container* is the Container widget which has *item* as a child, and the list of logical children of item is placed in *item_children*. The function returns the number of logical children found.

Usage

`XmContainerGetItemChildren`() is a convenience routine which allocates a WidgetList to contain the set of all Container children whose XmNentryParent resource matches that of a designated *item*.

If *item* is NULL, or if *item* is not a child of *container*, or if *item* has no logical children, the *item_children* parameter is not set and the function returns 0.

Storage for the returned WidgetList is allocated by the function, and it is the responsibility of the programmer to free the memory using `XtFree`() at an appropriate point.

See Also

XmContainerCut(1), XmContainerCopy(1), XmContainerCopyLink(1), XmContainerPaste(1), XmContainerPasteLink(1), XmContainerRelayout(1), XmContainerReorder(1), XmContainer(2).

Name

XmContainerPaste – pastes the clipboard selection into a Container.

Synopsis

#include <Xm/Container.h>

Boolean XmContainerPaste (Widget *container*)

Inputs

container Specifies a Container widget.

Returns

True if the clipboard selection is transferable to the Container, False otherwise.

Availability

Motif 2.0 and later.

Description

`XmContainerPaste`() initiates data transfer of the clipboard primary selection to the *container* widget.

If data is transferred from the clipboard, the function returns True, otherwise False.

Usage

`XmContainerPaste`() is a convenience routine that initiates copying of the clipboard primary selection to a Container widget. The procedures identified by the XmNdestinationCallback list of the Container are called: the selection member of the XmDestinationCallbackStruct passed to callbacks has the value CLIPBOARD, and the operation member is set to XmCOPY.
`XmContainerPaste`() does not transfer data itself: it is the responsibility of the programmer to supply a destination callback which will copy the clipboard selection into the Container. See `XmTransfer`(1) for specific details of the XmDestinationCallbackStruct, and of the Uniform Transfer Model (UTM) in general.

See Also

`XmContainerCut`(1), `XmContainerCopy`(1), `XmContainerCopyLink`(1), `XmContainerGetItemChildren`(1), `XmContainerPasteLink`(1), `XmContainerRelayout`(1), `XmContainerReorder`(1), `XmTransfer`(1), `XmContainer`(2).

Name

XmContainerPasteLink – copies links from the clipboard selection into a Container.

Synopsis

#include <Xm/Container.h>

Boolean XmContainerPasteLink (Widget *container*)

Inputs

container Specifies a Container widget.

Returns

True if the clipboard selection is transferable to the Container, False otherwise.

Availability

Motif 2.0 and later.

Description

`XmContainerPasteLink`() initiates data transfer of the clipboard primary selection to the *container* widget.

If data is transferred from the clipboard, the function returns True, otherwise False.

Usage

`XmContainerPasteLink`() is a convenience routine that initiates copying links from the clipboard primary selection into a Container widget. The procedures identified by the XmNdestinationCallback list of the Container are called: the selection member of the XmDestinationCallbackStruct passed to callbacks has the value CLIPBOARD, and the operation member is set to XmLINK. `XmContainerPasteLink`() does not transfer data itself: it is the responsibility of the programmer to supply a destination callback which will link the clipboard selection into the Container. See `XmTransfer`(1) for specific details of the XmConvertCallbackStruct, and of the Uniform Transfer Model (UTM) in general.

See Also

`XmContainerCut`(1), `XmContainerCopy`(1), `XmContainerCopyLink`(1), `XmContainerGetItemChildren`(1), `XmContainerPaste`(1), `XmContainerRelayout`(1), `XmContainerReorder`(1), `XmTransfer`(1), `XmContainer`(2).

Name

XmContainerRelayout – force relayout of a Container widget.

Synopsis

#include <Xm/Container.h>

void XmContainerRelayout (Widget *container*)

Inputs

container Specifies a Container widget.

Availability

Motif 2.0 and later.

Description

`XmContainerRelayout`() forces the *container* widget to recalculate the layout of all Container items.

Usage

`XmContainerRelayout`() is a convenience routine that recalculates the grid layout of a Container. The function has no effect if the widget is not realized, if XmNlayoutType is not XmSPATIAL, or if XmNspatialStyle is XmNONE.

The function does not cause geometry management effects when performing the relayout, although the Container window is completely cleared and redrawn if the widget is realized.

`XmContainerRelayout`() utilizes the place_item method of the Container widget class. If this is NULL in any derived class, `XmContainerRelayout`() will have no effect upon the layout of Container items.

See Also

`XmContainerCut`(1), `XmContainerCopy`(1), `XmContainerCopyLink`(1), `XmContainerGetItemChildren`(1), `XmContainerPaste`(1), `XmContainerPasteLink`(1), `XmContainerReorder`(1), `XmContainer`(2).

Name

XmContainerReorder – reorder children of a Container.

Synopsis

#include <Xm/Container.h>

void XmContainerReorder (Widget *container*, WidgetList *item_list*, int *item_count*)

Inputs

container	Specifies a Container widget.
item_list	Specifies a list of Container child widgets.
item_count	Specifies the number of widgets in item_list.

Availability

Motif 2.0 and later.

Description

`XmContainerReorder`() reorders an item_list set of items of a Container. item_count is the number of items within the item_list array.

Usage

`XmContainerReorder`() is a convenience routine that reorders Container items according to the value of the XmNpositionIndex constraint resource of each item, using a quicksort algorithm. If the XmNlayoutType is XmOUTLINE or XmDETAIL, the Container will subsequently relayout all the items within the widget.

Neither relayout nor reorder is performed if *item_count* is less than or equal to 1; there is no error checking performed on *item_list* to compare it with NULL, or to ensure that it matches the number of items specified by *item_count*.

See Also

`XmContainerCut`(1), `XmContainerCopy`(1), `XmContainerCopyLink`(1), `XmContainerGetItemChildren`(1), `XmContainerPaste`(1), `XmContainerPasteLink`(1), `XmContainerRelayout`(1), `XmContainer`(1).

Name

XmConvertStringToUnits – convert a string to an integer, optionally translating the units.

Synopsis

```
int XmConvertStringToUnits (    Screen      *screen,
                                String      spec,
                                int         orientation,
                                int         unit_type,
                                XtEnum      *error_return)
```

Inputs

screen	Specifies a pointer to the screen structure.
spec	Specifies a value to be converted.
orientation	Specifies whether to use horizontal or vertical screen resolution. Pass either XmHORIZONTAL or XmVERTICAL.
unit_type	The units required for the result.

Outputs

error_return	Returns the error status of the conversion.

Returns

The converted value.

Availability

Motif 2.0 and later.

Description

`XmConvertStringToUnits`() converts a string *spec* into an integer. The conversion of *spec* is into the units specified by *unit_type*. Resolution for the conversion is determined from the *screen*, and *orientation* determines whether the horizontal or vertical screen resolution is used. The converted value is returned by the function. The *error_return* parameter is set by the function to indicate any error in the conversion process.

Usage

`XmConvertStringToUnits`() converts a string into an integer, translating the units of the original string into those specified by unit_type. If the *screen* is NULL, or if *orientation* is an invalid value, or if an invalid *unit_type* is supplied, or if the string *spec* is not parsable, the function returns 0 (zero), and *error_return* is set True. Otherwise, *error_return* is set False, and the function returns the converted value.

The string *spec* is assumed to be in the following format:

<float> <unit>

where <float> is a floating point number. The <unit> specification is optional: if omitted, the default unit of XmPIXELS is used. Otherwise, <unit> is one of the following strings:

pix	pixel	pixels
in	inch	inches
cm	centimeter	centimeters
mm	millimeter	millimeters
pt	point	points
fu	font_unit	font_units

Structures

The possible values for unit_type are:

XmPIXELS	XmCENTIMETERS	XmMILLIMETERS
Xm100TH_MILLIMETERS	XmINCHES	
Xm1000TH_INCHES		
XmPOINTS	Xm100TH_POINTS	
XmFONT_UNITS		
Xm100TH_FONT_UNITS		

Example

The following are valid string specifications:

```
3.1415926 pix
-3.1 pt
6.3
0.3 font_units
1
```

See Also

`XmConvertUnits`(1), `XmScreen`(2).

Name

XmConvertUnits – convert a value to a specified unit type.

Synopsis

```
int XmConvertUnits (    Widget    widget,
                        int       orientation,
                        int       from_unit_type,
                        int       from_value,
                        int       to_unit_type)
```

Inputs

widget	Specifies the widget for which to convert the data.
orientation	Specifies the screen orientation that is used in the conversion. Pass either XmHORIZONTAL or XmVERTICAL.
from_unit_type	Specifies the unit type of the value that is being converted.
from_value	Specifies the value that is being converted.
to_unit_type	Specifies the new unit type of the value.

Returns

The converted value or 0 (zero) if the input parameters are not specified correctly.

Description

`XmConvertUnits`() converts the value specified in *from_value* into the equivalent value in a different unit of measurement. This function returns the resulting value if successful; it returns 0 (zero) if widget is NULL or if incorrect values are supplied for orientation or conversion unit arguments. *orientation* matters only when conversion values are font units, which are measured differently in the horizontal and vertical dimensions.

Usage

`XmConvertUnits`() allows an application to manipulate resolution-independent values. XmPIXELS specifies a normal pixel value, Xm100TH_MILLIMETERS specifies a value in terms of 1/100 of a millimeter, Xm1000TH_INCHES specifies a value in terms of 1/1000 of an inch, Xm100TH_POINTS specifies a value in terms of 1/100 of a point (1/72 of an inch), and Xm100TH_FONT_UNITS specifies a value in terms of 1/100 of a font unit. A font unit has horizontal and vertical components which are specified by the XmScreen resources XmNhorizontalFontUnit and XmNverticalFontUnit.

Structures

The possible values for from_unit_type and to_unit_type are:

XmPIXELS	XmCENTIMETERS
XmMILLIMETERS	Xm100TH_MILLIMETERS

XmINCHES	Xm1000TH_INCHES
XmPOINTS	Xm100TH_POINTS
XmFONT_UNITS	Xm100TH_FONT_UNITS

The values XmPOINTS, XmINCHES, XmCENTIMETERS, XmFONT_UNITS, and XmMILLIMETERS are available in Motif 2.0 and later.

See Also

`XmSetFontUnits`(1), `XmScreen`(2).

Name

XmCreate*Object* – create an instance of a particular widget class or compound object.

Synopsis

Simple Widgets

#include <Xm/ArrowB.h>
Widget XmCreateArrowButton (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/ArrowBG.h>
Widget XmCreateArrowButtonGadget (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/BulletinB.h>
Widget XmCreateBulletinBoard (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/CascadeB.h>
Widget XmCreateCascadeButton (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/CascadeBG.h>
Widget XmCreateCascadeButtonGadget (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Command.h>
Widget XmCreateCommand (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/ComboBox.h>
Widget XmCreateComboBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateDropDownComboBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateDropDownList (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Container.h>
Widget XmCreateContainer (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/DataF.h>
Widget XmCreateDataField (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/DialogS.h>
Widget XmCreateDialogShell (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/DragIcon.h>
Widget XmCreateDragIcon (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/DrawingA.h>
Widget XmCreateDrawingArea (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/DrawnB.h>
Widget XmCreateDrawnButton (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/DropDown.h>
Widget XmCreateDropDown (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/FileSB.h>
Widget XmCreateFileSelectionBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Form.h>
Widget XmCreateForm (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Frame.h>
Widget XmCreateFrame (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/GrabShell.h>
Widget XmCreateGrabShell (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/IconButton.h>
Widget XmCreateIconButton (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/IconG.h>
Widget XmCreateIconGadget (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Label.h>
Widget XmCreateLabel (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/LabelG.h>
Widget XmCreateLabelGadget (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/List.h>
Widget XmCreateList (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/MainW.h>
Widget XmCreateMainWindow (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/MenuShell.h>
Widget XmCreateMenuShell (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/MessageB.h>
Widget XmCreateMessageBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/MultiList.h>
Widget XmCreateMultiList (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/Notebook.h>
Widget XmCreateNotebook (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Outline.h>
Widget XmCreateOutline (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/PanedW.h>
Widget XmCreatePanedWindow (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/PushB.h>
Widget XmCreatePushButton (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/PushBG.h>
Widget XmCreatePushButtonGadget (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/RowColumn.h>
Widget XmCreateRowColumn (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

Widget XmCreateRadioBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateWorkArea (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Scale.h>
Widget XmCreateScale (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/ScrollBar.h>
Widget XmCreateScrollBar (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/ScrolledW.h>
Widget XmCreateScrolledWindow (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/SelectioB.h>
Widget XmCreateSelectionBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Separator.h>
Widget XmCreateSeparator (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/SeparatoG.h>
Widget XmCreateSeparatorGadget (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/SSpinB.h>
Widget XmCreateSimpleSpinBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/SpinB.h>
Widget XmCreateSpinBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Text.h>
Widget XmCreateText (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/TextF.h>
Widget XmCreateTextField (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/ToggleB.h>

Widget XmCreateToggleButton (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/ToggleBG.h>
Widget XmCreateToggleButtonGadget (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Tree.h>
Widget XmCreateTree (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

Dialog Objects

#include <Xm/BulletinB.h>
Widget XmCreateBulletinBoardDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <ButtonBox.h>
Widget XmCreateButtonBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <ColorS.h>
Widget XmCreateColorSelector (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Column.h>
Widget XmCreateColumn (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/FileSB.h>
Widget XmCreateFileSelectionDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/FontS.h>
Widget XmCreateFontSelector (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/Form.h>
Widget XmCreateFormDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/IconBox.h>
Widget XmCreateIconBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

#include <Xm/MessageB.h>
Widget XmCreateErrorDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

Widget XmCreateInformationDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateMessageDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateQuestionDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateTemplateDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateWarningDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateWorkingDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/SelectioB.h>
Widget XmCreatePromptDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateSelectionDialog (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/TabStack.h>
Widget XmCreateDialogShell (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Command.h>
Widget XmCreateTabStack (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argv*)

Menu Objects

#include <Xm/RowColumn.h>
Widget XmCreateMenuBar (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateOptionMenu (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreatePopupMenu (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreatePulldownMenu (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

Simple Menu Objects

#include <Xm/Xm.h>
Widget XmCreateSimpleCheckBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateSimpleMenuBar (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

Widget XmCreateSimpleOptionMenu (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateSimplePopupMenu (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateSimplePulldownMenu (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)
Widget XmCreateSimpleRadioBox (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

Scrolled Objects

#include <Xm/List.h>
Widget XmCreateScrolledList (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

#include <Xm/Text.h>
Widget XmCreateScrolledText (Widget *parent*, char **name*, ArgList *argv*, Cardinal *argc*)

Inputs

parent	Specifies the widget ID of the parent of the new widget.
name	Specifies the string name of the new widget for resource lookup.
argv	Specifies the resource name/value pairs used in creating the widget.
argc	Specifies the number of name/value pairs in *argv*.

Returns

The simple widget creation routines return the widget ID of the widget that is created. The dialog creation routines return the widget ID of the widget that is created as a child of the DialogShell. The menu creation routines return the widget ID of the RowColumn widget that is created. The scrolled object creation routines return the widget ID of the List or Text widget.

Availability

`XmCreateDragIcon`() and `XmCreateTemplateDialog`() are only available in Motif 1.2 and later.

`XmCreateGrabShell`(), `XmCreateIconGadget`(), `XmCreateComboBox`(), `XmCreateDropDownComboBox`(), `XmCreateDropDownList`(), `XmCreateNotebook`(), `XmCreateContainer`(), and `XmCreateSpinBox`() are available from Motif 2.0 onwards.

`XmCreateSimpleSpinBox`() is available from Motif 2.1 and onwards.

Description

The `XmCreate`*() routines are convenience routines for creating an instance of a particular widget class or a particular compound object. Each creation routine

takes the same four arguments: the *parent*'s widget ID, the *name* of the new widget, a list of resource name/value pairs, and the number of name/value pairs.

The simple creation routines create a single widget with the default resource settings for the widget class, except for `XmCreateRadioBox`() and `XmCreateWorkArea`(), which create specially configured RowColumn widgets.

The dialog creation routines are convenience routines for creating a particular unmanaged widget as a child of a DialogShell. The *parent* argument specifies the parent of the DialogShell and *name* specifies the string name of the particular widget that is created. The name of the DialogShell is the string that results from appending "_popup" to the *name* of the widget. The routines return the widget ID of the widget that is created as the child of the DialogShell.

The menu creation routines are convenience routines for creating particular types of menu objects. Each routine creates a RowColumn widget with specific resource settings that configure the widget to operate as the particular type of menu. `XmCreatePopupMenu`() and `XmCreatePulldownMenu`() create the RowColumn widget as the child of a MenuShell.

Except for `XmCreateSimpleSpinBox`(), the simple menu creation routines are convenience routines for creating particular configurations of RowColumn widgets and their children. For example, `XmCreateSimpleCheckBox`() creates a CheckBox with ToggleButtonGadgets as its children.

`XmCreateScrolledList`() and `XmCreateScrolledText`() are convenience routines that create a List or Text widget as the child of a ScrolledWindow. The *parent* argument specifies the parent of the ScrolledWindow and *name* specifies the string name of the List or Text widget. The *name* of the ScrolledWindow is the string that results from appending "SW" to the *name* of the widget. The routines return the widget ID of the List or Text widget.

Usage

Each widget or compound object that can be created with an `XmCreate`*() routine can also be created using `XtCreateWidget`(). The simple Motif creation routines are simply veneers to `XtCreateWidget`(). The rest of the Motif creation routines create multiple widgets and/or set specific widget resources. In order to use `XtCreateWidget`() to create these objects, you need to have a complete understanding of the compound object that you are trying to create. For more information on each widget and compound object that can be created, see the appropriate manual page in Section 2, *Motif and Xt Widget Classes*.

See Also

XmArrowButtonGadget(2), XmArrowButton(2), XmBulletinBoardDialog(2), XmBulletinBoard(2), XmCascadeButtonGadget(2), XmCascadeButton(2), XmCheckBox(2), XmComboBox(2), XmCommand(2), XmCommandDialog(2), XmContainer(2), XmDialogShell(2), XmDragIcon(2), XmDrawingArea(2), XmDrawnButton(2), XmErrorDialog(2), XmFileSelectionBox(2), XmFileSelectionDialog(2), XmFormDialog(2), XmForm(2), XmFrame(2), XmGrabShell(2), XmIconGadget(2), XmInformationDialog(2), XmLabelGadget(2), XmLabel(2), XmList(2), XmMainWindow(2), XmMenuBar(2), XmMenuShell(2), XmMessageBox(2), XmMessageDialog(2), XmNotebook(2), XmOptionMenu(2), XmPanedWindow(2), XmPopupMenu(2), XmPromptDialog(2), XmPulldownMenu(2), XmPushButtonGadget(2) XmPushButton(2), XmQuestionDialog(2), XmRadioBox(2), XmRowColumn(2), XmScale(2), XmScrollBar(2), XmScrolledList(2), XmScrolledText(2), XmScrolledWindow(2), XmSelectionBox(2), XmSelectionDialog(2), XmSeparatorGadget(2), XmSeparator(2), XmSpinBox(2), XmSimpleSpinBox(2), XmTemplateDialog(2), XmTextField(2), XmText(2), XmToggleButtonGadget(2), XmToggleButton(2), XmWarningDialog(2), XmWorkingDialog(2).

Name

XmCvtByteStreamToXmString – convert a byte stream to a compound string.

Synopsis

XmString XmCvtByteStreamToXmString (unsigned char **property*)

Inputs

property Specifies a byte stream.

Returns

An allocated compound string.

Availability

Motif 2.0 and later.

Description

`XmCvtByteStreamToXmString`() converts a stream of bytes to a compound string. The function is typically used by the destination of a data transfer operation.

Usage

`XmCvtByteStreamToXmString`() converts a compound string in byte stream format into an XmString. The function allocates storage for the returned compound string, and it is the responsibility of the programmer to free the allocated memory by calling `XmStringFree`() at an appropriate point.

See Also

`XmCvtXmStringToByteStream`(1), `XmStringFree`(1),

Name

XmCvtCTToXmString – convert compound text to a compound string.

Synopsis

XmString XmCvtCTToXmString (char **text*)

Inputs

text Specifies the compound text that is to be converted.

Returns

The converted compound string.

Description

`XmCvtCTToXmString`() converts the specified *text* string from compound text format, which is an X Consortium Standard defined in *Compound Text Encoding*, to a Motif compound string. The routine assumes that the compound text is NULL-terminated and NULLs within the compound text are handled correctly. If text contains horizontal tabulation (HT) control characters, the result is undefined. `XmCvtCTToXmString`() allocates storage for the converted compound string. The application is responsible for freeing this storage using XmStringFree().

Usage

Compound text is an encoding that is designed to represent text from any locale. Compound text strings identify their encoding using embedded escape sequences. The compound text representation was standardized for X11R4 for use as a text interchange format for interclient communication. An application must call `XtAppInitialize`() before calling `XmCvtCTToXmString`(). The conversion of compound text to compound strings is implementation dependent. `XmCvtCTToXmString`() is the complement of `XmCvtXmStringToCT`().

See Also

`XmCvtXmStringToCT`(1).

Name

XmCvtStringToUnitType – convert a string to a unit-type value.

Synopsis

```
void XmCvtStringToUnitType (  XrmValuePtr   args,
                              Cardinal      *num_args,
                              XrmValue      *from_val,
                              XrmValue      *to_val)
```

Inputs

args Specifies additional XrmValue arguments that are need to perform the conversion.

num_args Specifies the number of items in args.

from_val Specifies value to convert.

Outputs

to_val Returns the converted value.

Availability

In Motif 1.2, `XmCvtStringToUnitType`() is obsolete. It has been superseded by a new resource converter that uses the RepType facility.

Description

`XmCvtStringToUnitType`() converts the string specified in *from_val* to one of the unit-type values: XmPIXELS, Xm100TH_MILLIMETERS, Xm1000TH_INCHES, Xm100TH_POINTS, or Xm100TH_FONT_UNITS. This value is returned in *to_val*.

Usage

`XmCvtStringToUnitType`() should not be called directly; it should be installed as a resource converter using the R3 routine `XtAddConverter`(). The routine only needs to be installed if the XmNunitType resource for a widget is being set in a resource file. In this case, `XmCvtStringToUnitType`() must be installed with `XtAddConverter`() before the widget is created. Use the following call to `XtAddConverter`() to install the converter:

```
XtAddConverter (XmRString, XmRUnitType, XmCvtStringToUnitType,
NULL, 0);
```

In Motif 1.2, the use of `XmCvtStringToUnitType`() as a resource converter is obsolete. A new resource converter that uses the RepType facility has replaced the routine.

See Also

`XmGadget`*(2)*, `XmManager`*(2)*, `XmPrimitive`*(2)*.

Name

XmCvtTextPropertyToXmStringTable – convert an XTextProperty to a Compound String Table.

Synopsis

```
#include <Xm/TxtPropCv.h>

int XmCvtTextPropertyToXmStringTable (  Display        *display,
                                        XTextProperty  *text_prop,
                                        XmStringTable
*str_table_return,
                                        int            *count_return)
```

Inputs

display	Specifies the connection to the X server.
text_prop	Specifies a pointer to an XTextProperty structure.

Outputs

str_table_return	The XmStringTable array converted from text_prop.
count_return	The number of XmStrings in str_table_return.

Returns

Success if the conversion succeeded, XLocaleNotSupported if the current locale is unsupported, XConverterNotFound if no converter is available in the current locale.

Availability

Motif 2.0 and later.

Description

`XmCvtTextPropertyToXmStringTable`() converts the data specified within *text_prop* into an array of XmStrings, returned through *str_table_return*. The number of XmStrings in the array is returned in *count_return*.

Usage

The `XmCvtTextPropertyToXmStringTable`() function converts data specified within an XTextProperty structure into an XmStringTable. The data to be converted is the value member of *text_prop*, where value is an array of bytes, consisting of a series of concatenated items, each NULL separated. The number of such items is given by the nitems member of *text_prop*. The last item is terminated by two NULL bytes. The interpretation of each item depends upon the encoding member of *text_prop*.

If the encoding member of *text_prop* is COMPOUND_TEXT, the data is converted using the function `XmCvtCTToXmString`(). If encoding is COMPOUND_STRING, the data is converted using the function `XmCvt-`

ByteStreamToXmString(). Conversion requires that a converter has been registered for the current locale, otherwise the function returns XConverterNotFound. If encoding is XA_STRING, each returned XmString is converted through XmStringGenerate() with a tag of "ISO8859-1" and a text type of XmCHARSET_TEXT. If encoding is that of the current locale, each returned XmString is converted through XmStringGenerate() with a tag of _MOTIF_DEFAULT_LOCALE, and a text type of XmMULTIBYTE_TEXT. For other values of encoding, the function returns XLocaleNotSupported.

XmCvtTextPropertyToXmStringTable() returns allocated storage, and it is the responsibility of the programmer to free the utilized memory at an appropriate point by freeing each element of the array through XmStringFree(), and subsequently the array itself through XtFree().

Structures

The XTextProperty structure is defined in <X11/Xutil.h> as follows:

```
typedef struct {
    unsigned char   *value;     /* same as Property routines        */
    Atom            encoding;   /* the property type                */
    int             format;     /* property data format: 8, 16, or 32. */
    unsigned long   nitems;     /* number of data items in value    */
} XTextProperty;
```

See Also

XmCvtByteStreamToXmString(1), XmCvtCTToXmString(1), XmStringFree(1), XmStringGenerate(1).

Name

XmCvtXmStringTableToTextProperty – convert an XmStringTable to an XTextProperty.

Synopsis

```
#include <Xm/TxtPropCv.h>

int XmCvtXmStringTableToTextProperty (
        Display                 *display,
        XmStringTable           string_table,
        int                     count,
        XmICCEncodingStyle      style,
        XTextProperty           *prop_return)
```

Inputs

display	Specifies the connection to the X server.
string_table	Specifies an array of compound strings.
count	Specifies the number of compound strings in *string_table*.
style	Specifies the encoding style from which to convert *string_table*.

Outputs

prop_return	The XTextProperty structure converted from *string_table*.

Returns

Success if the conversion succeeded, XLocaleNotSupported if the current locale is unsupported.

Availability

Motif 2.0 and later.

Description

`XmCvtXmStringTableToTextProperty`() is the inverse function to `XmCvtTextPropertyToXmStringTable`(). It converts an array of compound strings, specified by *string_table*, into the elements of an XTextProperty structure. The number of compound strings within the *string_table* is given by *count*.

Usage

`XmCvtXmStringTableToTextProperty`() converts an XmStringTable into the elements of an XTextProperty structure. The encoding member contains an Atom representing the requested *style*. The value member contains a list of the converted items, each separated by NULL bytes, and terminated by two NULL bytes, the nitems member is the number of such items converted.

If *style* is XmSTYLE_COMPOUND_STRING, encoding is _MOTIF_COMPOUND_STRING, and value contains a list of XmStrings in byte stream format.

If *style* is XmSTYLE_COMPOUND_TEXT, encoding is COMPOUND_TEXT, and value contains compound text items.

If *style* is XmSTYLE_LOCALE, encoding is the Atom representing the encoding for the current locale. value contains items converted into the current locale.

If *style* is XmSTYLE_STRING, encoding is STRING, and value contains items converted into ISO8859-1 strings.

If *style* is XmSTYLE_TEXT, and all the XmStrings in *string_table* are convertible into the encoding for the current locale, the function behaves as though *style* is XmSTYLE_LOCALE. Otherwise, the function behaves as though *style* is XmSTYLE_COMPOUND_TEXT.

If *style* is XmSTYLE_STANDARD_ICC_TEXT, and all the XmStrings in *string_table* are convertible as though the *style* is XmSTYLE_STRING, the function behaves as though *style* is indeed XmSTYLE_STRING. Otherwise, the function behaves as though *style* is XmSTYLE_COMPOUND_TEXT.

`XmCvtXmStringTableToTextProperty`() returns XLocaleNotSupported if the conversion cannot be performed within the current locale, or if *style* is not valid. Otherwise, the function returns Success.

Structures

The XTextProperty structure is defined in <X11/Xutil.h> as follows:

```
typedef struct {
    unsigned char   *value;     /* same as Property routines        */
    Atom            encoding;   /* property type                    */
    int             format;     /* property data format: 8, 16, or 32 */
    unsigned long   nitems;     /* number of data items in value    */
} XTextProperty;
```

The possible values of the XmICCEncodingStyle parameter style are:

XmSTYLE_COMPOUND_STRING
XmSTYLE_COMPOUND_TEXT
XmSTYLE_LOCALE
XmSTYLE_STANDARD_ICC_TEXT
XmSTYLE_STRING
XmSTYLE_TEXT

See Also

`XmCvtByteStreamToXmString`(1), `XmCvtCTToXmString`(1), `XmCvtTextPropertyToStringTable`(1), `XmStringFree`(1), `XmStringGenerate`(1).

Name

XmCvtXmStringToByteStream – convert a compound string to byte stream format.

Synopsis

unsigned int XmCvtXmStringToByteStream (XmString *string*, unsigned char **prop_return*)

Inputs

string Specifies the compound string that is to be converted.

Outputs

prop_return The converted compound string in byte stream format.

Returns

The number of bytes in the byte stream.

Availability

Motif 2.0 and later.

Description

`XmCvtXmStringToByteStream`() converts a compound string *string* into a stream of bytes, returning the number of bytes required for the conversion. The byte stream is returned in *prop_return*. The function is the inverse of `XmCvtByteStreamToXmString`().

Usage

`XmCvtXmStringToByteStream`() converts an XmString into byte stream format. If *prop_return* is not NULL, the function places into *prop_return* the converted string, and returns its length in bytes. If *prop_return* is NULL, the number of bytes is calculated and returned, but no conversion is performed.

`XmCvtXmStringToByteStream`() returns allocated storage in *prop_return*, and it is the responsibility of the programmer to free the utilized memory at an appropriate point by calling `XtFree`().

See Also

`XmCvtByteStreamToXmString`(1).

Name

XmCvtXmStringToCT – convert a compound string to compound text.

Synopsis

char * XmCvtXmStringToCT (XmString *string*)

Inputs

string Specifies the compound string that is to be converted.

Returns

The converted compound text string.

Description

`XmCvtXmStringToCT`() converts the specified Motif compound *string* to a string in X11 compound text format, which is described in the X Consortium Standard *Compound Text Encoding.*

Usage

Compound text is an encoding that is designed to represent text from any locale. Compound text strings identify their encoding using embedded escape sequences. The compound text representation was standardized for X11R4 for use as a text interchange format for interclient communication. `XmCvtXmStringToCT`() is the complement of `XmCvtCTToXmString`().

In Motif 1.2 and later, an application must not call `XmCvtXmStringToCT`() until after `XtAppInitialize`() is called, so that the locale is established correctly. The routine uses the font list tag of each compound string segment to select a compound text format for the segment. A mapping between font list tags and compound text encoding formats is stored in a registry.

If the compound string segment tag is associated with XmFONTLIST_DEFAULT_TAG in the registry, the converter calls `XmbTextListToTextProperty`() with the XCompoundTextStyle encoding style and uses the resulting compound text for the segment. If the compound string segment tag is mapped to a registered MIT charset, the routine creates the compound text using the charset as defined in the X Consortium Standard Compound Text Encoding. If the compound string segment tag is associated with a charset that is not XmFONTLIST_DEFAULT_TAG or a registered charset, the converter creates the compound text using the charset and the text as an "extended segment" with a variable number of octets per character. If the compound string segment tag is not mapped in the registry, the result depends upon the implementation.

See Also

`XmCvtCTToXmString`(1), `XmMapSegmentEncoding`(1),

`XmRegisterSegmentEncoding`(1).

Name

XmDeactivateProtocol – deactivate a protocol.

Synopsis

#include <Xm/Protocols.h>

void XmDeactivateProtocol (Widget *shell*, Atom *property*, Atom *protocol*)

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocol	Specifies the protocol atom.

Description

`XmDeactivateProtocol`() deactivates the specified *protocol* without removing it. If the shell is realized, `XmDeactivateProtocol`() updates its protocol handlers and the specified *property.* A protocol may be active or inactive. If *protocol* is active, the protocol atom is stored in *property*; if *protocol* is inactive, the protocol atom is not stored in property.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. `XmDeactivateProtocol`() allows a client to temporarily stop participating in the communication. The inverse routine is `XmActivateProtocol`().

See Also

`XmActivateProtocol`(1), `XmDeactivateWMProtocol`(1), `XmInternAtom`(1), `VendorShell`(2).

Name

XmDeactivateWMProtocol – deactivate the XA_WM_PROTOCOLS protocol.

Synopsis

#include <Xm/Protocols.h>

void XmDeactivateWMProtocol (Widget *shell*, Atom *protocol*)

Inputs

shell	Specifies the widget associated with the protocol property.
protocol	Specifies the protocol atom.

Description

`XmDeactivateWMProtocol`() is a convenience routine that calls `XmDeactivateProtocol`() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. `XmDeactivateWMProtocol`() allows a client to temporarily stop participating in the communication with the window manager. The inverse routine is `XmActivateWMProtocol`().

See Also

`XmActivateWMProtocol`(1), `XmDeactivateProtocol`(1), `XmInternAtom`(1), `VendorShell`(2).

Name

XmDestroyPixmap – remove a pixmap from the pixmap cache.

Synopsis

Boolean XmDestroyPixmap (Screen **screen*, Pixmap *pixmap*)

Inputs

screen	Specifies the screen on which the pixmap is located.
pixmap	Specifies the pixmap.

Returns

True on success or False if there is no matching *pixmap* and *screen* in the cache.

Description

`XmDestroyPixmap`() removes the specified *pixmap* from the pixmap cache when it is no longer needed. A pixmap is not completely freed until there are no further reference to it.

Usage

The pixmap cache maintains a per-client list of the pixmaps that are in use. Whenever a pixmap is requested using `XmGetPixmap`(), an internal reference counter for the pixmap is incremented. `XmDestroyPixmap`() decrements this counter, so that when it reaches 0 (zero), the pixmap is removed from the cache.

See Also

`XmGetPixmap`(1), `XmInstallImage`(1), `XmUninstallImage`(1).

Name

XmDirectionMatch – compare two directions.

Synopsis

Boolean XmDirectionMatch (XmDirection *dir_1*, XmDirection *dir_2*)

Inputs

dir_1 Specifies a direction.
dir_2 Specifies a direction to compare with *dir_1*.

Returns

True if the directions match, otherwise False.

Availability

Motif 2.0 and later.

Description

`XmDirectionMatch`() is a convenience function which compares two direction values, *dir_1* and *dir_2*, returning True or False, depending upon whether the values are a logical match for each other.

Usage

An XmDirection consists of three parts: a horizontal component, a vertical component, and an order of precedence between each. XmDirection values match if both the horizontal components and vertical components of each are logically the same, and the order between the components is the same. If one value does not have a horizontal component, this always matches the horizontal component of the other value. Similarly, if one value has no vertical component, the vertical component in the other value is automatically considered to match. Where a match is found between the directions, the function returns True, otherwise False.

For example, suppose *dir_1* is XmTOP_TO_BOTTOM_LEFT_TO_RIGHT. This has a vertical component XmTOP_TO_BOTTOM, a horizontal component XmLEFT_TO_RIGHT, the vertical component being first in the order of precedence. If *dir_2* is XmLEFT_TO_RIGHT, this has no vertical component, which automatically matches the vertical component of *dir_1*. The horizontal components are identical, and therefore the two directions are considered a match (it is also a match if *dir_1* is XmLEFT_TO_RIGHT_TOP_TO_BOTTOM). If *dir_2* is XmRIGHT_TO_LEFT, or XmTOP_TO_BOTTOM_RIGHT_TO_LEFT, no match is found because the horizontal components differ, and the function returns False. If *dir_2* is XmLEFT_TO_RIGHT_TOP_TO_BOTTOM, the function also returns False because the horizontal and vertical components, although fully specified and equal in value, have different orders of precedence.

Structures

Valid XmDirection values for each of *dir_1* and *dir_2* are:

XmLEFT_TO_RIGHT
XmRIGHT_TO_LEFT
XmBOTTOM_TO_TOP
XmTOP_TO_BOTTOM
XmBOTTOM_TO_TOP_LEFT_TO_RIGHT
XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
XmTOP_TO_BOTTOM_LEFT_TO_RIGHT
XmTOP_TO_BOTTOM_RIGHT_TO_LEFT
XmLEFT_TO_RIGHT_BOTTOM_TO_TOP
XmRIGHT_TO_LEFT_BOTTOM_TO_TOP
XmLEFT_TO_RIGHT_TOP_TO_BOTTOM
XmRIGHT_TO_LEFT_TOP_TO_BOTTOM

See Also

`XmDirectionMatchPartial`(1),
`XmDirectionToStringDirection`(1),
`XmStringDirectionToDirection`(1),

Name

XmDirectionMatchPartial – partially compare two directions.

Synopsis

Boolean XmDirectionMatchPartial (XmDirection *dir_1*, XmDirection *dir_2*, XmDirection *mask*)

Inputs

dir_1	Specifies a direction.
dir_2	Specifies another direction to compare with *dir_1*.
mask	Specifies whether the horizontal component (XmHORIZONTAL_MASK), vertical component (XmVERTICAL_MASK), or the order of component precedence (XmPRECEDENCE_MASK) is compared.

Returns

True if the directions match, otherwise False.

Availability

Motif 2.0 and later.

Description

`XmDirectionMatchPartial`() is a convenience function which compares two direction values, *dir_1* and *dir_2* according to the comparison rule specified in mask.

Usage

An XmDirection consists of three logical parts: a horizontal component, a vertical component, and an order of precedence between each. The function compares corresponding logical parts of two XmDirection values. If *mask* is XmHORIZONTAL_MASK, the horizontal components of *dir_1* and *dir_2* are compared. If *mask* is XmVERTICAL_MASK, the vertical components are compared. If *mask* is XmPRECEDENCE_MASK, the order of precedence between the horizontal and vertical components is compared. If one value does not have a particular logical part, this always matches the logical part in the second value. Where a match is found, the function returns True, otherwise False.

For example, suppose *dir_1* is XmTOP_TO_BOTTOM_LEFT_TO_RIGHT, and that *dir_2* is XmBOTTOM_TO_TOP_LEFT_TO_RIGHT. If *mask* is XmHORIZONTAL_MASK, the two values match because each has an equivalent horizontal component (XmLEFT_TO_RIGHT). If *mask* is XmVERTICAL_MASK, there is no match because each has different vertical components. If *mask* is XmPRECEDENCE_MASK, the two values are a match because each has the vertical component before the horizontal.

Structures

Valid XmDirection values for each of *dir_1* and *dir_1* are:

XmLEFT_TO_RIGHT	XmRIGHT_TO_LEFT
XmBOTTOM_TO_TOP	XmTOP_TO_BOTTOM
XmBOTTOM_TO_TOP_LEFT_TO_RIGHT	
XmBOTTOM_TO_TOP_RIGHT_TO_LEFT	
XmTOP_TO_BOTTOM_LEFT_TO_RIGHT	
XmTOP_TO_BOTTOM_RIGHT_TO_LEFT	
XmLEFT_TO_RIGHT_BOTTOM_TO_TOP	
XmRIGHT_TO_LEFT_BOTTOM_TO_TOP	
XmLEFT_TO_RIGHT_TOP_TO_BOTTOM	
XmRIGHT_TO_LEFT_TOP_TO_BOTTOM	

See Also

`XmDirectionMatch`(1), `XmDirectionToStringDirection`(1), `XmStringDirectionToDirection`(1),

Name

XmDirectionToStringDirection – convert a direction to a string direction.

Synopsis

XmStringDirection XmDirectionToStringDirection (XmDirection *direction*)

Inputs

direction Specifies the direction to be converted.

Returns

The equivalent XmStringDirection.

Availability

Motif 2.0 and later.

Description

`XmDirectionToStringDirection`() converts an XmDirection value specified by *direction* into an XmStringDirection value.

Usage

`XmDirectionToStringDirection`() converts between the XmDirection and XmStringDirection data types. If *direction* has a horizontal component, that component is converted. If the horizontal component is XmLEFT_TO_RIGHT, the function returns XmSTRING_DIRECTION_LEFT_TO_RIGHT. If the horizontal component is XmRIGHT_TO_LEFT, the function returns XmSTRING_DIRECTION_RIGHT_TO_LEFT. If *direction* has no horizontal component, the function returns XmSTRING_DIRECTION_DEFAULT.

For example, if *direction* is XmRIGHT_TO_LEFT_TOP_TO_BOTTOM, the horizontal component is XmRIGHT_TO_LEFT, and the return value is XmSTRING_DIRECTION_RIGHT_TO_LEFT. If *direction* is XmBOTTOM_TO_TOP, the value has only a vertical component, and the function returns XmSTRING_DIRECTION_DEFAULT.

See Also

`XmDirectionMatch`(1), `XmDirectionMatchPartial`(1), `XmStringDirectionToDirection`(1).

Name

XmDragCancel – cancel a drag operation.

Synopsis

#include <Xm/DragDrop.h>

void XmDragCancel (Widget *dragcontext*)

Inputs

dragcontext Specifies the ID of the DragContext object for the drag operation that is being cancelled.

Description

`XmDragCancel`() cancels the drag operation that is in progress for the specified *dragcontext*. If the DragContext has any actions pending, they are terminated. The routine can only be called by the client that initiated the drag operation. `XmDragCancel`() frees the DragContext object associated with the drag operation.

Usage

`XmDragCancel`() allows an initiating client to cancel a drag operation if it decides that the operation should not continue for whatever reason. Calling `XmDragCancel`() is equivalent to the user pressing KCancel during the drag. The XmNdropStartCallback informs the initiating client of the cancellation by setting the dropAction field to XmDROP_CANCEL. So that it can undo any drag-under effects under the dynamic protocol, the receiving client gets an XmCR_DROP_SITE_LEAVE_MESSAGE when the drag is cancelled.

See Also

`XmDragStart`(1), `XmDragContext`(2).

Name

XmDragStart – start a drag operation.

Synopsis

#include <Xm/DragDrop.h>

Widget XmDragStart (Widget *widget*, XEvent **event*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies the widget or gadget that contains the data that is being dragged.
event	Specifies the event that caused the drag operation.
arglist	Specifies the resource name/value pairs used in creating the DragContext.
argcount	Specifies the number of name/value pairs in arglist.

Returns

The ID of the DragContext object that is created.

Availability

In Motif 2.0 and later, `XmDragStart`() is subsumed into the Uniform Transfer Model (UTM). The Motif widget classes do not call `XmDragStart`() directly, but install the XmQTtransfer trait to provide data transfer and conversion, and initiate the drag through UTM mechanisms which calls `XmDragStart`() internally.

Description

`XmDragStart`() starts a drag operation by creating and returning a DragContext object. The DragContext stores information that the toolkit needs to process a drag transaction. The DragContext object is widget-like, in that it uses resources to specify its attributes. The toolkit frees the DragContext upon completion of the drag and drop operation.

The *widget* argument to `XmDragStart`() should be the smallest widget that contains the source data for the drag operation. The *event* that starts the drag operation must be a ButtonPress event. The *arglist* and *argcount* parameters work as for any creation routine; any DragContext resources that are not set by the arguments are retrieved from the resource database or set to their default values.

Usage

Motif supports the drag and drop model of selection actions. In a widget that acts as a drag source, a user can make a selection and then drag the selection, using BTransfer, to other widgets that are registered as drop sites. These drop sites can be in the same application or another application.

The Text and TextField widgets, the List widget, and Label and its subclasses are set up to act as drag sources by the toolkit. In order for another widget to act as a drag source, it must have a translation for BTransfer. The action routine for the translation calls `XmDragStart`(), either directly or indirectly through the UTM, to initiate the drag and drop operation.

The only DragContext resource that must be specified when `XmDragStart`() is called is the XmNconvertProc procedure. This resource specifies a procedure of type XtConvertSelectionIncrProc that converts the source data to the format(s) requested by the receiving client. The specification of the other resources, such as those for operations and drag-over visuals, is optional. For more information about the DragContext object, see the manual page in Section 2, *Motif and Xt Widget Classes*].

Example

The following routines show the use of `XmDragStart`() in setting up a ScrollBar to function as a drag source. When the ScrollBar is created, the translations are overridden to invoke StartDrag when BTransfer is pressed. ConvertProc, which is not shown here, is set up by StartDrag to perform the translation of the scrollbar data into compound text format.

```
/*
** XmSCOMPOUND_TEXT is defined in Motif 2.0 and
later
*/
#ifndef   XmSCOMPOUND_TEXT
#define   XmSCOMPOUND_TEXT    "COMPOUND_TEXT"
#endif /* XmSCOMPOUND_TEXT */

/* global variable */
Atom COMPOUND_TEXT;

/* start the drag operation */
static void StartDrag( Widget      widget,
                       XEvent      *event,
                       String      *params,
                       Cardinal    *num_params)
{
   Arg args[10];
   int n = 0;
   Atom exportList[1];

   exportList[0] = COMPOUND_TEXT;
```

```
    XtSetArg (args[n], XmNexportTargets,
    exportList); n++;
    XtSetArg (args[n], XmNnumExportTargets, XtNumber
    (exportList));
    n++;
    XtSetArg (args[n], XmNdragOperations,
    XmDROP_COPY); n++;
    XtSetArg (args[n], XmNconvertProc, ConvertProc);
    n++;
    XtSetArg (args[n], XmNclientData, widget); n++;

    XmDragStart (widget, event, args, n);
}

/* define translations and actions */
static char dragTranslations[] =
    "#override <Btn2Down>: StartDrag()";

static XtActionsRec dragActions[] =
    { {"StartDrag", (XtActionProc) StartDrag} };

void main (unsigned int argc, char **argv)
{
    Arg            args[10];
    int            n;
    Widget         top, bboard, scrollbar;
    XtAppContext   app;
    XtTranslations parsed_trans;

    XtSetLanguageProc (NULL, (XtLanguageProc) NULL,
    NULL);

    top = XtAppInitialize (&app, "Drag", NULL, 0,
                           &argc, argv, NULL, NULL,
                           0);

    COMPOUND_TEXT = XInternAtom (XtDisplay (widget),
                                 XmSCOMPOUND_TEXT,
                                 False);

    n = 0;
    bboard = XmCreateBulletinBoard (top, "bboard",
    args, n);
    XtManageChild (bboard);

    /* override button two press to start a drag */
    parsed_trans = XtParseTranslationTable
    (dragTranslations);
```

```
        XtAppAddActions (app, dragActions, XtNumber
        (dragActions));

        n = 0;
        XtSetArg (args[n], XmNtranslations,
        parsed_trans); n++;
        XtSetArg (args[n], XmNorientation, XmHORIZON-
        TAL); n++;
        XtSetArg (args[n], XmNwidth, 100); n++;
        scrollbar = XmCreateScrollBar (bboard, "scroll-
        bar", args, n);
        XtManageChild (scrollbar);

        XtRealizeWidget (top);
        XtAppMainLoop (app);
    }
```

See Also

XmDragCancel(1), XmTransfer(1), XmDragContext(2).

Name

XmDropDownGetArrow –A DropDown function that returns the “arrow” child of the XmDropDown.

Synopsis

#include <Xm/DropDown.h>

Widget XmDropDownGetArrow(Widget *widget*)

Inputs

widget Specifies the DropDown widget ID.

Outputs

value_return Returns the widget ID of the “arrow” child of the DropDown widget.

Description

`XmDropDownGetArrow`() is used to access the “arrow” child within the DropDown *widget*.

Usage

`XmDropDownArrow`() is a convenience routine that returns the “arrow” child within the DropDown widget.

See Also

`XmDropDown`(2).

Name

XmDropDownGetChild –get the specified child of a DropDown widget.

Synopsis

#include <Xm/DropDown.h>

void XmDropDownGetChild (Widget *widget*, int *child*)

Inputs

widget Specifies the DropDown widget.
child Specifies a type of child of the DropDown widget.

Returns

The widget ID of the specified child of the DropDown widget.

Description

XmDropDownGetChild() returns the widget ID of the specified child of the DropDown widget.

Usage

The *child* XmDROPDOWN_LABEL specifies the label of the DropDown, XmDROPDOWN_TEXT specifies the text area of the DropDown, XmDROPDOWN_ARROW_BUTTON specifies the arrow button of the Drop-Down, and XmDROPDOWN_LIST specifies the list widget of DropDown.

Structures

The possible values for child are:

XmDROPDOWN_LABEL XmDROPDOWN_TEXT
XmDROPDOWN_ARROW_BUTTON XmDROPDOWN_LIST

Widget Hierarchy

The following names are associated with the DropDown children:

"Label"	XmDROPDOWN_LABEL
"Arrow"	XmDROPDOWN_ARROW_BUTTON
"Text"	XmDROPDOWN_TEXT
"List"	XmDROPDOWN_LIST

See Also

`XmDropDown`(2).

Name

XmDropDownGetLabel – A DropDown function that returns the “label” child of the XmDropDown.

Synopsis

#include <Xm/DropDown.h>

Widget XmDropDownGetLabel(Widget *widget*)

Inputs

widget Specifies the DropDown widget ID.

Outputs

value_return Returns the widget ID of the “label” child of the DropDown widget.

Description

`XmDropDownGetLabel`() is used to access the “label” child within the DropDown widget.

Usage

`XmDropDownGetLabel`() is a convenience routine that returns the “label” child of the DropDown widget.

See Also

`XmDropDown`(2).

Name

XmDropDownGetList – A DropDown function that returns the “list” child of the XmDropDown.

Synopsis

#include <Xm/DropDown.h>

Widget XmDropDownGetList(Widget *widget*)

Inputs

widget Specifies the DropDown widget ID.

Outputs

value_return Returns the widget ID of the “list” child of the DropDown widget.

Description

`XmDropDownGetList`() is used to access the “list” child within the DropDown *widget*.

Usage

`XmDropDownGetList`() is a convenience routine that returns the “list” child of the DropDown widget.

See Also

`XmDropDown`(2).

Name

XmDropDownGetText– A DropDown function that returns the “text” child of the XmDropDown.

Synopsis

#include <Xm/DropDown.h>

Widget XmDropDownGetText (Widget *widget*)

Inputs

widget Specifies the DropDown widget ID.

Outputs

value_return Returns the widget ID of the “text” child of the DropDown widget.

Description

`XmDropDownGetText`() is used to access the “text” child within the DropDown widget.

Usage

`XmDropDownGetText`() is a convenience routine that returns the “text” child of the DropDown widget.

See Also

`XmDropDown`(2).

Name

XmDropDownGetValue–retrieve the value from DropDown.

Synopsis

#include <Xm/DropDown.h>

void String XmDropDownGetValue (Widget *w*)

Inputs

widget Specifies the DropDown widget ID.

Outputs

value_return Returns the string contained within the text widget of the DropDown widget.

Description

The `XmDropDownGetValue()` function returns the string contained within the text widget of the DropDown widget.

Usage

`XmDropDownGetValue`() is a convenience routine that returns the string contained within the text widget of the DropDown widget. This string has been allocated with `XtMalloc()` and must be freed by the application with `XtFree()`. Always leave the specified margin between its edge and the nearest child. A new String to fillOption resource converter has been registered to convert the following strings to fill options: “none”, “major”, “minor”, “all”. This resource can therefore be set in an application defaults file.

See Also

`XmDropDown`(2).

Name

XmDropSiteConfigureStackingOrder – change the stacking order of a drop site.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteConfigureStackingOrder (Widget *widget*, Widget *sibling*, Cardinal *stack_mode*)

Inputs

widget	Specifies the widget ID associated with the drop site.
sibling	Specifies an optional widget ID of a sibling drop site.
stack_mode	Specifies the stacking position. Pass either XmABOVE or XmBELOW.

Description

`XmDropSiteConfigureStackingOrder`() changes the stacking order of a drop site relative to its siblings. The routine changes the stacking order of the drop site associated with the specified *widget*. The stacking order is changed only if the drop sites associated with *widget* and *sibling* are siblings in both the widget hierarchy and the drop site hierarchy. The parent of both of the widgets must be registered as a composite drop site.

If *sibling* is specified, the stacking order of the drop site is changed relative to the stack position of the drop site associated with *sibling*, based on the value of *stack_mode*. If *stack_mode* is XmABOVE, the drop site is positioned just above the sibling; if *stack_mode* is XmBELOW, the drop site is positioned just below the sibling. If *sibling* is not specified, a *stack_mode* of XmABOVE causes the drop site to be placed at the top of the stack, while a *stack_mode* of XmBELOW[1] causes it to be placed at the bottom of the stack.

Usage

A drop site for drag and drop operations can be a composite drop site, which means that it has children which are also drop sites. The stacking order of the drop sites controls clipping of drag-under effects during a drag and drop operation. When drop sites overlap, the drag-under effects of the drop sites lower in the stacking order are clipped by the drop sites above them, regardless of whether or not the drop sites are active. You can use `XmDropSiteConfigureStackingOrder`() to modify the stacking order. Use `XmDropSiteQueryStackingOrder`() to get the current stacking order.

See Also

1.Erroneously given as BELOW in 1st and 2nd editions.

```
XmDropSiteQueryStackingOrder(1),
XmDropSiteRegister(1), XmDropSite(2)
```

Name

XmDropSiteEndUpdate – end an update of multiple drop sites.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteEndUpdate (Widget *widget*)

Inputs

widget Specifies any widget in the hierarchy associated with the drop sites that are to be updated.

Description

`XmDropSiteEndUpdate`() finishes an update of multiple drop sites. The *widget* parameter specifies a widget in the widget hierarchy that contains all of the widgets associated with the drop sites being updated. The routine uses *widget* to identify the shell that contains all of the drop sites.

Usage

`XmDropSiteEndUpdate`() is used with `XmDropSiteStartUpdate`() and `XmDropSiteUpdate`() to update information about multiple drop sites in the DropSite registry. `XmDropSiteStartUpdate`() starts the update processing, `XmDropSiteUpdate`() is called multiple times to update information about different drop sites, and `XmDropSiteEndUpdate`() completes the processing. These routines optimize the updating of drop site information. Calls to `XmDrop-SiteStartUpdate`() and `XmDropSiteEndUpdate`() can be nested recursively.

See Also

`XmDropSiteStartUpdate`(1), `XmDropSiteUpdate`(1), `XmDropSite`(2).

Name

XmDropSiteQueryStackingOrder – get the stacking order of a drop site.

Synopsis

```
#include <Xm/DragDrop.h>

Status XmDropSiteQueryStackingOrder (  Widget    widget,
                                       Widget    *parent_return,
                                       Widget    **child_returns,
                                       Cardinal  *num_child_returns)
```

Inputs

widget	Specifies the widget ID associated with a composite drop site.

Outputs

parent_return	Returns the widget ID of the parent of the specified *widget*.
child_returns	Returns a list of the children of *widget* that are registered as drop sites.
num_child_returns	Returns the number of children in *child_returns*.

Returns

A non-zero value on success or 0 (zero) on failure.

Description

`XmDropSiteQueryStackingOrder`() retrieves information about the stacking order of drop sites. For the specified *widget*, the routine returns its parent and a list of its children that are registered as drop sites. The children are returned in *child_returns*, which lists the children in the current stacking order, with the lowest child in the stacking order at the beginning of the list and the top child at the end of the list. `XmDropSiteQueryStackingOrder`() allocates storage for the list of returned children. The application is responsible for managing this storage, which can be freed using `XtFree`(). The routine returns a non-zero value on success or 0 (zero) on failure.

Usage

A drop site for drag and drop operations can be a composite drop site, which means that it has children which are also drop sites. The stacking order of the drop sites controls clipping of drag-under effects during a drag and drop operation. When drop sites overlap, the drag-under effects of the drop sites lower in the stacking order are clipped by the drop sites above them, regardless of whether or not the drop sites are active. Use `XmDropSiteQueryStackingOrder`() to get the current stacking order for a composite drop site. You can use `XmDropSiteConfigureStackingOrder`() to modify the stacking order.

Text, TextField, and Container widgets are automatically registered as drop sites by the Motif toolkit.

See Also

`XmDropSiteConfigureStackingOrder`(1), `XmDropSiteRegister`(1), `XmDropSite`(2).

Name

XmDropSiteRegister – register a drop site.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteRegister (Widget *widget*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies the widget ID that is to be associated with the drop site.
arglist	Specifies the resource name/value pairs used in registering the drop site.
argcount	Specifies the number of name/value pairs in arglist.

Availability

In Motif 2.0 and later, `XmDropSiteRegister`() is subsumed into the Uniform Transfer Model (UTM). The Motif widget classes do not call `XmDropSiteRegister`() directly, but initiate the site through UTM mechanisms which call `XmDropSiteRegister`() internally. The callbacks specified by the XmNdestinationCallback resource of a widget handle the data drop.

Description

`XmDropSiteRegister`() registers the specified widget as a drop site, which means the widget has a drop site associated with it in the DropSite registry. Drop sites are widget-like, in that they use resources to specify their attributes. The arglist and argcount parameters work as for any creation routine; any drop site resources that are not set by the arguments are retrieved from the resource database or set to their default values. If the drop site is registered with XmNdropSiteActivity set to XmDROP_SITE_ACTIVE and XmNdropProc set to NULL, the routine generates a warning message.

Usage

Motif supports the drag and drop model of selection actions. In a widget that acts as a drag source, a user can make a selection and then drag the selection, using BTransfer, to other widgets that are registered as drop sites. The DropSite registry stores information about all of the drop sites for a display. Text and TextField widgets are automatically registered as drop sites when they are created. An application can register other widgets as drop sites using `XmDropSiteRegister`(). Once a widget is registered as a drop site, it can participate in drag and drop operations. A drop site can be removed from the registry using `XmDropSiteUnregister`(). When a drop site is removed, the widget no longer participates in drag and drop operations.

A drop site for drag and drop operations can be a composite drop site, which means that it has children which are also drop sites. If the drop site being registered is a descendant of a widget that has already been registered as a drop site, the XmNdropSiteType resource of the ancestor must be set to XmDROP_SITE_COMPOSITE. A composite drop site must be registered as a drop site before its descendants are registered. The stacking order of the drop sites controls clipping of drag-under effects during a drag and drop operation. When drop sites overlap, the drag-under effects of the drop sites lower in the stacking order are clipped by the drop sites above them, regardless of whether or not the drop sites are active. When a descendant drop site is registered, it is stacked above all of its sibling drop sites that have already been registered.

Example

The following routine shows the use of `XmDropSiteRegister`() to register a Label widget as a drop site. When a drop operation occurs in the Label, the HandleDrop routine, which is not shown here, handles the drop:

```
/* global variable */
Atom COMPOUND_TEXT;

void main (unsigned int argc, char **argv)
{
   Arg          args[10];
   int          n;
   Widget       top, bb, label;
   XtAppContext app;
   Atom         importList[1];

   XtSetLanguageProc (NULL, (XtLanguageProc) NULL,
   NULL);
   top = XtAppInitialize (&app, "Drop", NULL, 0,
                          &argc, argv, NULL, NULL,
                          0);

   n = 0;
   bb = XmCreateBulletinBoard (top, "bb", args, n);
   XtManageChild (bb);

   COMPOUND_TEXT = XInternAtom (XtDisplay (top),
                                "COMPOUND_TEXT",
                                False);

   n = 0;
   label = XmCreateLabel (bb, "Drop Here", args,
   n);
```

```
    XtManageChild (label);

    /* register the label as a drop site */
    importList[0] = COMPOUND_TEXT;

    n = 0;
    XtSetArg (args[n], XmNimportTargets,
    importList); n++;
    XtSetArg (args[n], XmNnumImportTargets, XtNumber
    (importList)); n++;
    XtSetArg (args[n], XmNdropSiteOperations,
    XmDROP_COPY); n++;
    XtSetArg (args[n], XmNdropProc, HandleDrop);
    n++;
    XmDropSiteRegister (label, args, n);

    XtRealizeWidget (top);
    XtAppMainLoop (app);
}
```

See Also

XmDropSiteConfigureStackingOrder(1), XmDropSiteEndUpdate(1), XmDropSiteQueryStackingOrder(1), XmDropSiteRetrieve(1), XmDropSiteStartUpdate(1), XmDropSiteUpdate(1), XmDropSiteUnregister(1), XmTransfer(1), XmDisplay(2), XmDropSite(2), XmScreen(2).

Name

XmDropSiteRetrieve – get the resource values for a drop site.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteRetrieve (Widget *widget*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies the widget ID associated with the drop site.
arglist	Specifies the resource name/address pairs that contain the resource names and addresses into which the resource values are stored.
argcount	Specifies the number of name/value pairs in *arglist*.

Description

`XmDropSiteRetrieve`() gets the specified resources for the drop site associated with the specified *widget*. Drop sites are widget-like, in that they use resources to specify their attributes. The *arglist* and *argcount* parameters work as for `XtGetValues`().

Usage

`XmDropSiteRetrieve`() can be used to get the current attributes of a drop site from the DropSite registry. The DropSite registry stores information about all of the drop sites for a display. An initiating client can also use `XmDropSiteRetrieve`() to retrieve information about the current drop site by passing the DragContext for the operation to the routine. The initiator can access all of the drop site resources except XmNdragProc and XmNdropProc[1] using this technique.

See Also

`XmDropSiteRegister`(1), `XmDropSiteUpdate`(1), `XmDropSite`(2).

1.Erroneously given as XmdropProc in 1st and 2nd editions.

Name

XmDropSiteStartUpdate – start an update of multiple drop sites.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteStartUpdate (Widget *widget*)

Inputs

widget Specifies any widget in the hierarchy associated with the drop sites that are to be updated.

Description

`XmDropSiteStartUpdate`() begins an update of multiple drop sites. The *widget* parameter specifies a widget in the widget hierarchy that contains all of the widgets associated with the drop sites being updated. The routine uses *widget* to identify the shell that contains all of the drop sites.

Usage

`XmDropSiteStartUpdate`() is used with `XmDropSiteUpdate`() and `XmDropSiteEndUpdate`() to update information about multiple drop sites in the DropSite registry. `XmDropSiteStartUpdate`() starts the update processing, `XmDropSiteUpdate`() is called multiple times to update information about different drop sites, and `XmDropSiteEndUpdate`() completes the processing. These routines optimize the updating of drop site information. Calls to `XmDropSiteStartUpdate`() and `XmDropSiteEndUpdate`() can be nested recursively.

See Also

`XmDropSiteEndUpdate`(1), `XmDropSiteUpdate`(1), `XmDropSite`(2).

Name

XmDropSiteUnregister – remove a drop site.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteUnregister (Widget *widget*)

Inputs

widget Specifies the widget ID associated with the drop site.

Description

`XmDropSiteUnregister`() removes the drop site associated with the specified *widget* from the DropSite registry. After the routine is called, the widget cannot be the receiver in a drag and drop operation. The routine frees all of the information associated with the drop site.

Usage

Motif supports the drag and drop model of selection actions. In a widget that acts as a drag source, a user can make a selection and then drag the selection, using BTransfer, to other widgets that are registered as drop sites. Once a widget is registered as a drop site with `XmDropSiteRegister`(), it can participate in drag and drop operations. Text and TextField widgets are automatically registered as drop sites when they are created. `XmDropSiteUnregister`() provides a way to remove a drop site from the registry, so that the widget no longer participates in drag and drop operations.

See Also

`XmDropSiteRegister`(1), `XmDropSite`(2).

Name

XmDropSiteUpdate – change the resource values for a drop site.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteUpdate (Widget *widget*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies the widget ID associated with the drop site.
arglist	Specifies the resource name/value pairs used in updating the drop site.
argcount	Specifies the number of name/value pairs in arglist.

Description

`XmDropSiteUpdate`() changes the resources for the drop site associated with the specified *widget*. Drop sites are widget-like, in that they use resources to specify their attributes. The *arglist* and *argcount* parameters work as for `XtSetValues`().

Usage

`XmDropSiteUpdate`() can be used by itself to update the attributes of a drop site. The routine can also be used with `XmDropSiteStartUpdate`() and `XmDropSiteEndUpdate`() to update information about multiple drop sites in the DropSite registry. `XmDropSiteStartUpdate`() starts the update processing, `XmDropSiteUpdate`() is called multiple times to update information about different drop sites, and `XmDropSiteEndUpdate`() completes the processing. The DropSite registry stores information about all of the drop sites for a display. These routines optimize the updating of drop site information by sending all of the updates at once, rather than processing each one individually.

See Also

`XmDropSiteEndUpdate`(1), `XmDropSiteRegister`(1), `XmDropSiteStartUpdate`(1), `XmDropSiteUnregister`(1), `XmDropSite`(2).

Name

XmDropTransferAdd – add drop transfer entries to a drop operation.

Synopsis

```
#include <Xm/DragDrop.h>

void XmDropTransferAdd ( Widget                  drop_transfer,
                         XmDropTransferEntryRec  *transfers,
                         Cardinal                num_transfers)
```

Inputs

drop_transfer Specifies the ID of the DropTransfer object to which the entries are being added.

transfers Specifies the additional drop transfer entries.

num_transfer Specifies the number of drop transfer entries in *transfers*.

Availability

In Motif 2.0 and later, the drag and drop mechanisms are rationalized as part of the Uniform Transfer Model. Motif widget classes do not call `XmDropTransferAdd`() directly, but call XmTransferValue() to transfer data to a destination. `XmTransferValue`() calls `XmDropTransferAdd`() internally as the need arises.

Description

`XmDropTransferAdd`() specifies a list of additional drop transfer entries that are to be processed during a drop operation. The *widget* argument specifies the DropTransfer object associated with the drop operation. *transfers* is an array of XmDropTransferEntryRec structures that specifies the targets of the additional drop transfer operations. `XmDropTransferAdd`() can be used to modify the DropTransfer object until the last call to the XmNtransferProc is made. After the last call, the result of modifying the DropTransfer object is undefined.

Usage

The toolkit uses the DropTransfer object to manage the transfer of data from the drag source to the drop site during a drag and drop operation. `XmDropTransferAdd`() provides a way for a drop site to specify additional target formats after a drop operation has started. The routine adds the entries to the XmNdropTransfers resource. The attributes of a DropTransfer object can also be manipulated with `XtSetValues`() and `XtGetValues`().

Structures

XmDropTransferEntryRec is defined as follows:

```
typedef struct {
    XtPointer    client_data;    /* data passed to the transfer proc */
    Atom         target;         /* target format of the transfer    */
} XmDropTransferEntryRec, *XmDropTransferEntry;
```

See Also

`XmDropTransferStart`(1), `XmTransferValue`(1), `XmDragContext`(2), `XmDropTransfer`(2).

Name

XmDropTransferStart – start a drop operation.

Synopsis

#include <Xm/DragDrop.h>

Widget XmDropTransferStart (Widget *widget*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies the ID of the DragContext object associated with the operation.
arglist	Specifies the resource name/value pairs used in creating the DropTransfer.
argcount	Specifies the number of name/value pairs in arglist.

Returns

The ID of the DropTransfer object that is created.

Availability

In Motif 2.0 and later, the drag and drop mechanisms are rationalized as part of the Uniform Transfer Model. `XmDropTransferStart`() is called on request internally as the need arises by the destination callback handlers, or through the `XmTransferValue`() and `XmTransferDone`() functions.

Description

`XmDropTransferStart`() starts a drop operation by creating and returning a DropTransfer object. The DropTransfer stores information that the toolkit needs to process a drop transaction. The DropTransfer is widget-like, in that it uses resources to specify its attributes. The toolkit frees the DropTransfer upon completion of the drag and drop operation.

The *widget* argument to `XmDropTransferStart`() is the DragContext object associated with the drag operation. The *arglist* and *argcount* parameters work as for any creation routine; any DropTransfer resources that are not set by the arguments are retrieved from the resource database or set to their default values.

Usage

Motif 1.2 supports the drag and drop model of selection actions. In a widget that acts as a drag source, a user can make a selection and then drag the selection, using BTransfer, to other widgets that are registered as drop sites. These drop sites can be in the same application or another application. The toolkit uses the DropTransfer object to manage the transfer of data from the drag source to the drop site. XmDropTransferStart() is typically called from within the XmNdropProc procedure of the drop site.

The attributes of a DropTransfer object can be manipulated with `XtSetValues`() and `XtGetValues`() until the last call to the XmNtransferProc procedure is made. You can also use `XmDropTransferAdd`() to add drop transfer entries to be processed. After the last call to XmNtransferProc, the result of using the DropTransfer object is undefined. For more information about the DropTransfer object, see the manual page in Section 2, *Motif and Xt Widget Classes*.

Example

The following routine shows the use of `XmDropTransferStart`() in the HandleDrop routine, which is the XmNdropProc procedure for a Label widget that is being used as a drop site. The data transfer procedure TransferProc() which presumably translates the data in the Label into compound text format, is not shown.

```
/* global variable */
Atom COMPOUND_TEXT;

static void HandleDrop(Widget    widget,
                       XtPointer  client_data,
                       XtPointer  call_data)
{
   XmDropProcCallback      DropData;
   XmDropTransferEntryRec  transferEntries[1];
   XmDropTransferEntry     transferList;
   Arg                     args[10];
   int                     n;

   DropData = (XmDropProcCallback) call_data;

   n = 0;

   if ( (DropData->dropAction != XmDROP) ||
        (DropData->operation != XmDROP_COPY)) {
      XtSetArg (args[n], XmNtransferStatus,
      XmTRANSFER_FAILURE);
      n++;
   }
   else {
      transferEntries[0].target = COMPOUND_TEXT;
      transferEntries[0].client_data = (XtPointer)
      widget;
      transferList = transferEntries;
      XtSetArg (args[n], XmNdropTransfers, trans-
      ferEntries); n++;
      XtSetArg (args[n], XmNnumDropTransfers,
```

```
                    XtNumber (transferEntries)); n++;
            XtSetArg (args[n], XmNtransferProc, Transfer-
            Proc); n++;
         }

         XmDropTransferStart (DropData->dragContext,
      args, n);
      }
```

See Also

XmDropTransferAdd(1), XmTransferValue(1), XmTransferDone(1), XmDragContext(2), XmDropTransfer(2).

Name

XmFileSelectionBoxGetChild – get the specified child of a FileSelectionBox widget.

Synopsis

#include <Xm/FileSB.h>

Widget XmFileSelectionBoxGetChild (Widget *widget*, unsigned char *child*)

Inputs

widget Specifies the FileSelectionBox widget.
child Specifies the child of the FileSelectionBox widget. Possible values are defined below.

Returns

The widget ID of the specified child of the FileSelectionBox.

Availability

From Motif 2.0, `XmFileSelectionBoxGetChild`() is deprecated code. `XtNameToWidget`() is the preferred method of accessing children of the widget.

Description

`XmFileSelectionBoxGetChild`() returns the widget ID of the specified *child* of the FileSelectionBox *widget*.

Usage

XmDIALOG_APPLY_BUTTON, XmDIALOG_CANCEL_BUTTON, XmDIALOG_HELP_BUTTON, and XmDIALOG_OK_BUTTON specify the action buttons in the widget. XmDIALOG_DEFAULT_BUTTON specifies the current default button. XmDIALOG_DIR_LIST and XmDIALOG_DIR_LIST_LABEL specify the directory list and its label, while XmDIALOG_LIST and XmDIALOG_LIST_LABEL specify the file list and its label. XmDIALOG_FILTER_LABEL and XmDIALOG_FILTER_TEXT specify the filter text entry area and its label, while XmDIALOG_TEXT and XmDIALOG_SELECTION_LABEL specify the file text entry area and its label. XmDIALOG_SEPARATOR specifies the separator and XmDIALOG_WORK_AREA specifies any work area child that has been added to the FileSelectionBox.

In Motif 2.0 and later, if the resource XmNpathMode is XmPATH_MODE_RELATIVE, the directory pattern specification is displayed in two text fields, rather than the single filter text entry area. When this is the case, the pattern is displayed in the original filter text area, and the directory portion is displayed in an additional text field called DirText. The Label associated

with the DirText child is called DirL. No corresponding mask has been defined to access this extra text field or its Label through `XmFileSelectionBoxGetChild`(): `XtNameToWidget`() should be used to access the DirText widget ID when required.

For more information on the different children of the FileSelectionBox, see the manual page in Section 2, *Motif and Xt Widget Classes*.

Widget Hierarchy

As of Motif 2.0, most Motif composite child fetch routines are marked as deprecated. However, since it is not possible to fetch the XmDIALOG_DEFAULT_BUTTON or XmDIALOG_WORK_AREA children using a public interface except through XmSelectionBoxGetChild()[1], the routine should not be considered truly deprecated. For consistency with the preferred new style, when fetching all other child values, consider giving preference to the Intrinsics routine XtNameToWidget(), passing one of the following names as the second parameter:

“Apply”	(XmDIALOG_APPLY_BUTTON)
“Cancel”	(XmDIALOG_CANCEL_BUTTON)
“OK”	(XmDIALOG_OK_BUTTON)
“Separator”	(XmDIALOG_SEPARATOR)
“Help”	(XmDIALOG_HELP_BUTTON)
“Symbol”	(XmDIALOG_SYMBOL_LABEL)
“Message”	(XmDIALOG_MESSAGE_LABEL)
“*ItemsList”[2]	(XmDIALOG_LIST)
“Items”	(XmDIALOG_LIST_LABEL)
“Selection”	(XmDIALOG_SELECTION_LABEL)
“Text”	(XmDIALOG_TEXT)
“*DirList“[3]	(XmDIALOG_DIR_LIST)
“Dir“	(XmDIALOG_DIR_LIST_LABEL)
“FilterLabel“	(XmDIALOG_FILTER_LABEL)
“FilterText“	(XmDIALOG_FILTER_TEXT)
“DirL“	(no macro - must use XtNameToWidget()))
“DirText“	(no macro - must use XtNameToWidget())

1.Called internally by XmFileSelectionBoxGetChild().

2.The “*” is important: the Files List is not a direct child of the SelectionBox, but of a ScrolledList.

3.As above; the Directories list is a child of a ScrolledWindow, not the SelectionBox itself.

CDE variants of the Motif 2.1 toolkit may support a ComboBox in place of the Directory Text field (DirText). This is known as “DirComboBox”, and also has no defined public macro[1]:

“DirComboBox” (no macro - must use XtNameToWidget())

Structures

The possible values for child are:

XmDIALOG_APPLY_BUTTON	XmDIALOG_LIST
XmDIALOG_CANCEL_BUTTON	XmDIALOG_LIST_LABEL
XmDIALOG_DEFAULT_BUTTON	XmDIALOG_OK_BUTTON
XmDIALOG_DIR_LIST	
XmDIALOG_SELECTION_LABEL	
XmDIALOG_DIR_LIST_LABEL	XmDIALOG_SEPARATOR
XmDIALOG_FILTER_LABEL	XmDIALOG_TEXT
XmDIALOG_FILTER_TEXT	XmDIALOG_WORK_AREA
XmDIALOG_HELP_BUTTON	

See Also

`XmFileSelectionBox`(2).

1.The ComboBox, containing a List of directories, is enabled if the CDE resource XmNenableFsbPickList is true.

Name

XmFileSelectionDoSearch – start a directory search.

Synopsis

#include <Xm/FileSB.h>

void XmFileSelectionDoSearch (Widget *widget*, XmString *dirmask*)

Inputs

widget	Specifies the FileSelectionBox widget.
dirmask	Specifies the directory mask that is used in the directory search.

Description

`XmFileSelectionDoSearch`() starts a directory and file search for the specified FileSelectionBox *widget*. *dirmask* is a text pattern that can include wildcard characters. `XmFileSelectionDoSearch`() updates the lists of directories and files that are displayed by the FileSelectionBox. If *dirmask* is non-NULL, the routine restricts the search to directories that match the *dirmask*.

Usage

`XmFileSelectionDoSearch`()[1] allows you to force a FileSelectionBox to reinitialize itself, which is useful if you want to set the directory mask directly.

See Also

`XmFileSelectionBox`(2).

1.Erroneously given as XmFileSelectionBoxDoSearch() in 1st and 2nd editions.

Name

XmFontListAdd – create a new font list.

Synopsis

XmFontList XmFontListAdd (XmFontList *oldlist*, XFontStruct **font*, XmStringCharSet *charset*)

Inputs

oldlist	Specifies the font list to which font is added.
font	Specifies the font structure.
charset	Specifies a tag that identifies the character set for the font.

Returns

The new font list, oldlist if font or charset is NULL, or NULL if oldlist is NULL.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively. To maintain backwards compatibility, the XmFontList is re-implemented as a render table.

Description

`XmFontListAdd`() makes a new font list by adding the font structure specified by *font* to the old font list. The routine returns the new font list and deallocates *oldlist*. *charset* specifies the character set that is associated with the font. It can be XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

`XmFontListAdd`() searches the font list cache for a font list that matches the new font list. If the routine finds a matching font list, it returns that font list and increments its reference count. Otherwise, the routine allocates space for the new font list and caches it. In either case, the application is responsible for managing the memory associated with the font list. When the application is done using the font list, it must be freed using `XmFontListFree`().

Usage

In Motif 1.1 and 1.2, a font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a font or a font set and an associated tag. In Motif 2.0 and later, the XmFontList is implemented using the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListAdd`() returns a reference counted render table.

`XmFontListAdd`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListAppendEntry`(1), `XmFontListFree`(1), `XmRenderTableAddRenditions`(1), `XmRenditionCreate`(1), `XmRendition`(2).

Name

XmFontListAppendEntry – append a font entry to a font list.

Synopsis

XmFontList XmFontListAppendEntry (XmFontList *oldlist*, XmFontListEntry *entry*)

Inputs

oldlist	Specifies the font list to which entry is appended.
entry	Specifies the font list entry.

Returns

The new font list or oldlist if entry is NULL.

Availability

Motif 1.2 and later. In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListAppendEntry`() makes a new font list by appending the specified *entry* to the old font list. If *oldlist* is NULL, the routine creates a new font list that contains the single entry. `XmFontListAppendEntry`() returns the new font list and deallocates *oldlist*. The application is responsible for freeing the font list entry using `XmFontListEntryFree`().

`XmFontListAppendEntry`() searches the font list cache for a font list that matches the new font list. If the routine finds a matching font list, it returns that font list and increments its reference count. Otherwise, the routine allocates space for the new font list and caches it. In either case, the application is responsible for managing the memory associated with the font list. When the application is done using the font list, it should be freed using XmFontListEntryFree().

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. Before a font list can be added to a font list, it has to be created with `XmFontListEntryCreate`() or `XmFontListEntryLoad`(). In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListAppendEntry`() returns a reference counted render table.

`XmFontListAppendEntry`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

XmFontListEntryCreate(1), XmFontListEntryFree(1), XmFontListEntryLoad(1), XmFontListFree(1), XmFontListRemoveEntry(1), XmRenderTableAddRenditions(1), XmRenditionCreate(1), XmRendition(2).

Name

XmFontListCopy – copy a font list.

Synopsis

XmFontList XmFontListCopy (XmFontList *fontlist*)

Inputs

fontlist Specifies the font list to be copied.

Returns

The new font list or NULL if fontlist is NULL.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListCopy`() makes and returns a copy of *fontlist*.

The routine searches the font list cache for the font list, returns the font list, and increments its reference count. The application is responsible for managing the memory associated with the font list. When the application is done using the font list, it should be freed using `XmFontListFree`().

Usage

A font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a font or a font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListCopy`() is a convenience routine which calls `XmRenderTableCopy`() to copy and return a reference counted render table.

`XmFontListCopy`() makes a correct copy of the font list regardless of the type of entries in the list.

When a font list is assigned to a widget, the widget makes a copy of the font list, so it is safe to free the font list. When you retrieve a font list from a widget using `XtGetValues`(), you should not alter the font list directly. If you need to make changes to the font list, use `XmFontListCopy`() to make a copy of the font list and then change the copy.

`XmFontListCopy`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

XmFontListFree(1), XmRenderTableCopy(1), XmRenditionCreate(1), XmRendition(2)

Name

XmFontListCreate – create a font list.

Synopsis

XmFontList XmFontListCreate (XFontStruct **font*, XmStringCharSet *charset*)

Inputs

font	Specifies the font structure.
charset	Specifies a tag that identifies the character set for the font.

Returns

The new font list or NULL if font or charset is NULL.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListCreate`() creates a new font list that contains a single entry with the specified *font* and *charset*. *charset* specifies the character set that is associated with the font. It can be XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

`XmFontListCreate`() searches the font list cache for a font list that matches the new font list. If the routine finds a matching font list, it returns that font list and increments its reference count. Otherwise, the routine allocates space for the new font list and caches it. In either case, the application is responsible for managing the memory associated with the font list. When the application is done using the font list, it should be freed using `XmFontListFree`().

Usage

A font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a font or a font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListCreate`() is a convenience routine which calls XmRenditionCreate() to create a rendition object for the font. The rendition object is added to a render table by the `XmRenderTableAddRenditions`() function. The render table is returned.

`XmFontListCreate`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

`XmFontListCreate`() is not multi-thread safe if the application has multiple application contexts. In Motif 2.1, the function `XmFontListCreate_r`() is to be preferred within multi-threaded applications.

Fonts must not be shared between displays in a multi-threaded environment.

See Also

`XmFontListAppendEntry`(1), `XmRenderTableAddRenditions`(1), `XmRenditionCreate`(1), `XmRendition`(2).

Name

XmFontListCreate_r – create a font list in a thread-safe manner.

Synopsis

XmFontList XmFontListCreate_r (XFontStruct **font*, XmStringCharSet *charset*, Widget *widget*)

Inputs

font	Specifies the font structure.
charset	Specifies a tag that identifies the character set for the font.
widget	Specifies a widget.

Returns

The new font list or NULL if font or charset is NULL.

Availability

Motif 2.1 and later.

Description

`XmFontListCreate_r`() is identical to `XmFontListCreate`(), except that it is multi-thread safe. The additional widget parameter is used to obtain a lock upon the application context associated with *widget*. The older routine `XmFontListCreate`() is not safe in threaded environments which have multiple application contexts.

Usage

The *widget* does not need to be the widget which uses font. It must be on the same display. The sharing of fonts or fontlists across multiple displays is not safe for multi-threaded applications.

Although the XmFontList is obsolete in Motif 2.0 and later, `XmFontListCreate_r`() is provided for backwards compatibility with applications, using the XmFontList interface, which are intended to run in multi-threaded environments. `XmFontListCreate_r`() should not be used in applications using the newer XmRendition and XmRenderTable interface.

See Also

`XmFontListCreate`(1), `XmRendition`(2).

Name

XmFontListEntryCreate – create a font list entry.

Synopsis

XmFontListEntry XmFontListEntryCreate (char *_tag_, XmFontType _type_, XtPointer _font_)

Inputs

tag	Specifies the tag for the font list entry.
type	Specifies the type of the font argument. Pass either XmFONT_IS_FONT, XmFONT_IS_FONTSET, or XmFONT_IS_XFT if support of Xft is enabled.
font	Specifies the font or font set.

Returns

A font list entry.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively. To maintain backwards compatibility, the XmFontList is re-implemented as a render table.

Description

`XmFontListEntryCreate`() makes a font list entry that contains the specified _font_, which is identified by _tag_. _type_ indicates whether _font_ specifies an XFontSet or a pointer to an XFontStruct. _tag_ is a NULL-terminated string that identifies the font list entry. It can have the value XmFONTLIST_DEFAULT_TAG, which identifies the default font list entry in a font list.

`XmFontListEntryCreate`() allocates space for the new font list entry. The application is responsible for managing the memory associated with the font list entry. When the application is done using the font list entry, it should be freed using `XmFontListEntryFree`().

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. `XmFontListEntryCreate`() creates a font list entry using an XFontStruct returned by `XLoadQueryFont`() or an XFontSet returned by `XCreateFontSet`(). The routine does not copy the font structure, so the XFontStruct or XFontSet must not be freed until all references to it have been freed. The font list entry can be added to a font list using `XmFontListAppendEntry`().

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryCreate`() returns a rendition object.

`XmFontListEntryCreate`() is not multi-thread safe if the application has multiple application contexts. In Motif 2.1, the function `XmFontListEntryCreate_r`() is to be preferred within multi-threaded applications.

Fonts must not be shared between displays in a multi-threaded environment.

`XmFontListEntryCreate`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

Example

The following code fragment shows how to create font list entries using XmFontListEntryCreate():

```
Widget          toplevel;
XFontStruct    *font1, *font2; /* Previously loaded
fonts */
XFontSet        fontset3;      /* Previously created
font sets */
XmFontListEntry entry1, entry2, entry3;
XmFontList      fontlist;

entry1 = XmFontListEntryCreate("tag1", XmFONT_IS_FONT,
font1);
entry2 = XmFontListEntryCreate("tag2", XmFONT_IS_FONT,
font2);
entry3 = XmFontListEntryCreate("tag3",
XmFONT_IS_FONTSET, fontset3);
fontlist = XmFontListAppendEntry (NULL, entry1);
fontlist = XmFontListAppendEntry (fontlist, entry2);
fontlist = XmFontListAppendEntry (fontlist, entry3);

/* Bug in Motif 1.2.1: see XmFontListEntryFree() */
#if ((XmVERSION == 1) && (XmREVISION == 2) &&
(XmUPDATE_LEVEL == 1))
   XtFree (entry1);
   XtFree (entry2);
   XtFree (entry3);
#else /* Motif 1.2.1 */
   XmFontListEntryFree (entry1);
   XmFontListEntryFree (entry2);
```

```
   XmFontListEntryFree (entry3);
#endif /* Motif 1.2.1 */

XtVaCreateManagedWidget ("widget_name", xmLabelWidget-
                         Class, toplevel, XmNfontList,
                         fontlist, NULL);
XmFontListFree (fontlist);
...
```

See Also

XmFontListAppendEntry(1), XmFontListEntryFree(1), XmFontListEntryCreate_r(1), XmFontListEntryGetFont(1), XmFontListEntryGetTag(1), XmFontListEntryLoad(1), XmFontListRemoveEntry(1), XmRenditionCreate(1), XmRendition(2).

Name

XmFontListEntryCreate_r – create a font list entry in a thread-safe manner.

Synopsis

```
XmFontListEntry XmFontListEntryCreate_r (  char        *tag,
                                           XmFontType  type,
                                           XtPointer   font,
                                           Widget      widget)
```

Inputs

tag	Specifies the tag for the font list entry.
type	Specifies the type of the font argument. Pass either XmFONT_IS_FONT, XmFONT_IS_FONTSET, or XmFONT_IS_XFT if support of Xft is enabled.
font	Specifies the font or font set.
widget	Specifies a widget.

Returns

A font list entry.

Availability

Motif 2.1 and later.

Description

`XmFontListEntryCreate_r`() is in all respects identical to `XmFontListEntryCreate`(), except that `XmFontListEntryCreate_r`() is provided for multi-threaded applications: the additional *widget* parameter is used to obtain a lock upon an application context. The older routine `XmFontListEntryCreate`() is not safe in threaded environments which have multiple application contexts.

Usage

The *widget* does not need to be the widget which uses font. It must be on the same display. The sharing of fonts or fontlists across multiple displays is not safe for multi-threaded applications.

Although the XmFontList is obsolete in Motif 2.0 and later, `XmFontListEntryCreate_r`() is provided for backwards compatibility with applications, using the XmFontList interface, which are intended to run in multi-threaded environments. `XmFontListEntryCreate_r`() should not be used in applications using the newer XmRendition and XmRenderTable interface.

See Also

`XmFontListEntryCreate`(1), `XmRendition`(2).

Name

XmFontListEntryFree – free the memory used by a font list entry.

Synopsis

void XmFontListEntryFree (XmFontListEntry **entry*)

Inputs

entry Specifies the address of the font list entry that is to be freed.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListEntryFree`() deallocates storage used by the specified font list *entry.* The routine does not free the XFontSet or XFontStruct data structure associated with the font list entry.

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. A font list entry can be created using `XmFontListEntryCreate`() or `XmFontListEntryLoad`() and then appended to a font list with `XmFontListAppendEntry`(). Once the entry has been appended to the necessary font lists, it should be freed using `XmFontListEntryFree`().

In Motif 1.2.1, there is a bug in `XmFontListEntryFree`() that causes it to free the font or font set, rather than the font list entry. As a workaround for this specific version, you can use `XtFree`() to free the font list entry.

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryFree`() is a simple convenience routine which calls `XmRenditionFree`().

`XmFontListEntryFree`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListAppendEntry`(1), `XmFontListEntryCreate`(1), `XmFontListEntryLoad`(1), `XmFontListNextEntry`(1), `XmFontListRemoveEntry`(1), `XmRenditionFree`(1), `XmRendition`(2).

Name

XmFontListEntryGetFont – get the font information from a font list entry.

Synopsis

XtPointer XmFontListEntryGetFont (XmFontListEntry *entry*, XmFontType **type_return*)

Inputs

entry Specifies the font list entry.

Outputs

type_return Returns the type of the font information that is returned. Valid types are XmFONT_IS_FONT, XmFONT_IS_FONTSET, or XmFONT_IS_XFT if support of Xft is enabled.

Returns

An XFontSet or a pointer to an XFontStruct.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListEntryGetFont`() retrieves the font information for the specified font list *entry*. When the font list *entry* contains a font, *type_return* is XmFONT_IS_FONT and the routine returns a pointer to an XFontStruct. When the font list *entry* contains a font set, *type_return* is XmFONT_IS_FONTSET and the routine returns the XFontSet. The XFontSet or XFontStruct that is returned is not a copy of the data structure, so it must not be freed by an application. When the font list entry contains an xft font, type_return is Xm_FONT_IS_XFT and the runtime returns the XftFont*.

Usage

The XmFontList and XmFontListEntry types are opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list entries and retrieve information about them. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListEntryGetFont`() can be used to get the font structure for a font list entry once it has been retrieved from the font list using `XmFontListNextEntry`().

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryGetFont`() is a convenience routine which fetches the

XmNfont and XmNfontType values of the rendition object represented by entry. The values are fetched through the function `XmRenditionRetrieve`(). type_return is set to the value of the XmNfontType resource, and the function `XmFontListEntryGetFont`() returns the value of the XmNfont resource of the rendition object.

`XmFontListEntryGetFont`()[1] is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListEntryCreate`(1), `XmFontListEntryGetTag`(1), `XmFontListEntryLoad`(1), `XmFontListNextEntry`(1), `XmRenditionRetrieve`(1), `XmRendition`(2).

1.Erroneously given as XmFontListGetFont() in 2nd edition.

Name

XmFontListEntryGetTag – get the tag of a font list entry.

Synopsis

char* XmFontListEntryGetTag (XmFontListEntry *entry*)

Inputs

entry Specifies the font list entry.

Returns

The tag for the font list entry.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListEntryGetTag`() retrieves the tag of the specified font list *entry*. The routine allocates storage for the tag string; the application is responsible for freeing the memory using `XtFree`().

Usage

The XmFontList and XmFontListEntry types are opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list entries and retrieve information about them. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListEntryGetTag`() can be used to get the tag of a font list entry once it has been retrieved from the font list using `XmFontListNextEntry`().

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryGetTag`() is a convenience routine which fetches and returns a copy of the XmNtag value of the rendition object represented by entry. The value is fetched through the function `XmRenditionRetrieve`().

`XmFontListEntryGetTag`()[1] is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListEntryCreate`(1), `XmFontListEntryGetFont`(1), `XmFontListEntryLoad`(1), `XmFontListNextEntry`(1), `XmRenditionRetrieve`(1), `XmRendition`(2).

1.Erroneously given as XmFontListGetTag() in 2nd edition.

Name

XmFontListEntryLoad – load a font or create a font set and then create a font list entry.

Synopsis

```
XmFontListEntry XmFontListEntryLoad (  Display     *display,
                                       char        *font_name,
                                       XmFontType  type,
                                       char        *tag)
```

Inputs

display Specifies a connection to an X server; returned from `XOpenDisplay`() or `XtDisplay`().

font_name Specifies an X Logical Font Description (XLFD) string.

type Specifies the type of font_name. Pass either XmFONT_IS_FONT, XmFONT_IS_FONTSET, or XmFONT_IS_XFT.

tag Specifies the tag for the font list entry.

Returns

A font list entry or NULL if the font cannot be found or the font set cannot be created.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListEntryLoad`() either loads a font or creates a font set depending on the value of *type* and then creates a font list entry that contains the font data and the specified *tag*. *font_name* is an XLFD string which is parsed as either a font name or a base font name list. *tag* is a NULL-terminated string that identifies the font list entry. It can have the value XmFONTLIST_DEFAULT_TAG, which identifies the default font list entry in a font list.

If *type* is set to XmFONT_IS_FONT, the routine uses the `XtCvtStringToFontStruct`() converter to load the font struct specified by font_name. If the value of *type* is XmFONT_IS_FONTSET, XmFontListEntryLoad uses the `XtCvtStringToFontSet`() converter to create a font set in the current locale.

`XmFontListEntryLoad`() allocates space for the new font list entry. The application is responsible for managing the memory associated with the font list entry. When the application is done using the font list entry, it should be freed using `XmFontListEntryFree`().

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. `XmFontListEntryLoad`() sets up the font data and creates a font list entry. The font list entry can be added to a font list using `XmFontListAppendEntry`().

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryLoad`() is a convenience routine which creates and returns a rendition object whose XmNfontName resource is set to font_name, and XmNfontType value is type. The rendition object is created with an XmNloadModel of XmLOAD_IMMEDIATE.

`XmFontListEntryLoad`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListAppendEntry`(1), `XmFontListEntryCreate`(1), `XmFontListEntryFree`(1), `XmFontListEntryGetFont`(1), `XmFontListEntryGetTag`(1), `XmFontListRemoveEntry`(1), `XmRenditionCreate`(1), `XmRendition`(2).

Name

XmFontListFree – free the memory used by a font list.

Synopsis

void XmFontListFree (XmFontList *fontlist*)

Inputs

fontlist Specifies the font list that is to be freed.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListFree`() deallocates storage used by the specified *fontlist*. The routine does not free the XFontSet or XFontStruct data structures associated with the font list.

Usage

A font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a font or a font set and an associated tag. `XmFontListFree`() frees the storage used by the font list but does not free the associated font data structures. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListFree`() is a convenience function which simply calls `XmRenderTableFree`().

It is important to call `XmFontListFree`() rather than `XtFree`() because Motif caches font lists. A call to `XmFontListFree`() decrements the reference count for the font list; the font list is not actually freed until the reference count reaches 0 (zero).

`XmFontListFree`() is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListAppendEntry`(1), `XmFontListCopy`(1), `XmFontListEntryFree`(1), `XmFontListRemoveEntry`(1), `XmRenderTableFree`(1).

Name

XmFontListFreeFontContext – free a font context.

Synopsis

void XmFontListFreeFontContext (XmFontContext *context*)

Inputs

context Specifies the font list context that is to be freed.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListFreeFontContext`() deallocates storage used by the specified font list *context*.

Usage

The XmFontList type is opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListFreeFontContext`() is the last of the three font context routines that an application should call when processing a font list, as it frees the font context data structure. An application begins by calling `XmFontListInitFontContext`() to create a font context and then makes repeated calls to `XmFontListNextEntry`() or `XmFontListGetNextFont`() to cycle through the font list.

`XmFontListFreeFontContext`() is retained for compatibility with Motif 1.2, and should not be used in newer applications.

See Also

`XmFontListGetNextFont`(1), `XmFontListInitFontContext`(1), `XmFontListNextEntry`(1), `XmRenderTableAddRendition`(1), `XmRenditionCreate`(1), `XmRendition`(2).

Name

XmFontListGetNextFont – retrieve information about the next font list element.

Synopsis

```
Boolean XmFontListGetNextFont (  XmFontContext     context,
                                 XmStringCharSet   *charset,
                                 XFontStruct       **font)
```

Inputs

context Specifies the font context for the font list.

Outputs

charset Returns the tag that identifies the character set for the font.

font Returns the font structure for the current font list element.

Returns

True if the values being returned are valid or False otherwise.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListGetNextFont`() returns the character set and font for the next element of the font list. *context* is the font context created by `XmFontListInitFontContext`(). The first call to `XmFontListGetNextFont`() returns the first font list element. Repeated calls to `XmFontListGetNextFont`() using the same *context* access successive font list elements. The routine returns False when it has reached the end of the font list.

Usage

A font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a font or a font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

If the routine is called with a font context that contains a font set, it returns the first font of the font set.

The XmFontList type is opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list. These routines use a XmFontContext to maintain an arbitrary position in a

font list. XmFontListGetNextFont() cycles through the fonts in a font list. XmFontListInitFontContext() is called first to create the font context. When an application is done processing the font list, it should call XmFontListFreeFontContext() with the same context to free the allocated data.

XmFontListGetNextFont() is retained for compatibility with Motif 1.2, and should not be used in newer applications.

See Also

XmFontListFreeFontContext(1), XmFontListInitFontContext(1), XmFontListNextEntry(1), XmRendition(2).

Name

XmFontListInitFontContext – create a font context.

Synopsis

Boolean XmFontListInitFontContext (XmFontContext **context*, XmFontList *fontlist*)

Inputs

fontlist Specifies the font list.

Outputs

context Returns the allocated font context structure.

Returns

True if the font context is allocated or False otherwise.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListInitFontContext`() creates a font context for the specified *fontlist*. This font context allows an application to access the information that is stored in the font list. `XmFontListInitFontContext`() allocates space for the font *context*. The application is responsible for managing the memory associated with the font context. When the application is done using the font *context*, it should be freed using `XmFontListFreeFontContext`().

Usage

The XmFontList type is opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListInitFontContext`() is the first of the three font context routines that an application should call when processing a font list, as it creates the font context data structure. The context is passed to `XmFontListNextEntry`() or `XmFontListGetNextFont`() to cycle through the font list. When an application is done processing the font list, it should call `XmFontListFreeFontContext`() with the same context to free the allocated data.

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

`XmFontListInitFontContext`() is retained for compatibility with Motif 1.2, and should not be used in newer applications.

See Also

`XmFontListFreeFontContext`(1), `XmFontListGetNextFont`(1), `XmFontListInitFontContext`(1), `XmFontListNextEntry`(1), `XmRendition`(2).

Name

XmFontListNextEntry – retrieve the next font list entry in a font list.

Synopsis

XmFontListEntry XmFontListNextEntry (XmFontContext *context*)

Inputs

context Specifies the font context for the font list.

Returns

A font list entry or NULL if the context refers to an invalid entry or if it is at the end of the font list.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListNextEntry`() returns the next font list entry in a font list. *context* is the font context created by `XmFontListInitFontContext`(). The first call to `XmFontListNextEntry`() returns the first entry in the font list. Repeated calls to `XmFontListNextEntry`() using the same *context* access successive font list entries. The routine returns NULL when it has reached the end of the font list.

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

The XmFontList and XmFontListEntry types are opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list entries and retrieve information about them. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListInitFontContext`() is called first to create the font context. `XmFontListNextEntry`() cycles through the font entries in a font list. `XmFontListEntryGetFont`() and `XmFontListEntryGetTag`() access the information in a font list entry. When an application is done processing the font list, it should call `XmFontListFreeFontContext`() with the same context to free the allocated data.

XmFontListNextEntry() is retained for compatibility with Motif 1.2, and should not be used in newer applications.

See Also

`XmFontListEntryFree`(1), `XmFontListEntryGetFont`(1), `XmFontListEntryGetTag`(1), `XmFontListFreeFontContext`(1), `XmFontListInitFontContext`(1), `XmRendition`(2).

Name

XmFontListRemoveEntry – remove a font list entry from a font list.

Synopsis

XmFontList XmFontListRemoveEntry (XmFontList *oldlist*, XmFontListEntry *entry*)

Inputs

oldlist	Specifies the font list from which entry is removed.
entry	Specifies the font list entry.

Returns

The new font list, oldlist if entry is NULL or no entries are removed, or NULL if oldlist is NULL.

Availability

In Motif 2.0 and later, the XmFontList and XmFontListEntry are obsolete. They are superseded by the XmRenderTable type and the XmRendition object respectively.

Description

`XmFontListRemoveEntry`() makes a new font list by removing any entries in *oldlist* that match the specified *entry.* The routine returns the new font list and deallocates *oldlist*. `XmFontListRemoveEntry`() does not deallocate the font list *entry*, so the application should free the storage using `XmFontListEntryFree`().

`XmFontListRemoveEntry`() searches the font list cache for a font list that matches the new font list. If the routine finds a matching font list, it returns that font list and increments its reference count. Otherwise, the routine allocates space for the new font list and caches it. In either case, the application is responsible for managing the memory associated with the font list. When the application is done using the font list, it should be freed using `XmFontListFree`().

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

An application can use `XmFontListRemoveEntry`() to remove a font list entry from a font list. If an application needs to process the font list to determine which entries to remove, it can use `XmFontListInitFontContext`() and `XmFontListNextEntry`() to cycle through the entries in the font list.

`XmFontListRemoveEntry`() is retained for compatibility with Motif 1.2, and should not be used in newer applications.

See Also

`XmFontListAppendEntry`(1), `XmFontListEntryCreate`(1), `XmFontListEntryFree`(1), `XmFontListEntryLoad`(1), `XmFontListFree`(1), `XmRendition`(2).

Name

XmGetAtomName – get the string representation of an atom.

Synopsis

#include <Xm/AtomMgr.h>

String XmGetAtomName (Display **display*, Atom *atom*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

atom Specifies the atom for the property name to be returned.

Returns

The string that represents atom.

Availability

In Motif 2.0 and later, `XGetAtomName`() is preferred.

Description

`XmGetAtomName`() returns the string that is used to represent a given *atom*. This routine works like Xlib's `XGetAtomName`() routine, but the Motif routine provides the added feature of client-side caching. `XmGetAtomName`() allocates space for the returned string; the application is responsible for freeing this storage using `XtFree`() when the atom is no longer needed.

Usage

An Atom is a number that identifies a property. Properties also have string names. XmGetAtomName() returns the string name specified in the original call to `XmInternAtom`() or `XInternAtom`(), or for predefined atoms, a string version of the symbolic constant without the XA_ attached.

In Motif 2.0 and later, `XmGetAtomName`() is no more than a convenience routine which calls `XGetAtomName`(). While `XmGetAtomName`() is not yet obsolete, `XGetAtomName`() is to be preferred.

See Also

`XmInternAtom`(1).

Name

XmGetColorCalculation – get the procedure that calculates default colors.

Synopsis

XmColorProc XmGetColorCalculation (void)

Returns

The procedure that calculates default colors.

Description

`XmGetColorCalculation`() returns the procedure that calculates the default foreground, top and bottom shadow, and select colors. The procedure calculates these colors based on the background color that is passed to the procedure.

Usage

Motif widgets rely on the use of shadowed borders to achieve their three-dimensional appearance. The top and bottom shadow colors are lighter and darker shades of the background color; these colors are reversed to make a component appear raised out of the screen or recessed into the screen. The select color is a slightly darker shade of the background color that indicates that a component is selected. The default foreground color is either black or white, depending on which color provides the most contrast with the background color. `XmGetColorCalculation`() returns the procedure that calculates these colors. Use `XmSetColorCalculation`() to change the calculation procedure.

In Motif 2.0 and later, color calculation procedures can be specified on a per-screen basis by specifying a value for the XmScreen object XmNcolorCalculationProc resource. Where a particular XmScreen does not have an assigned calculator, the procedure specified by `XmGetColorCalculation`() is used as the default.

Procedures

The XmColorProc has the following syntax:

```
typedef void (*XmColorProc) (   XColor  *bg_color,    /* specifies the
background color */
                                XColor  *fg_color,    /* returns the fore-
ground color */
                                XColor  *sel_color,   /* returns the select
color */
                                XColor  *ts_color,    /* returns the top
shadow color */
                                XColor  *bs_color)    /* returns the bot-
tom shadow color */
```

An XmColorProc takes five arguments. The first argument, *bg_color*, is a pointer to an XColor structure that specifies the background color. The red, green, blue, and pixel fields in the structure contain valid values. The rest of the arguments are pointers to XColor structures for the colors that are to be calculated. The procedure fills in the red, green, and blue fields in these structures.

See Also

`XmChangeColor`(1), `XmGetColors`(1), `XmSetColorCalculation`(1). `XmScreen`(2).

Name

XmGetColors – update the colors for a widget.

Synopsis

```
void XmGetColors (  Screen      *screen,
                    Colormap    color_map,
                    Pixel       background,
                    Pixel       *foreground_return,
                    Pixel       *top_shadow_return,
                    Pixel       *bottom_shadow_return,
                    Pixel       *select_return)
```

Inputs

screen Specifies the screen for which colors are to be allocated.

color_map Specifies a Colormap from which the colors are allocated.

background Specifies the background from which to calculate allocated colors.

Outputs

foreground_return Specifies an address into which the foreground Pixel is returned.

top_shadow_return Specifies an address into which the top shadow Pixel is returned.

bottom_shadow_return Specifies an address into which the bottom shadow Pixel is returned.

select_return Specifies an address into which the select Pixel is returned.

Description

`XmGetColors`() allocates and returns a set of pixels within a Colormap associated with a given *screen* for use as the foreground, top shadow, bottom shadow, and select colors of a widget. The returned values are calculated based upon a supplied background.

Usage

`XmGetColors`() allocates a set of pixels from a colormap. The pixels required are based upon a supplied background pixel. If any return address is specified as NULL, the relevant pixel is not allocated. In Motif 1.2 and earlier, pixels are allocated using the current color calculation procedure, which can be specified using `XmSetColorCalculation`(). In Motif 2.0 and later, per-screen color calculation procedures are supported: if the XmNcolorCalculationProc resource of the XmScreen object associated with screen is not NULL, the procedure specified by the resource is used to calculate the pixels. Otherwise, the current color calculation procedure is used.

See Also

XmGetColorCalculation(1), XmSetColorCalculation(1).
XmScreen(2).

Name

XmGetDestination – get the current destination widget.

Synopsis

Widget XmGetDestination (Display **display*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

Returns

The widget ID of the current destination widget or NULL if there is no current destination widget.

Description

`XmGetDestination`() returns the widget ID of the current destination widget for the specified *display*. The destination widget is usually the widget most recently changed by a select, edit, insert, or paste operation. `XmGetDestination`() identifies the widget that serves as the destination for quick paste operations and some clipboard routines. This routine returns NULL if there is no current destination, which occurs when no edit operations have been performed on a widget.

Usage

XmGetDestination() provides a way for an application to retrieve the widget that would be acted on by various selection operations, so that the application can do any necessary processing before the operation occurs.

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1).

Name

XmGetDragContext – get information about a drag and drop operation.

Synopsis

#include <Xm/DragDrop.h>

Widget XmGetDragContext (Widget *widget*, Time *timestamp*)

Inputs

widget Specifies a widget on the display where the drag and drop operation is taking place.

timestamp Specifies a timestamp that identifies a DragContext.

Returns

The ID of the DragContext object or NULL if no active DragContext is found.

Availability

Motif 1.2 and later.

Description

`XmGetDragContext`() retrieves the DragContext object associated with the display of the specified *widget* that is active at the specified *timestamp*. When more that one drag operation has been started on a display, a timestamp can uniquely identify the active DragContext. If the specified *timestamp* corresponds to a timestamp processed between the beginning and end of a single drag and drop operation, `XmGetDragContext`() returns the DragContext associated with the operation. If there is no active DragContext for the time-stamp, the routine returns NULL.

Usage

Motif 1.2 and later supports the drag and drop model of selection actions. Every drag and drop operation has a DragContext object associated with it that stores information about the drag operation. Both the initiating and the receiving clients use information in the DragContext to process the drag transaction. The DragContext object is widget-like, in that it uses resources to specify its attributes. These resources can be checked using `XtGetValues`() and modified using `XtSetValues`().

`XmGetDragContext`() provides a way for an application to retrieve a DragContext object. The application can then use `XtGetValues`() and `XtSetValues`() to manipulate the DragContext.

See Also

`XmDragCancel`(1), `XmDragStart`(1), `XmDragContext`(2).

Name

XmGetFocusWidget – get the widget that has the keyboard focus.

Synopsis

Widget XmGetFocusWidget (Widget *widget*)

Inputs

widget Specifies the widget whose hierarchy is to be traversed.

Returns

The widget ID of the widget with the keyboard focus or NULL if no widget has the focus.

Availability

Motif 1.2 and later.

Description

`XmGetFocusWidget`() returns the widget ID of the widget that has keyboard focus in the widget hierarchy that contains the specified *widget*. The routine searches the widget hierarchy that contains the specified widget up to the nearest shell ancestor. `XmGetFocusWidget`() returns the widget in the hierarchy that currently has the focus, or the widget that last had the focus when the user navigated to another hierarchy. If no widget in the hierarchy has the focus, the routine returns NULL.

Usage

XmGetFocusWidget() provides a means of determining the widget that currently has the keyboard focus, which can be useful if you are trying to control keyboard navigation in an application.

See Also

`XmGetTabGroup`(1), `XmGetVisibility`(1), `XmIsTraversable`(1), `XmProcessTraversal`(1).

Name

XmGetMenuCursor – get the current menu cursor.

Synopsis

Cursor XmGetMenuCursor (Display **display*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

Returns

The cursor ID for the current menu cursor or None if no cursor has been defined.

Availability

In Motif 1.2 and later, `XmGetMenuCursor`() is obsolete. It has been superseded by getting the Screen resource XmNmenuCursor.

Description

`XmGetMenuCursor`() returns the cursor ID of the menu cursor currently in use by the application on the specified display. The routine returns the cursor for the default screen of the display. If the cursor is not yet defined because the application called the routine before any menus were created, then XmGetMenuCursor() returns the value None.

Usage

The menu cursor is the pointer shape that is used whenever a menu is posted. This cursor can be different from the normal pointer shape. In Motif 1.2 and later, the new Screen object has a resource, XmNmenuCursor, that specifies the menu cursor. `XmGetMenuCursor`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

See Also

`XmSetMenuCursor`(1), `XmScreen`(2).

Name

XmGetPixmap – create and return a pixmap.

Synopsis

Pixmap XmGetPixmap (Screen **screen*, char **image_name*, Pixel *foreground*, Pixel *background*)

Inputs

screen Specifies the screen on which the pixmap is to be drawn.
image_name Specifies the string name of the image used to make the pixmap.
foreground Specifies the foreground color that is combined with the image when it is a bitmap.
background Specifies the background color that is combined with the image when it is a bitmap.

Returns

A pixmap on success or XmUNSPECIFIED_PIXMAP when the specified image_name cannot be found.

Description

`XmGetPixmap`() generates a pixmap, stores it in the pixmap cache, and returns its resource ID. Before the routine actually creates the pixmap, it checks the pixmap cache for a pixmap that matches the specified *image_name*, *screen*, *foreground*, and *background*. If a match is found, the reference count for the pixmap is incremented and the resource ID for the pixmap is returned. If no pixmap is found, `XmGetPixmap`() checks the image cache for a image that matches the specified *image_name*. If a matching image is found, it is used to create the pixmap that is returned.

When no matches are found, `XmGetPixmap`() begins a search for an X10 or X11 bitmap file, using *image_name* as the filename. If a file is found, its contents are read, converted into an image, and cached in the image cache. Then, the image is used to generate a pixmap that is subsequently cached and returned. The depth of the pixmap is the default depth of the *screen*. If *image_name* specifies a bitmap, the *foreground* and *background* colors are combined with the image. If no file is found, the routine returns XmUNSPECIFIED_PIXMAP.

Usage

When *image_name* starts with a slash (/), it specifies a full pathname and `XmGetPixmap`() opens the specified file. Otherwise, *image_name* specifies a filename which causes `XmGetPixmap`() to look for the file using a search path. In Motif 1.2 and earlier, the XBMLANGPATH environment variable specifies the search path for X bitmap files. In Motif 2.0 and later, the environment variables XMICONSEARCHPATH and XMICONBMSEARCHPATH specify search

paths for pixmap files: XMICONSEARCHPATH is used if a color server is running, XMICONBMSEARCHPATH otherwise, and XBMLANGPATH is used as a fallback.

The search path can contain the substitution character %B, where image_name is substituted for %B. The search path can also use the substitution characters accepted by `XtResolvePathname`(), where %T is mapped to bitmaps and %S is mapped to NULL.

If XBMLANGPATH is not set, `XmGetPixmap`() uses a default search path. If the XAPPLRESDIR environment variable is set, the routine searches the following paths:

%B	
$XAPPLRESDIR/%L/bitmaps/%N/%B	/usr/lib/X11/%L/bitmaps/%N/%B
$XAPPLRESDIR/%l_%t/bitmaps/%N/%B	/usr/lib/X11/%l_%t/bitmaps/%N/%B
$XAPPLRESDIR/%l/bitmaps/%N/%B	/usr/lib/X11/%l/bitmaps/%N/%B
$XAPPLRESDIR/bitmaps/%N/%B	/usr/lib/X11/bitmaps/%N/%B
$XAPPLRESDIR/%L/bitmaps/%B	/usr/lib/X11/%L/bitmaps/%B
$XAPPLRESDIR/%l_%t/bitmaps/%B	/usr/lib/X11/%l_%t/bitmaps/%B
$XAPPLRESDIR/%l/bitmaps/%B	/usr/lib/X11/%l/bitmaps/%B
$XAPPLRESDIR/bitmaps/%B	/usr/lib/X11/bitmaps/%B
	/usr/include/X11/bitmaps/%B
$HOME/bitmaps/%B	$HOME/%B

If XAPPLRESDIR is not set, `XmGetPixmap`() searches the same paths, except that XAPPLRESDIR is replaced by HOME. These search paths are vendor-dependent and a vendor may use different directories for /usr/lib/X11 and /usr/include/X11. In the search paths, the image name is substituted for %B, the class name of the application is substituted for %N, the language string of the display is substituted for %L, the language component of the language string is substituted for %l, and the territory string is substituted for %t.

See Also

`XmDestroyPixmap`(1), `XmGetPixmapByDepth`(1), `XmInstallImage`(1), `XmUninstallImage`(1).

Name

XmGetPixmapByDepth – create and return a pixmap of the specified depth.

Synopsis

```
Pixmap XmGetPixmapByDepth (Screen   *screen,
                           char     *image_name,
                           Pixel    foreground,
                           Pixel    background,
                           int      depth)
```

Inputs

screen Specifies the screen on which the pixmap is to be drawn.
image_name Specifies the string name of the image used to make the pixmap.
foreground Specifies the foreground color that is combined with the image when it is a bitmap.
background Specifies the background color that is combined with the image when it is a bitmap.
depth Specifies the depth of the pixmap.

Returns

A pixmap on success or XmUNSPECIFIED_PIXMAP when the specified image_name cannot be found.

Availability

Motif 1.2 and later.

Description

`XmGetPixmapByDepth`() generates a pixmap, stores it in the pixmap cache, and returns its resource ID. Before the routine actually creates the pixmap, it checks the pixmap cache for a pixmap that matches the specified *image_name*, *screen*, *foreground*, *background*, and *depth*. If a match is found, the reference count for the pixmap is incremented and the resource ID for the pixmap is returned. If no pixmap is found, `XmGetPixmapByDepth`() checks the image cache for a image that matches the specified *image_name*. If a matching image is found, it is used to create the pixmap that is returned.

When no matches are found, `XmGetPixmapByDepth`() begins a search for an X10 or X11 bitmap file, using *image_name* as the filename. If a file is found, its contents are read, converted into an image, and cached in the image cache. Then, the image is used to generate a pixmap that is subsequently cached and returned. The depth of the pixmap is the specified *depth*. If *image_name* specifies a bitmap, the foreground and background colors are combined with the image. If no file is found, the routine returns XmUNSPECIFIED_PIXMAP.

Usage

XmGetPixmapByDepth() works just like XmGetPixmap() except that the depth of the pixmap can be specified. With XmGetPixmap(), the depth of the returned pixmap is the default depth of the screen. See XmGetPixmap() for an explanation of the search path that is used to find the image.

See Also

XmDestroyPixmap(1), XmGetPixmap(1), XmInstallImage(1), XmUninstallImage(1).

Name

XmGetPostedFromWidget – get the widget that posted a menu.

Synopsis

#include <Xm/RowColumn.h>

Widget XmGetPostedFromWidget (Widget *menu*)

Inputs

menu Specifies the menu widget.

Returns

The widget ID of the widget that posted the menu.

Description

`XmGetPostedFromWidget`() returns the widget from which the specified *menu* is posted. The value that is returned depends on the type of menu that is specified. For a PopupMenu, the routine returns the widget from which *menu* is popped up. For a PulldownMenu, the routine returns the RowColumn widget from which *menu* is pulled down. For cascading submenus, the returned widget is the original RowColumn widget at the top of the menu system. For tear-off menus in Motif 1.2 and later, `XmGetPostedFromWidget`() returns the widget from which the menu is torn off.

Usage

If an application uses the same menu in different contexts, it can use `XmGet-PostedFromWidget`() in an activate callback to determine the context in which the menu callback should be interpreted.

See Also

`XmRowColumn`(2), `XmPopupMenu`(2), `XmPulldownMenu`(2).

Name

XmGetScaledPixmap – create and return a scaled pixmap.

Synopsis

```
Pixmap XmGetScaledPixmap (Widget   widget,
                          char     *image_name,
                          Pixel    foreground,
                          Pixel    background,
                          int      depth,
                          double   scaling_ratio)
```

Inputs

widget Specifies a widget.

image_name Specifies the string name of the image used to make the pixmap.

foreground Specifies the foreground color that is combined with the image when it is a bitmap.

background Specifies the background color that is combined with the image when it is a bitmap.

depth Specifies the depth of the pixmap.

scaling_ratio Specifies a scaling ratio applied to the pixmap.

Returns

A pixmap on success or XmUNSPECIFIED_PIXMAP when the specified image_name cannot be found.

Availability

Motif 2.1 and later.

Description

`XmGetScaledPixmap`() is similar to `XmGetPixmapByDepth`() except that the returned pixmap is scaled.

Usage

widget is used to find a PrintShell by wandering up the widget hierarchy, and secondly to find a Screen on which to create the pixmap. If *scaling_ratio* is zero and an ancestral PrintShell is found, the ratio applied is given by

(printer resolution / default pixmap resolution)

where the default pixmap resolution is the XmNdefaultPixmapResolution resource of the PrintShell, and the printer resolution is fetched by the PrintShell using Xp extensions to communicate with the XPrint server. The default value of the PrintShell XmNdefaultPixmapResolution resource is 100.

At present, any resolution specified within the pixmap file itself is currently ignored, although it is intended that this should take precedence over any PrintShell setting.
Although otherwise fully documented, the function does not have a functional prototype in any of the supplied public headers.

See Also

`XmDestroyPixmap`(1), `XmGetPixmapByDepth`(1), `XmPrintShell`(2).

Name

XmGetSecondaryResourceData – retrieve secondary widget resource data.

Synopsis

Cardinal XmGetSecondaryResourceData (WidgetClass *widget_class*,
XmSecondaryResourceData *******secondary_data_return*)

Inputs

widget_class Specifies the widget class.

Outputs

secondary_data_return Returns an array of XmSecondaryResourceData pointers.

Returns

The number of secondary resource data structures associated with the widget class.

Availability

Motif 1.2 and later.

Description

`XmGetSecondaryResourceData`() provides access to the secondary widget resource data associated with a widget class. Some Motif widget classes have resources that are not accessible with the functions `XtGetResourceList`() and `XtGetConstraintResourceList`(). If the specified *widget_class* has secondary resources, `XmGetSecondaryResourceData`() provides descriptions of the resources in one or more data structures and returns the number such structures. If the *widget_class* does not have secondary resources, the routine returns 0 (zero) and the value of *secondary_data_return* is undefined.
If the *widget_class* has secondary resources, `XmGetSecondaryResource-Data()` allocates an array of pointers to the corresponding data structures. The application is responsible for freeing the allocated memory using `XtFree`(). The resource list in each structure (the value of the resources field), the structures, and the array of pointers to the structures all need to be freed.

Usage

`XmGetSecondaryResourceData`()[1] only returns the secondary resources for a widget class if the class has been initialized. You can initialize a widget class by creating an instance of the class or any of its subclass. VendorShell and Text are two Motif widget classes that have secondary resources. The two fields in the XmSecondaryResourceData structure that are of interest to an application are resources and num_resources. These fields contain a list of the secondary resources and the number of such resources.

Most applications do not need to query a widget class for the resources it supports. `XmGetSecondaryResourceData`() is intended to support interface builders and applications like *editres* that allow a user to view the available resources and set them interactively. Use `XtGetResourceList`() and `XtGetConstraintResourceList`*()* to get the regular and constraint resources for a widget class.

Example

The following code fragment shows the use of `XmGetSecondaryResourceData`() to print the names of the secondary resources of the VendorShell widget:

```
XmSecondaryResourceData *res; Cardinal num_res, i,
j;

if (num_res = XmGetSecondaryResourceData (vendor-
                                          Shell-
                                          WidgetCla
                                          ss,
                                          &res)) {
   for (i = 0; i < num_res; i++) {
      for (j = 0; j < res[i]->num_resources; j++) {
         printf ("%s\n", res[i]-
         >resources[j].resource_name);
      }
      XtFree ((char*) res[i]->resources);
      XtFree ((char*) res[i]);
   }
   XtFree ((char*) res);
}
```

1.Erroneously given as XmGetSecondaryResources() in 1st and 2nd edition.

Structures

The XmSecondaryResourceData structure is defined as follows:

```
typedef struct {
    XmResourceBaseProc   base_proc;
    XtPointer            client_data;
    String               name;
    String               res_class;
    XtResourceList       resources;
    Cardinal             num_resources;
}XmSecondaryResourceDataRec, *XmSecondaryResourceData;
```

See Also

`VendorShell`(2), `XmText`(2).

Name

XmGetTabGroup – get the tab group for a widget.

Synopsis

Widget XmGetTabGroup (Widget *widget*)

Inputs

widget Specifies the widget whose tab group is to be returned.

Returns

The widget ID of the tab group of widget.

Availability

Motif 1.2 and later.

Description

`XmGetTabGroup`() returns the widget ID of the widget that is the tab group for the specified widget. If *widget* is a tab group or a shell, the routine returns *widget*. If *widget* is not a tab group and no ancestor up to the nearest shell ancestor is a tab group, the routine returns the nearest shell ancestor. Otherwise, `XmGetTabGroup`() returns the nearest ancestor of *widget* that is a tab group.

Usage

`XmGetTabGroup`() provides a way to find out the tab group for a particular widget in an application. A tab group is a group of widgets that can be traversed using the keyboard rather than the mouse. Users move from widget to widget within a single tab group by pressing the arrow keys. Users move between different tab groups by pressing the Tab or Shift-Tab keys. If the tab group widget is a manager, its children are all members of the tab group (unless they are made into separate tab groups). If the widget is a primitive, it is its own tab group. Certain widgets must not be included with other widgets within a tab group. For example, each List, ScrollBar, OptionMenu, or multi-line Text widget must be placed in a tab group by itself, since these widgets define special behavior for the arrow or Tab keys, which prevents the use of these keys for widget traversal.

See Also

`XmGetFocusWidget`(1), `XmGetVisibility`(1), `XmIsTraversable`(1), `XmProcessTraversal`(1), `XmManager`(2), `XmPrimitive`(2).

Name

XmGetTearOffControl – get the tear-off control for a menu.

Synopsis

#include <Xm/RowColumn.h>

Widget XmGetTearOffControl (Widget *menu*)

Inputs

menu Specifies the RowColumn widget whose tear-off control is to be returned.

Returns

The widget ID of the tear-off control or NULL if no tear-off control exists.

Availability

Motif 1.2 and later.

Description

`XmGetTearOffControl`() retrieves the widget ID of the widget that is the tear-off control for the specified *menu*. When the XmNtearOffModel resource of a RowColumn widget is set to XmTEAR_OFF_ENABLED for a PulldownMenu or a PopupMenu, the RowColumn creates a tear-off button for the menu. The tear-off button, which contains a dashed line by default, is the first element in the menu. When the button is activated, the menu is torn off. If the specified *menu* does not have a tear-off control, `XmGetTearOffControl`() returns NULL.

Usage

In Motif 1.2, a RowColumn that is configured as a PopupMenu or a PulldownMenu supports tear-off menus. When a menu is torn off, it remains on the screen after a selection is made so that additional selections can be made. The tear-off control is a button that has a Separator-like appearance. Once you retrieve the widget ID of the tear-off control, you can set resources to specify its appearance. You can specify values for the following resources: XmNbackground, XmNbackgroundPixmap, XmNbottomShadowColor, XmNforeground, XmNheight, XmNmargin, XmNseparatorType, XmNshadowThickness, and XmNtopShadowColor. You can also set these resources in a resource file by using the name of the control, which is TearOffControl.

See Also

`XmRepTypeInstallTearOffModelConverter`(1), `XmPopupMenu`(2), `XmPulldownMenu`(2), `XmRowColumn`(2), `XmSeparator`(2).

Name

XmGetVisibility – determine whether or not a widget is visible.

Synopsis

XmVisibility XmGetVisibility (Widget *widget*)

Inputs

widget Specifies the widget whose visibility state is to be returned.

Returns

XmVISIBILITY_UNOBSCURED if widget is completely visible, XmVISIBILITY_PARTIALLY_OBSCURED if widget is partially visible, XmVISIBILITY_FULLY_OBSCURED or if widget is not visible.

Availability

Motif 1.2 and later.

Description

`XmGetVisibility`() determines whether or not the specified *widget* is visible. The routine returns XmVISIBILITY_UNOBSCURED if the entire rectangular area of the widget is visible. It returns XmVISIBILITY_PARTIALLY_OBSCURED if a part of the rectangular area of the widget is obscured by its ancestors. `XmGetVisibility`() returns XmVISIBILITY_FULLY_OBSCURED if the widget is completely obscured by its ancestors or if it is not visible for some other reason, such as if it is unmapped or unrealized.

Usage

XmGetVisibility() provides a way for an application to find out the visibility state of a particular widget. This information can be used to help determine whether or not a widget is eligible to receive the keyboard focus. In order for a widget to receive the keyboard focus, it and all of its ancestors must not be in the process of being destroyed and they must be sensitive to input. The widget and its ancestors must also have their XmNtraversalOn resources set to True. If the widget is viewable, which means that it and its ancestors are managed, mapped, and realized and some part of the widget is visible, then the widget is eligible to receive the keyboard focus. A fully-obscured widget is not eligible to receive the focus unless part of it is within the work area of a ScrolledWindow with an XmNscrollingPolicy of XmAUTOMATIC that has an XmNtraverseObscuredCallback.

Structures

XmVisibility is defined as follows:

```
typedef enum {
    XmVISIBILITY_UNOBSCURED,
    XmVISIBILITY_PARTIALLY_OBSCURED,
    XmVISIBILITY_FULLY_OBSCURED
} XmVisibility;
```

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1), `XmIsTraversable`(1), `XmProcessTraversal`(1), `XmManager`(2), `XmScrolledWindow`(2).

Name

XmGetXmDisplay – get the Display object for a display.

Synopsis

#include <Xm/Display.h>

Widget XmGetXmDisplay (Display **display*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

Returns

The Display object for the display.

Availability

Motif 1.2 and later.

Description

`XmGetXmDisplay`() retrieves the Display object for the specified *display.*

Usage

In Motif 1.2, the Display object stores display-specific information for use by the toolkit. An application has a Display object for each display it accesses. When an application creates its first shell on a display, typically by calling `XtAppInitialize`() or `XtAppCreateShell`(), a Display object is created automatically. There is no way to create a Display independently. Use `XmGetXmDisplay`() to get the ID of the Display object, so that you can use `XtGetValues`() and `XtSetValues`() to access and modify Display resources.

See Also

`XmDisplay`(2), `XmScreen`(2).

Name

XmGetXmScreen – get the Screen object for a screen.

Synopsis

Widget XmGetXmScreen (Screen **screen*)

Inputs

screen Specifies a screen on a display; returned by `XtScreen`().

Returns

The Screen object for the screen.

Availability

Motif 1.2 and later.

Description

`XmGetXmScreen`() retrieves the Screen object for the specified *screen*.

Usage

In Motif 1.2, the Screen object stores screen-specific information for use by the toolkit. An application has a Screen object for each screen that it accesses. When an application creates its first shell on a screen, typically by calling `XtAppInitialize`() or `XtAppCreateShell`(), a Screen object is created automatically. There is no way to create a Screen independently. Use `XmGetXmScreen`() to get the ID of the Screen object, so that you can use `XtGetValues`() and `XtSetValues`() to access and modify Screen resources.

See Also

`XmDisplay`(2), `XmScreen`(2).

Name

XmHierarchyGetChildNodes – A Hierarchy function that returns the list of "node children" of the node widget.

Synopsis

#include <Xm/Hierarchy.h>

WidgetList XmHierarchyGetChildNodes(Widget *widget*, int **value_return*)

Inputs

widget	Specifies the Node widget whose list of node children is requested.

Outputs

value_return	Returns the list of "node children" of the node widget.

Description

`XmHierarchyGetChildNodes`() returns a list of widgets that name the node widget as their `XmNparentNode`; that is, it returns the list of "node children" of the node widget. Not that all the widgets are children (in the Xt sense) of the `XmHierarchy` or its subclass. The returned list is NULL if there are no node children.

Usage

`XmHierarchyGetChildNodes`() is a convenience routine that returns the list of "node children" of the node widget. The array of widgets returned is NULL-terminated. It should be freed with `XtFree()`.

See Also

`XmHierarchy`(2).

Name

XmHierarchyOpenAllAncestors – A Hierarchy function that opens all ancestors of a given node.

Synopsis

#include <Xm/Hierarchy.h>

void XmHierarchyOpenAllAncestors (Widget *widget*)

Inputs

widget Specifies the node widget that wishes to be shown.

Outputs

value_return Returns the current slider position for the Scale.

Description

`XmHierarchyOpenAllAncestors`() opens all ancestors of the given node, so that the node is displayed.

Usage

`XmHierarchyOpenAllAncestors`() is a convenience routine that opens all ancestors of the given node.

See Also

`XmHierarchy`(2).

Name

XmIconBoxIsCellEmpty – An IconBox function that determines whether a cell in the IconBox is empty.

Synopsis

#include <Xm/IconBox.h>

Boolean XmIconBoxIsCellEmpty (Widget *widget*,

Position *cell_x*

Position *cell_y*

Widget *ignore*)

Inputs

widget	Specifies the IconBox widget.
cell_x	Specifies the x location of the cell to check.
cell_y	Specifies the y location of the cell to check.
ignore	Specifies widget ID what will be ignored if it is present in the specified cell.

Outputs

value_return	Returns *True* if the specified cell contains no child, and *False* otherwise.

Description

`XmIconBoxIsCellEmpty`() determines whether a cell in the IconBox is empty.

Usage

`XmIconBoxIsCellEmpty`() is a convenience routine that determines whether a cell in the IconBox is empty. If the widget id specified by *ignore* is present in the specified cell, it will be ignored and the function will return *True*.

See Also

`XmIconBox(2`.

Name

XmIm– introduction to input methods.

Synopsis

Public Header:
<Xm/XmIm.h>

Functions/Macros:
XmImCloseXIM(), XmImFreeXIC(), XmImGetXIC(), XmImGetXIM(),
XmImMbLookupString(), XmImMbResetIC(), XmImRegister(),
XmImSetFocusValues(), XmImSetValues(), XmImSetXIC(),
XmImUnregister(), XmImUnsetFocus(), XmImVaSetFocusValues(),
XmImVaSetValues()

Availability

Motif 1.2 and later.

Description

Many languages are ideographic, and have considerably more characters than there are keys on the keyboard: the Ascii keyboard was not originally designed for languages that are not based upon the Latin alphabet. For such languages, in order to provide a mapping between the alphabet and the keyboard, it is necessary to represent particular characters by a key sequence rather than a single keystroke. An input method is the means by which X maps between the characters of the language, and the representative key sequences. The most common use of an input method is in implementing language-independent text widget input. As the user types the key sequences, the input method displays the actual keystrokes until the sequence completes a character, when the required character is displayed in the text widget. The process of composing a character from a key sequence is called pre-editing.

In order to facilitate pre-editing, the input method may maintain several areas on the screen: a status area, a pre-edit area, and an auxiliary area. The status area is an output-only window which provides feedback on the interaction with the input method. The pre-edit area displays the keyboard sequence as it is typed. The auxiliary area is used for popup menus, or for providing customized controls required by the particular input method. The location of the pre-edit area is determined by the XmNpreeditType resource of VendorShell. The value OnTheSpot displays the key sequence as it is typed into the destination text widget itself. OverTheSpot superimposes an editing window over the top of the text widget. OffTheSpot creates a dedicated editing window, usually at the bottom of the dia-

log. Root uses a pre-edit window which is a child of the root window of the display.
To control the interaction between the application and the input method, X defines a structure called an input context, which the programmer can fetch and manipulate where the need arises. Each widget registered with the input method has an associated input context, which may or may not be shared amongst the registered widgets. Motif extends the mechanisms provided by the lower level X libraries, and provides a caching mechanism whereby input contexts are shared between widgets.

Usage

Input methods are usually supplied by the vendors of the hardware, and the application generally connects to the input method without the need for any special coding by the programmer. The Motif widgets are fully capable of connecting to an input method when required, and although Motif provides a functional interface to enable the programmer to interact with an input method, the interface is not required for the Motif widgets. The exceptions are where the programmer is writing new widgets, or where internationalized input is required for the DrawingArea.

`XmImRegister`() registers a widget with an input method. `XmImSetValues`() manipulates an input context by registering callbacks which respond to specific states. `XmImSetFocusValues`() is similar, except that after the input context has been modified, the focus is reset to the widget providing the input. `XmImMbLookupString`() performs the necessary key sequence to character translation on behalf of the input widget. `XmImUnRegister`() unregisters the widget with the input method. Typically, `XmImRegister`() is called within the Initialize method of a widget, `XmImUnRegister`() is called by the Destroy method, and `XmImMbLookupString`() is called within an action or callback routine of the widget in response to an event. These are the primary functions which a programmer may need to call, and are all that are required to implement internationalized input for the Motif text widget.

Note that an input method does not need to support all styles of XmNpreeditType.

See Also

`XmImCloseXIM`(1), `XmImFreeXIC`(1), `XmImGetXIM`(1), `XmImGetXIC`(1), `XmImMbLookupString`(1), `XmImMbResetIC`(1), `XmImRegister`(1), `XmImSetFocusValues`(1), `XmImSetValues`(1), `XmImSetXIC`(1), `XmImUnregister`(1), `XmImUnsetFocus`(1), `XmImVaSetFocusValues`(1), `XmImVaSetValues(1).`

Name

XmImCloseXIM – close all input contexts.

Synopsis

#include <Xm/XmIm.h>

void XmImCloseXIM (Widget *widget*)

Inputs

widget Specifies a widget used to determine the display connection.

Availability

Motif 2.0 and later.

Description

`XmImCloseXIM`() is a convenience function which closes all input contexts associated with the current input method. The *widget* parameter is used to identify the XmDisplay object of the application.

Usage

`XmImCloseXIM`() uses the *widget* parameter to deduce the input method associated with the XmDisplay object. The application's connection to the input method is closed, and all widgets which are registered with any input context associated with the input method are unregistered. In order to close the input context associated with a single widget, rather than closing down all connections, use XmImUnregister().

The Motif widgets internally register and unregister themselves with the input manager using XmImRegister() and `XmImUnregister`() as required. The VendorShell calls `XmImCloseXIM`() within its Destroy method once the last VendorShell is destroyed in order to clean up the connection to the input method. An application which dynamically switches between input methods in a multi-language application may need to invoke `XmImCloseXIM`() because Motif only supports a single input method at any given instance. Application programmers will not normally need to use `XmImCloseXIM`() directly.

See Also

`XmImRegister`(1), `XmImUnregister`(1), `XmIm`(1).

Name

XmImFreeXIC – free an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImFreeXIC (Widget *widget*, XIC *xic*)

Inputs

widget	Specifies a widget from which the input context registry is deduced.
xic	Specifies the input context which is to be freed.

Availability

Motif 2.0 and later.

Description

`XmImFreeXIC`() is a convenience function which unregisters all widgets associated with the input context *xic*, and then frees the input context.

Usage

`XmImFreeXIC`() uses the *widget* parameter to deduce an ancestral VendorShell, from which the X input context registry is found. All widgets associated with the input context *xic* within the registry are unregistered, and the input context is freed.

See Also

`XmImGetXIC`(1), `XmImRegister`(1). `XmImSetXIC`(1), `XmImUnregister`(1), `XmIm`(1).

Name

XmImGetXIC – create an input context for a widget.

Synopsis

#include <Xm/XmIm.h>

XIC XmImGetXIC (Widget *widget*, XmInputPolicy *input_policy*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies a widget for which the input context is required.
input_policy	Specifies the policy for creating input contexts.
arglist	Specifies a list of arguments consisting of name/value pairs.
argcount	Specifies the number of arguments in arglist.

Returns

The input context associated with widget.

Availability

Motif 2.0 and later.

Description

`XmImGetXIC`() creates and registers a new input context for a widget, depending upon the *input_policy*. If *input_policy* is XmPER_WIDGET, a new input context is created for the widget. If the value is XmPER_SHELL, a new input context is created only if an input context associated with the ancestral shell of *widget* does not already exist, otherwise the widget is registered with the existing input context. If the policy is XmINHERIT_POLICY, the input policy is inherited by taking the value of the XmNinputPolicy resource from the nearest ancestral VendorShell. The set of attributes for the input context is specified through the resource list *arglist*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *argcount*. The name/value pairs are passed through to the function `XCreateIC`() if the input context is created. `XmImGetXIC`() returns either the input context which is newly created if the input policy is XmPER_WIDGET, otherwise it returns the shared context.

Usage

In Motif 1.2, the supported attributes for configuring the created input context are XmNbackground, XmNforeground, XmNbackgroundPixmap, XmNspotLocation, XmNfontList, and XmNarea.

In Motif 2.0 and later, the list is extended to include XmNpreeditCaretCallback, XmNpreeditDoneCallback, XmNpreeditDrawCallback, and XmNpreeditStartCallback resources.

You are referred to the `XCreateIC`() entry within the Xlib Reference Manual for the interpretation of each of the resource types. The function allocates storage associated with the created input context, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmImFreeXIC`().

Structures

The enumerated type XmInputPolicy has the following possible values:

XmINHERIT_POLICY
XmPER_WIDGET
XmPER_SHELL

See Also

`XmImFreeXIC`(1), `XmImSetXIC`(1), `XmIm`(1).

Name

XmImGetXIM – retrieve the input method for a widget.

Synopsis

#include <Xm/XmIm.h>

XIM XmImGetXIM (Widget *widget*)

Inputs

widget Specifies a widget registered with the input manager.

Returns

The input method associated with *widget*.

Availability

Motif 1.2 and later.

Description

`XmImGetXIM`() returns a pointer to an opaque data structure which represents the input method which the input manager has opened for the specified widget.

Usage

Widgets are normally registered with the input manager through a call to `XmIm-Register`(). If no input method is associated with the *widget*, the procedure uses any specified XmNinputMethod resource of the nearest ancestral VendorShell in order to open an input method. If the resource is NULL, the input method associated with the current locale is opened. If no input method can be opened, the function returns NULL.

`XmImGetXIM`() allocates storage for the opaque data structure which is returned, and it is the responsibility of the programmer to reclaim the space by a call to `XmImCloseXIM`() at a suitable point. `XmImGetXIM`() is not a procedure which an application programmer needs to use: the routine is of more use to the programmer of new widgets.

See Also

`XmImRegister`(1), `XmImCloseXIM`(1), `XmIm`(1).

Name

XmImMbLookupString – retrieve a composed string from an input method.

Synopsis

```
#include <Xm/XmIm.h>

int XmImMbLookupString ( Widget             widget,
                         XKeyPressedEvent   *event,
                         char               *buffer,
                         int                num_bytes,
                         KeySym             *keysym,
                         int                *status)
```

Inputs

widget	Specifies a widget registered with the input manager.
event	Specifies a key press event.
num_bytes	Specifies the length of the buffer array.

Outputs

buffer	Returns the composed string.
keysym	Returns any keysym associated with the input keyboard event.
status	Returns the status of the lookup.

Returns

The length of the composed string in bytes.

Availability

Motif 1.2 and later.

Description

`XmImMbLookupString`() translates an *event* into a composed character, and/or a keysym, using the input context associated with a given *widget*. Any composed string which can be deduced from the *event* is placed in *buffer*; the composed string consists of multi-byte characters in the encoding of the locale of the input context. If a keysym is associated with the *event*, this is returned at the address specified by *keysym*. The function returns the number of bytes placed into *buffer*.

Usage

A widget is registered with an input method through the function `XmImRegister`(). If no input context is associated with the *widget*, the function uses `XLookupString`() to map the key *event* into composed text. Otherwise the function calls `XmbLookupString`() with the input context as the first parameter. If the programmer is not interested in keysym values, a NULL value can be passed as the *keysym* parameter. `XmImMbLookupString`() places into *buffer* any composed character string associated with the key event: if the event at the given point in the input sequence does not signify a unique character in the language of the current locale, the function returns zero: subsequent key events may be required before a character is composed.

Structures

The possible values returned in *status* are the same as those returned from `XmbLookupString`(): you are referred to the Xlib Reference Manual for a full description and interpretation of the values.

```
XBufferOverflow     /* buffer size insufficient to hold composed sequence */
XLookupNone         /* no character sequence matching the input exists    */
XLookupChars        /* input characters were composed                     */
XLookupKeysym       /* input is keysym rather than composed character     */
XLookupBoth         /* both a keysym and composed character are returned */
```

See Also

`XmImRegister`(1), `XmIm`(1).

Name

XmImMbResetIC – reset an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImMbResetIC (Widget *widget*, char ***mb_text*)

Inputs

widget Specifies a widget registered with the Input Manager.

Outputs

mb_text Returns pending input on the input context.

Availability

Motif 2.0 and later.

Description

`XmImMbResetIC`() resets the input context associated with a widget.

Usage

`XmImMbResetIC`() is a convenience function which resets an input context to the initial state. The function is no more than a wrapper onto the function `XmbResetIC`(), which clears the pre-edit area and updates the status area of the input context. The return value of `XmbResetIC`() is placed into the address specified by *mb_text*. This data is implementation dependent, and may be NULL. If data is returned, the programmer is responsible for freeing it by calling `XFree`().

See Also

`XmImRegister`(1), `XmIm`(1).

Name

XmImRegister – register a widget with an Input Manager.

Synopsis

#include <Xm/XmIm.h>

void XmImRegister (Widget *widget*, unsigned int *reserved*)

Inputs

widget	Specifies a widget to register with the input manager.
reserved	This parameter is current unused.

Availability

Motif 1.2 and later.

Description

`XmImRegister`() is a convenience function which registers a widget with the input manager to establish a connection to the current input method. The function is called when an application needs to specially arrange for internationalized input to a widget.

Usage

The Motif widgets internally register themselves with the input manager as required. Only a programmer who is writing a new widget, or who requires internationalized input for the DrawingArea needs to call `XmImRegister`() directly. If the VendorShell ancestor containing the *widget* already has an associated input context, the function simply returns. Otherwise, the XmNinputPolicy resource of the nearest VendorShell ancestor is fetched to determine whether to share an existing input context. The function opens an input method by inspecting the XmNinputMethod resource of the VendorShell. If the resource is NULL, a default input method is opened using information from the current locale. `XmImRegister`() should not be called twice using the same *widget* parameter without unregistering the widget from the input method first.

The programmer is responsible for closing down the connection to the input method by calling `XmImUnregister`(). The Destroy method of the widget is an appropriate place to call this.

See Also

`XmImUnregister`(1), `XmIm`(1).

Name

XmImSetFocusValues – set the values and focus for an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImSetFocusValues (Widget *widget*, ArgList arglist, Cardinal *argcount*)

Inputs

widget	Specifies a widget registered with the input manager.
arglist	Specifies a list of resources consisting of name/value pairs.
argcount	Specifies the number of arguments in arglist.

Availability

Motif 1.2 and later.

Description

`XmImSetFocusValues`() notifies the input manager that a widget has received the input focus. If the previous values of the input context associated with the widget do not allow the context to be reused, the old context is unregistered, and a new one registered with the widget.

Usage

`XmImSetFocusValues`() is identical in all respects to `XmImSetValues`(), except that after the input context has been reset, the focus window attribute of the input context is set to the window of the input *widget*.

The Motif widgets invoke `XmImSetFocusValues`() as and when required. For example, the Text and TextField widgets automatically invoke XmImSetFocusValues() in response to FocusIn and EnterNotify events. A programmer who is implementing internationalized input for a DrawingArea or creating a new widget may need to call this function when the widget receives the input focus.

See Also

`XmImRegister`(1), `XmImSetValues`(1), `XmIm`(1).

Name

XmImSetValues – set the values for an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImSetValues (Widget *widget*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies a widget registered with the Input Manager.
arglist	Specifies a list of resources consisting of name/value pairs.
argcount	Specifies the number of arguments in arglist.

Availability

Motif 1.2 and later.

Description

`XmImSetValues`() sets the attributes for the input context associated with the specified *widget*. The set of attributes to be modified is specified through the resource list *arglist*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *argcount*.

Usage

`XmImSetValues`() is a convenience routine which invokes `XSetICValues`() in order to configure an input context. You are referred to the Xlib Reference Manual for the set of attributes supported by `XSetICValues`(), and for their interpretation.

The Motif widgets invoke `XmImSetValues`() as and when required. For example, the Text and TextField widgets automatically invoke `XmImSetValues`() when the widget is resized or the font changed. A programmer who is implementing internationalized input for a DrawingArea or creating a new widget may need to call this function when, for example, the widget needs to reconfigure the spot location.

See Also

`XmImSetFocusValues`(1), `XmImRegister`(1), `XmIm`(1).

Name

XmImSetXIC – register a widget with an existing input context.

Synopsis

#include <Xm/XmIm.h>

XIC XmImSetXIC (Widget *widget*, XIC *xic*)

Inputs

widget	Specifies a widget to be registered with the input context.
xic	Specifies an input context where the widget is to be registered.

Returns

The input context where the widget is registered.

Availability

Motif 2.0 and later.

Description

`XmImSetXIC`() is a convenience function which registers a *widget* with an input context. If the *widget* is registered with another input context, the *widget* is firstly unregistered with that context. The widget is then registered with the input context *xic*. If *xic* is NULL, the function creates a new input context and registers the widget with it. The function returns the input context where the widget is registered.

Usage

`XmImSetXIC`() allocates storage when it creates a new input context, and it is the responsibility of the programmer to free the space at an appropriate point by calling `XmImFreeXIC`().

See Also

`XmImFreeXIC`(1), `XmImRegister`(1), `XmIm`(1).

Name

XmImUnregister – unregister the input context for a widget.

Synopsis

#include <Xm/XmIm.h>

void XmImUnregister (Widget *widget*)

Inputs

widget Specifies a widget whose input context is to be unregistered.

Availability

Motif 1.2 and later.

Description

`XmImUnregister`() is a convenience function which unregisters the input context associated with a given *widget*. The function is the inverse of `XmImRegister`(), which is called when an application needs to specially arrange for internationalized input to a widget.

Usage

The Motif widgets internally register themselves with the input manager as required. Only a programmer who is writing a new widget, or who requires internationalized input for the DrawingArea needs to call `XmImRegister`() directly. Where `XmImRegister`() has been called by the application, it is the responsibility of the programmer to also call `XmImUnregister`(), usually within the `Destroy`() method of the widget for which internationalized input is required. `XmImUnregister`() uses the *widget* parameter to deduce the input method associated with a display connection. Any input context associated with the input method is unregistered.

See Also

`XmImRegister`(1), `XmIm`(1).

Name

XmImUnsetFocus – unset focus for input context.

Synopsis

#include <Xm/XmIm.h>

void XmImUnsetFocus (Widget *widget*)

Inputs

widget Specifies a widget which has lost the input focus.

Availability

Motif 1.2 and later.

Description

`XmImUnsetFocus`() notifies the input manager that a widget has lost the input focus.

Usage

`XmImUnsetFocus`() is a convenience routine which invokes `XUnsetICFocus`() using the input context associated with the specified *widget*. The input method is notified that no more input is expected from the widget.

The Motif widgets invoke `XmImUnsetFocus`() as and when required. For example, the Text and TextField widgets automatically invoke `XmImUnsetFocus`()[1] in response to FocusOut and LeaveNotify events. A programmer who is implementing internationalized input for a DrawingArea or creating a new widget may need to call this function when the widget loses the input focus.

See Also

`XmImSetFocusValues`(1), `XmImVaSetFocusValues`(1), `XmIm`(1).

1.Erroneously given as XmUnsetFocus() in 2nd edition.

Name

XmImVaSetFocusValues – set the values and focus for an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImVaSetFocusValues (Widget *widget*,....,NULL)

Inputs

widget	Specifies a widget registered with the Input Manager.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Availability

Motif 1.2 and later.

Description

`XmImVaSetFocusValues`() notifies the input manager that a *widget* has received the input focus. If the previous values of the input context associated with the *widget* do not allow the context to be reused, the old context is unregistered, and a new one registered with the widget.

Usage

`XmImVaSetFocusValues`() is simply a convenience routine with a variable length argument list which constructs internal arglist and argcount parameters to a `XmImSetFocusValues`() call.

See Also

`XmImSetFocusValues`(1).

Name

XmImVaSetValues – set the values for an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImVaSetValues (Widget *widget*,...,NULL)

Inputs

widget	Specifies a widget registered with the Input Manager.
...,NULL	A NULL-terminated variable-length list of resource name/value pairs.

Availability

Motif 1.2 and later.

Description

`XmImVaSetValues`()[1] sets the attributes for the input context associated with the specified *widget*.

Usage

`XmImVaSetValues`() is simply a convenience routine with a variable length argument list which constructs internal arglist and argcount parameters to a `XmImSetValues`() call.

See Also

`XmImSetValues`(1).

1.Erroneously given as XmImSetValues() in 2nd edition.

Name

XmInstallImage – install an image in the image cache.

Synopsis

Boolean XmInstallImage (XImage **image*, char **image_name*)

Inputs

image	Specifies the image to be installed.
image_name	Specifies the string name of the image.

Returns

True on success or False if *image* or *image_name* is NULL or *image_name* duplicates an image name already in the cache.

Description

`XmInstallImage`() installs the specified *image* in the image cache. The *image* can later be used to create a pixmap. When the routine installs the image, it does not make a copy of the image, so an application should not destroy the image until it has been uninstalled. The routine also expands the resource converter that handles images so that *image_name* can be used in a resource file. In order to allow references from a resource file, `XmInstallImage`() must be called to install an image before any widgets that use the image are created.

Usage

An application can use `XmInstallImage`() to install and cache images, so that the images can be shared throughout the application. Once an image is installed, it can be used to create a pixmap with `XmGetPixmap`(). The toolkit provides the following pre-installed images that can be referenced in a resource file or used to create a pixmap:

Image Name	Image Description
background	Solid background tile
25_foreground	A 25% foreground, 75% background tile
50_foreground	A 50% foreground, 50% background tile
75_foreground	A 75% foreground, 25% background tile
horizontal_tile	Horizontal lines tile, in Motif 1.2.3 and later.
vertical_tile	Vertical lines tile, in Motif 1.2.3 and later.
horizontal	As horizontal_tile: maintained for 1.2.2 compatibility.
vertical	As vertical_tile: maintained for 1.2.2 compatibility.
slant_right	Right slanting lines tile
slant_left	Left slanting lines tile

Image Name	Image Description
menu_cascade	An arrow pointing to the right, in Motif 2.0 and later.
menu_cascade_rtol	An arrow pointing to the left, in Motif 2.0 and later.
menu_checkmark	A tick mark, in Motif 2.0 and later.
menu_dash	A horizontal line, in Motif 2.0 and later.
collapsed	A filled arrow pointing to the right, in Motif 2.0 and later.
collapsed_rtol	A filled arrow pointing to the left, in Motif 2.0 and later.
expanded	A filled arrow pointing downwards, in Motif 2.0 and later.

Example

You might use the following code to define and install an image:

```
#define bitmap_width 16
#define bitmap_height 16

static char bitmap_bits[] = {
    0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
    0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
    0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF,
    0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF
};

static XImage ximage = {
    bitmap_width,       /* width */
    bitmap_height,      /* height */
    0,                  /* xoffset */
    XYBitmap,           /* format */
    bitmap_bits,        /* data */
    MSBFirst,           /* byte_order */
    8,                  /* bitmap_unit */
    LSBFirst,           /* bitmap_bit_order */
    8,                  /* bitmap_pad */
    1,                  /* depth */
    2,                  /* bytes_per_line */
    NULL                /* obdata */
};
...
XmInstallImage (&ximage, "image_name");
...
```

See Also

XmDestroyPixmap(1), XmGetPixmap(1), XmUninstallImage(1).

Name

XmInternAtom – return an atom for a given property name string.

Synopsis

#include <Xm/AtomMgr.h>

Atom XmInternAtom (Display **display*, String *name*, Boolean *only_if_exists*)

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
name	Specifies the string name of the property for which you want the atom.
only_if_exists	Specifies a Boolean value that indicates whether or not the atom is created if it does not exist.

Returns

An atom on success or None.

Availability

In Motif 2.0 and later, `XInternAtom`() is preferred.

Description

`XmInternAtom`() returns the atom that corresponds to the given property *name*. This routine works like Xlib's `XInternAtom`() routine, but the Motif routine provides the added feature of client-side caching. If no atom exists with the specified *name* and *only_if_exists* is True, `XmInternAtom`() does not create a new atom; it simply returns None. If *only_if_exists* is False, the routine creates the atom and returns it.

Usage

An atom is a number that identifies a property. Properties also have string names. XmInternAtom() returns the atom associated with a property if it exists, or it may create the atom if it does not exist. The atom remains defined even after the client that defined it has exited. An atom does not become undefined until the last connection to the X server closes. Predefined atoms are defined in <X11/Xatom.h> and begin with the prefix XA_. Predefined atoms do not need to be interned with `XmInternAtom`().

In Motif 2.0 and later, `XmInternAtom`() is no more than a convenience routine which calls `XInternAtom`(). While `XmInternAtom`() is not yet officially obsolete, `XInternAtom`() is to be preferred.

See Also

`XmGetAtomName`(1).

Name

XmIsMotifWMRunning – check whether the Motif Window Manager (*mwm*) is running.

Synopsis

Boolean XmIsMotifWMRunning (*Widget shell*)

Inputs

shell *Specifies the shell widget whose screen is queried.*

Returns

True if *mwm* is running or False *otherwise.*

Description

`XmIsMotifWMRunning`() checks for the presence of the _MOTIF_WM_INFO property on the root window of the screen of the specified *shell* to determine whether the Motif Window Manager (*mwm*) is running on the screen.

Usage

mwm defines additional types of communication between itself and client programs. This communication is optional, so an application should not depend on the communication or the presence of *mwm* for any functionality. `XmIsMotifWMRunning`() allows an application to check if *mwm* is running and act accordingly.

See Also

`mwm`(4).

Name

XmIs*Object* – determine whether a widget is a subclass of a class.

Synopsis

#include <Xm/Gadget.h>
Boolean XmIsGadget (Widget widget)

#include <Xm/Manager.h>
Boolean XmIsManager (Widget widget)

#include <Xm/Primitive.h>
Boolean XmIsPrimitive (Widget widget)

#include <Xm/ArrowB.h>
Boolean XmIsArrowButton (Widget widget)

#include <Xm/ArrowBG.h>
Boolean XmIsArrowButtonGadget (Widget widget)

#include <Xm/BulletinB.h>
Boolean XmIsBulletinBoard (Widget widget)

#include <Xm/CascadeB.h>
Boolean XmIsCascadeButton (Widget widget)

#include <Xm/CascadeBG.h>
Boolean XmIsCascadeButtonGadget (Widget widget)

#include <Xm/ComboBox.h>
Boolean XmIsComboBox (Widget widget)

#include <Xm/Command.h>
Boolean XmIsCommand (Widget widget)

#include <Xm/Container.h>
Boolean XmIsContainer (Widget widget)

#include <Xm/DialogS.h>
Boolean XmIsDialogShell (Widget widget)

#include <Xm/Display.h>
Boolean XmIsDisplay (Widget widget)

#include <Xm/DragC.h>
Boolean XmIsDragContext (Widget widget)

#include <Xm/DragIcon.h>
Boolean XmIsDragIconObjectClass (Widget widget)

#include <Xm/DrawingA.h>
Boolean XmIsDrawingArea (Widget widget)

#include <Xm/DrawnB.h>
Boolean XmIsDrawnButton (Widget widget)

#include <Xm/DropSMgr.h>
Boolean XmIsDropSiteManager (Widget widget)

#include <Xm/DropTrans.h>
Boolean XmIsDropTransfer (Widget widget)

#include <Xm/FileSB.h>
Boolean XmIsFileSelectionBox (Widget widget)

#include <Xm/Form.h>
Boolean XmIsForm (Widget widget)

#include <Xm/Frame.h>
Boolean XmIsFrame (Widget widget)

#include <Xm/GrabShell.h>
Boolean XmIsGrabShell (Widget widget)

#include <Xm/IconG.h>
Boolean XmIsIconGadget (Widget widget)

#include <Xm/Label.h>
Boolean XmIsLabel (Widget widget)

#include <Xm/LabelG.h>
Boolean XmIsLabelGadget (Widget widget)

#include <Xm/List.h>
Boolean XmIsList (Widget widget)

#include <Xm/MainW.h>
Boolean XmIsMainWindow (Widget widget)

#include <Xm/MenuShell.h>
Boolean XmIsMenuShell (Widget widget)

#include <Xm/MessageB.h>
Boolean XmIsMessageBox (Widget widget)

#include <Xm/Notebook.h>
Boolean XmIsNotebook (Widget widget)

#include <Xm/PanedW.h>
Boolean XmIsPanedWindow (Widget widget)

#include <Xm/PrintS.h>
Boolean XmIsPrintShell (Widget widget)

#include <Xm/PushB.h>
Boolean XmIsPushButton (Widget widget)

#include <Xm/PushBG.h>
Boolean XmIsPushButtonGadget (Widget widget)

#include <Xm/RowColumn.h>
Boolean XmIsRowColumn (Widget widget)

#include <Xm/Scale.h>
Boolean XmIsScale (Widget widget)

#include <Xm/Screen.h>
Boolean XmIsScreen (Widget widget)

#include <Xm/ScrollBar.h>
Boolean XmIsScrollBar (Widget widget)

#include <Xm/ScrolledW.h>
Boolean XmIsScrolledWindow (Widget widget)

#include <Xm/SelectioB.h>
Boolean XmIsSelectionBox (Widget widget)

#include <Xm/Separator.h>
Boolean XmIsSeparator (Widget widget)

#include <Xm/SeparatoG.h>
Boolean XmIsSeparatorGadget (Widget widget)

#include <Xm/Text.h>
Boolean XmIsText (Widget widget)

#include <Xm/TextF.h>
Boolean XmIsTextField (Widget widget)

#include <Xm/ToggleB.h>
Boolean XmIsToggleButton (Widget widget)

#include <Xm/ToggleBG.h>
Boolean XmIsToggleButtonGadget (Widget widget)

#include <Xm/VendorS.h>
Boolean XmIsVendorShell (Widget widget)

Inputs

widget Specifies the widget ID of the widget whose class is to be checked.

Returns

True if widget is of the specified class or False otherwise.

Availability

`XmIsDisplay`(), `XmIsDragContext`(), `XmIsDragIconObject-Class`(), `XmIsDropSiteManager`(), `XmIsDropTransfer`(), and `XmIsScreen`() are only available in Motif 1.2 and later.

`XmIsComboBox`(), `XmIsContainer`(), `XmIsNotebook`(), `XmIsIcon-Gadget`(), and `XmIsGrabShell`() are available in Motif 2.0 and later.

`XmIsPrintShell`() is available in Motif 2.1. Note that although the SpinBox class is available in Motif 2.0, and the SimpleSpinBox class in Motif 2.1, neither `XmIsSpinBox`() nor `XmIsSimpleSpinBox`() are defined. [1]

Description

The XmIs*() routines are macros that check the class of the specified widget. The macros returns True if widget is of the specified class or a subclass of the specified class. Otherwise, the macros return False.

Usage

An application can use the XmIs*() macros to check the class of a particular widget. All of the macros use `XtIsSubclass`() to determine the class of the widget.

Example

The missing macro `XmIsSpinBox`() could be defined as follows:

```
#include <Xm/SpinB.h>
#ifndef   XmIsSpinBox
#define   XmIsSpinBox(w) XtIsSubclass(w, xmSpinBoxWidgetClass)
#endif /* XmIsSpinBox */
```

1.Be warned that certain platforms, although they ship the PrintShell headers, do not compile the component into the native Motif toolkit. Sun Solaris is a case in point.

See Also

XmCreateObject(1), VendorShell(2), XmArrowButton(2), XmArrowButtonGadget(2), XmBulletinBoard(2), XmCascadeButton(2), XmCascadeButtonGadget(2), XmComboBox(2), XmCommand(2), XmContainer(2), XmDialogShell(2), XmDisplay(2), XmDragContext(2), XmDragIcon(2), XmDrawingArea(2), XmDrawnButton(2), XmDropSite(2), XmDropTransfer(2), XmFileSelectionBox(2), XmForm(2), XmFrame(2), XmGadget(2), XmGrabShell(2), XmIconGadget(2), XmLabel(2), XmLabelGadget(2), XmList(2), XmMainWindow(2), XmManager(2), XmMenuShell(2), XmMessageBox(2), XmNotebook(2), XmPanedWindow(2), XmPrimitive(2), XmPrintShell(2), XmPushButton(2), XmPushButtonGadget(2), XmRowColumn(2), XmScale(2), XmScreen(2), XmScrollBar(2), XmScrolledWindow(2), XmSelectionBox(2), XmSeparator(2), XmSeparatorGadget(2), XmSpinBox(2), XmSimpleSpinBox(2), XmText(2), XmTextField(2), XmToggleButton(2), XmToggleButtonGadget(2).

Name

XmIsTraversable – determine whether or not a widget can receive the keyboard focus.

Synopsis

Boolean XmIsTraversable (Widget *widget*)

Inputs

widget Specifies the widget whose traversability state is to be returned.

Returns

True if widget is eligible to receive the keyboard focus or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmIsTraversable`() determines whether or not the specified *widget* can receive the keyboard focus. The routine returns True if the *widget* is eligible to receive the keyboard focus; otherwise it returns False.

Usage

In order for a widget to receive the keyboard focus, it and all of its ancestors must not be in the process of being destroyed and they must be sensitive to input. The widget and its ancestors must also have their XmNtraversalOn resources set to True. If the widget is viewable, which means that it and its ancestors are managed, mapped, and realized and some part of the widget is visible, then the widget is eligible to receive the keyboard focus. A fully-obscured widget is not eligible to receive the focus unless part of it is within the work area of a ScrolledWindow with an XmNscrollingPolicy of XmAUTOMATIC that has an XmNtraverseObscuredCallback.

Primitive widgets and gadgets can receive the keyboard focus, while most manager widgets cannot, even if they have traversable children. However, some managers may be eligible to receive the keyboard focus under certain conditions. For example, a DrawingArea can receive the keyboard focus if it meets the conditions above and it does not have any children with the XmNtraversalOn resource set to True.

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1), `XmGetVisibility`(1), `XmProcessTraversal`(1), `XmManager`(2), `XmScrolledWindow`(2).

Name

XmListAddItem, XmListAddItems – add an item/items to a list.

Synopsis

#include <Xm/List.h>

void XmListAddItem (Widget *widget*, XmString *item*, int *position*)
void XmListAddItems (Widget *widget*, XmString **items*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be added.
items	Specifies a list of items that are to be added.
item_count	Specifies the number of items to be added.
position	Specifies the position at which to add the new item(s).

Description

`XmListAddItem`() inserts the specified *item* into the list, while `XmListAddItems`() inserts the specified list of *items*. If *item_count* is smaller than the number of items, only the first *item_count* items of the array are added. The *position* argument specifies the location of the new item(s) in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. An inserted item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListAddItem`() and `XmListAddItems`() are convenience routines that allow you to add items to a list. The routines add items to the list by internally manipulating the arrays of compound strings specified by the XmNitems, XmNitemCount, XmNselectedItems, and XmNselectedItemCount resources. If an item being added to the list duplicates an item that is already selected, the new item appears as selected. You should only use these routines if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to add items with these routines, you have to create a compound string for each item.

See Also

`XmListAddItemUnselected`(1), `XmListReplaceItems`(1), `XmListReplaceItemsPos`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplacePositions`(1), `XmList`(2).

Name

XmListAddItemUnselected, XmListAddItemsUnselected – add an item/items to a list.

Synopsis

#include <Xm/List.h>

void XmListAddItemUnselected (Widget *widget*, XmString *item*, int *position*)
void XmListAddItemsUnselected (Widget *widget*, XmString **items*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be added.
items	Specifies a list of items that are to be added.
item_count	Specifies the number of items to be added.
position	Specifies the position at which to add the new item(s).

Availability

`XmListAddItemsUnselected`() is only available in Motif 1.2 and later.

Description

`XmListAddItemUnselected`() inserts the specified *item* into the list, while `XmListAddItemsUnselected`() inserts the specified list of *items*. If *item_count* is smaller than the number of items, only the first *item_count* items of the array are added. The *position* argument specifies the location of the new item(s) in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. An inserted item does not appear selected, even if it matches an item in the XmNselectedItems list.

Usage

`XmListAddItemUnselected`() and `XmListAddItemsUnselected`() are convenience routines that allow you to add items to a list. These routines add items to the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If an item being added to the list duplicates an item that is already selected, the new item does not appear as selected. In order to add items with these routines, you have to create a compound string for each item.

Example

The following callback routine shows how to use of XmListAddItemUnselected() to insert an item into a list in alphabetical order:

```
void add_item (Widget      text_w,
```

```
                XtPointer   client_data,
                XtPointer   call_data)
{
   char       *text, *newtext = XmTextFieldGetString
   (text_w);
   XmString   str, *strlist;
   int        u_bound, l_bound = 0;
   Widget     list_w = (Widget) client_data;

   /* newtext is the text typed in the TextField
   widget */
   if (!newtext || !*newtext) {
      XtFree (newtext);
      return;
   }

   /* get the current entries (and number of entries)
   from the List */
   XtVaGetValues (list_w, XmNitemCount, &u_bound,
                          XmNitems, &strlist, NULL);

   u_bound--;

   /* perform binary search */
   while (u_bound >= l_bound) {
      int i = l_bound + (u_bound - l_bound)/2;

      text = (char *) XmStringUnparse (strlist[i],
                                       NULL,
                                       XmCHARSET_TEXT,
                                       XmCHARSET_TEXT,
                                       NULL, 0,
                                       XmOUTPUT_ALL);

      if (!text)
         break;

      if (strcmp (text, newtext) > 0)
         u_bound = i-1;
      else
         l_bound = i+1;

      XtFree (text);
   }

   /* insert item at appropriate location */
```

```
        str = XmStringCreateLocalized (newtext);
        XmListAddItemUnselected (list_w, str, l_bound+1);
        XmStringFree (str);
        XtFree (newtext);
    }
```

See Also

XmListAddItem(1), XmListReplaceItems(1), XmListReplaceItemsPos(1), XmListReplaceItemsPosUnselected(1), XmListReplaceItemsUnselected(1), XmListReplacePositions(1), XmList(2).

Name

XmListDeleteAllItems – delete all of the items from a list.

Synopsis

#include <Xm/List.h>

void XmListDeleteAllItems (Widget *widget*)

Inputs

widget Specifies the List widget.

Description

`XmListDeleteAllItems`() removes all of the items from the specified List *widget*.

Usage

`XmListDeleteAllItems`() is a convenience routine that allows you to remove all of the items from a list. The routine removes items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources.

See Also

`XmListDeleteItem`(1), `XmListDeleteItemsPos`(1), `XmListDeletePos`(1), `XmListDeletePositions`(1), `XmList`(2).

Name

XmListDeleteItem, XmListDeleteItems – delete an item/items from a list.

Synopsis

```
#include <Xm/List.h>

void XmListDeleteItem (Widget widget, XmString item)
void XmListDeleteItems (Widget widget, XmString *items, int item_count)
```

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be deleted.
items	Specifies a list of items that are to be deleted.
item_count	Specifies the number of items to be deleted.

Description

`XmListDeleteItem`()[1] removes the first occurrence of the specified *item* from the list, while `XmListDeleteItems`() removes the first occurrence of each of the elements of *items*. If an item does not exist, a warning message is displayed.

Usage

`XmListDeleteItem`() and `XmListDeleteItems`() are convenience routines that allow you to remove items from a list. The routines remove items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If there is more than one occurrence of an item in the list, the routines only remove the first occurrence. In order to remove items with these routines, you have to create a compound string for each item. The routines use a linear search to locate the items to be deleted.

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItemsPos`(1), `XmListDeletePos`(1), `XmListDeletePositions`(1), `XmList`(2).

1.Erroneously given as ListDeleteItem() in 1st and 2nd editions.

Name

XmListDeleteItemsPos – delete items starting at a specified position from a list.

Synopsis

#include <Xm/List.h>

void XmListDeleteItemsPos (Widget *widget*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
item_count	Specifies the number of items to be deleted.
position	Specifies the position from which to delete items.

Description

`XmListDeleteItemsPos`() removes *item_count* items from the list, starting at the specified *position*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. If the number of items between *position* and the end of the list is less than *item_count*, the routine deletes all of the items up through the last item in the list.

Usage

`XmListDeleteItemsPos`() is a convenience routine that allows you to remove items from a list. The routine removes items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. Since you are specifying the position of the items to be removed, you do not have to create compound strings for the items. The routine does not have to search for the items, so it avoids the linear search that is used by XmListDeleteItems().

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItem`(1), `XmListDeletePos`(1), `XmListDeletePositions`(1), `XmList`(2).

Name

XmListDeletePos – delete an item at the specified position from a list.

Synopsis

#include <Xm/List.h>

void XmListDeletePos (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position from which to delete an item.

Description

`XmListDeletePos`() removes the item at the specified *position* from the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. If the list does not have the specified *position*, a warning message is displayed.

Usage

`XmListDeletePos`() is a convenience routine that allows you to remove an item from a list. The routine removes items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. Since you are specifying the position of the item to be removed, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListDeleteItem`().

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItem`(1), `XmListDeleteItemsPos`(1), `XmListDeletePositions`(1), `XmList`(2).

Name

XmListDeletePositions – delete items at the specified positions from a list.

Synopsis

#include <Xm/List.h>

void XmListDeletePositions (Widget *widget*, int **position_list*, int *position_count*)

Inputs

widget	Specifies the List widget.
position_list	Specifies a list of positions from which to delete items.
position_count	Specifies the number of positions to be deleted.

Availability

Motif 1.2 and later.

Description

`XmListDeletePositions`() removes the items that appear at the positions specified in *position_list* from the list. A position value of 1 indicates the first item, a value of 2 indicates the second item, and so on. If the list does not have the specified position, a warning message is displayed. If *position_count* is smaller than the number of positions in *position_list*, only the first *position_count* items of the array are deleted.

Usage

`XmListDeletePositions`() is a convenience routine that allows you to remove items from a list. The routine remove the items by modifying the XmNitems and XmNitemCount resources. Since you are specifying the positions of the items to be removed, you do not have to create compound strings for the items. The routine does not have to search for the items, so it avoids the linear search that is used by `XmListDeleteItems`().

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItem`(1), `XmListDeleteItemsPos`(1), `XmListDeletePos`(1), `XmList`(2).

Name

XmListDeselectAllItems – deselect all items in a list.

Synopsis

#include <Xm/List.h>

void XmListDeselectAllItems (Widget *widget*)

Inputs

widget Specifies the List widget.

Description

`XmListDeselectAllItems`() unhighlights all of the selected items in the specified *widget* and removes these items from the XmNselectedItems list. If the list is in normal mode, the item with the keyboard focus remains selected; if the list is in add mode, all of the items are deselected.

Usage

`XmListDeselectAllItems`() is a convenience routine that allows you to deselect all of the items in a list. The routine deselects items in the list by internally manipulating the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. This routine does not invoke any selection callbacks for the list when the items are deselected.

See Also

`XmListDeselectItem`(1), `XmListDeselectPos`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListDeselectItem – deselect an item from a list.

Synopsis

#include <Xm/List.h>

void XmListDeselectItem (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be deselected.

Description

`XmListDeselectItem`() unhighlights and removes from the XmNselectedItems list the first occurrence of the specified *item*.

Usage

`XmListDeselectItem`() is a convenience routine that allows you to deselect an item in a list. The routine deselects items in the list by internally manipulating the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. This routine does not invoke any selection callbacks for the list when the item is deselected. If there is more than one occurrence of an item in the list, the routine only deselects the first occurrence. In order to deselect an item with this routine, you have to create a compound string for the item. The routine uses a linear search to locate the item to be deselected.

See Also

`XmListDeselectAllItems`(1), `XmListDeselectPos`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListDeselectPos – deselect an item at the specified position from a list.

Synopsis

#include <Xm/List.h>

void XmListDeselectPos (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position at which to deselect an item.

Description

`XmListDeselectPos`() unhighlights the item at the specified *position* in the list and removes the item from the XmNselectedItems list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. If the list does not have the specified *position*, the routine does nothing.

Usage

`XmListDeselectPos`() is a convenience routine that allows you to deselect an item in a list. The routine deselects items in the list by internally manipulating the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. This routine does not invoke any selection callbacks for the list when the item is deselected. Since you are specifying the position of the item to be deselected, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListDeselectItem`().

See Also

`XmListDeselectAllItems`(1), `XmListDeselectPos`(1), `XmListGetSelectedPos`(1), `XmListPosSelected`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListGetKbdItemPos – get the position of the item in a list that has the location cursor.

Synopsis

#include <Xm/List.h>

int XmListGetKbdItemPos (Widget *widget*)

Inputs

widget Specifies the List widget.

Returns

The position of the item that has the location cursor.

Availability

Motif 1.2 and later.

Description

`XmListGetKbdItemPos`() retrieves the position of the item in the specified List *widget* that has the location cursor. A returned value of 1 indicates the first item, a value of 2 indicates the second item, and so on. The value 0 (zero) specifies that the list is empty.

Usage

`XmListGetKbdItemPos`() provides a way to determine which item in a list has the keyboard focus. This information is useful if you need to perform actions based on the position of the location cursor in the list.

See Also

`XmListSetAddMode`(1), `XmListSetKbdItemPos`(1), `XmList`(2).

Name

XmListGetMatchPos – get all occurrences of an item in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListGetMatchPos (Widget *widget*, XmString *item*, int **position_list*, int **position_count*)

Inputs

widget	Specifies the List widget.
item	Specifies the item whose positions are to be retrieved.

Outputs

position_list	Returns a list of the positions of the item.
position_count	Returns the number of items in position_list.

Returns

True if the item is in the list or False otherwise.

Description

`XmListGetMatchPos`() determines whether the specified *item* exists in the list. If the list contains *item*, the routine returns True and *position_list* returns a list of positions that specify the location(s) of the *item*. A position value of 1 indicates the first item, a position value of 2 indicates the second item, and so on. `XmListGetMatchPos`() allocates storage for the *position_list* array when the item is found; the application is responsible for freeing this storage using `XtFree`(). If the list does not contain *item*, the routine returns False, and *position_count* is set to zero. In Motif 1.2.3 and earlier, the value of *position_list* is undefined if *item* is not within the list. From Motif 1.2.4 and later, *position_list* is set to NULL.

Usage

`XmListGetMatchPos`() is a convenience routine that provides a way to locate all of the occurrences of an item in a list. Alternatively, you could obtain this information yourself using the XmNitems resource and `XmListItemPos`().

Example

The following code fragments show the use of `XmListGetMatchPos`():

```
Widget    list_w;
int      *pos_list;
int       pos_cnt, i;
char     *choice = "A Sample Text String";
XmString str    = XmStringCreateLocalized (choice);
```

```
if (!XmListGetMatchPos (list_w, str, &pos_list,
&pos_cnt))
   XtWarning ("Can't get items in list");
else {
   printf ("%s exists at %d positions:", choice,
   pos_cnt);
   for (i = 0; i < pos_cnt; i++)
      printf (" %d", pos_list[i]);
   puts ("");
   XtFree (pos_list);
}
XmStringFree (str);
```

See Also

XmListGetSelectedPos(1), XmList(2).

Name

XmListGetSelectedPos – get the positions of the selected items in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListGetSelectedPos (Widget *widget*, int **position_list*, int *position_count*)

Inputs

widget	Specifies the List widget.

Outputs

position_list	Returns a list of the positions of the selected items.
position_count	Returns the number of items in position_list.

Returns

True if there are selected items in the list or False otherwise.

Description

`XmListGetSelectedPos`() determines whether there are any selected items in the list. If the list has selected items, the routine returns True and *position_list* returns a list of positions that specify the location(s) of the items. A position value of 1 indicates the first item, a position value of 2 indicates the second item, and so on. `XmListGetSelectedPos`() allocates storage for the *position_list* array when there are selected items; the application is responsible for freeing this storage using *XtFree*(). If the list does not contain any selected items, the routine returns False and *position_count* is set to zero. In Motif 1.2.3 and earlier, the value of *position_list* is undefined if there are no selected items within the list. From Motif 1.2.4 and later, *position_list* is set to NULL.

Usage

`XmListGetSelectedPos`() is a convenience routine that provides a way to determine the positions of all of the selected items in a list. Alternatively, you could obtain this information yourself using the XmNselectedItems resource and `XmListItemPos`().

See Also

`XmListGetMatchPos`(1), `XmList`(2).

Name

XmListItemExists – determine if a specified item is in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListItemExists (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the List widget.
item	Specifies the item whose presence in the list is checked.

Returns

True if the item is in the list or False otherwise.

Description

`XmListItemExists`() determines whether the list contains the specified *item*. The routine returns True if the *item* is present and False if it is not.

Usage

`XmListItemExists`() is a convenience routine that determines whether or not an item is in a list. In order to use the routine, you have to create a compound string for the item. The routine uses a linear search to locate the item. You may be able to obtain this information more effectively by searching the XmNitems list using your own search procedure.

See Also

`XmListGetMatchPos`(1), `XmListItemPos`(1), `XmList`(2).

Name

XmListItemPos – return the position of an item in a list.

Synopsis

#include <Xm/List.h>

int XmListItemPos (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the List widget.
item	Specifies the item whose position is returned.

Returns

The position of the item in the list or 0 (zero) if the *item* is not in the list.

Description

`XmListItemPos`() returns the position of the first occurrence of the specified *item* in the list. A position value of 1 indicates the first item, a position value of 2 indicates the second item, and so on. If item is not in the list, `XmListItem-Pos`() returns 0 (zero).

Usage

`XmListItemPos`() is a convenience routine that finds the position of an item in a list. If there is more than one occurrence of the item in the list, the routine only returns the position of the first occurrence. In order to use the routine, you have to create a compound string for the item. The routine uses a linear search to locate the item.

Example

The following routines show how to make sure that a given item in a list is visible:

```
void MakePosVisible (Widget list_w, int item_no)
{
   int top, visible;

   XtVaGetValues (list_w,  XmNtopItemPosition,
   &top,
                           XmNvisibleItemCount,
   &visible,
                           NULL);

   if (item_no < top)
      XmListSetPos (list_w, item_no);
   else if (item_no >= top+visible)
      XmListSetBottomPos (list_w, item_no);
```

```
}
void MakeItemVisible (Widget list_w, XmString item)
{
   int item_no = XmListItemPos (list_w, item);
   if (item_no > 0)
      MakePosVisible (list_w, item_no);
}
```

See Also

`XmListItemExists`(1), `XmListPosSelected`(1), `XmList`(2).

Name

XmListPosSelected – check if the item at a specified position is selected in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListPosSelected (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position that is checked.

Returns

True if the item is selected or False if the item is not selected or the *position* is invalid.

Availability

Motif 1.2 and later.

Description

`XmListPosSelected`() determines whether or not the list item at the specified *position* is selected. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. The value 0 (zero) specifies the last item in the list. The routine returns True if the list item is selected. It returns False if the item is not selected or the list does not have the specified *position*.

Usage

XmListPosSelected() is a convenience routine that lets you check if an item at a particular position is selected. Alternatively, you could check the list of positions returned by `XmListGetSelectedPos`() to see if the item at a position is selected.

See Also

`XmListDeselectPos`(1), `XmListGetSelectedPos`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListPosToBounds – return the bounding box of an item at the specified position in a list.

Synopsis

```
#include <Xm/List.h>

Boolean XmListPosToBounds (  Widget     widget,
                             int        position,
                             Position   *x,
                             Position   *y,
                             Dimension  *width,
                             Dimension  *height)
```

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item for which to return the bounding box.

Outputs

x	Returns the x-coordinate of the bounding box for the item.
y	Returns the y-coordinate of the bounding box for the item.
width	Returns the width of the bounding box for the item.
height	Returns the height of the bounding box for the item.

Returns

True if item at the specified position is visible or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmListPosToBounds`() returns the bounding box of the item at the specified *position* in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. The routine returns the *x* and *y* coordinates of the upper left corner of the bounding box in relation to the upper left corner of the List widget. `XmListPosToBounds`() also returns the *width* and *height* of the bounding box. Passing a NULL value for any of the *x*, *y*, *width*, or *height* parameters indicates that the value for the parameter should not be returned. If the item at the specified *position* is not visible, `XmListPosToBounds`() returns False and the return values are undefined.

Usage

XmListPosToBounds() provides a way to determine the bounding box of an item in a list. This information is useful if you want to perform additional event processing or draw special graphics for the list item.

See Also

XmListYToPos(1), XmList(2).

Name

XmListReplaceItems – replace specified items in a list.

Synopsis

```
#include <Xm/List.h>

void XmListReplaceItems (  Widget     widget,
                           XmString   *old_items,
                           int        item_count,
                           XmString   *new_items)
```

Inputs

widget	Specifies the List widget.
old_items	Specifies a list of the items that are to be replaced.
item_count	Specifies the number of items that are to be replaced.
new_items	Specifies a list of the new items.

Description

`XmListReplaceItems`() replaces the first occurrence of each item in the *old_items* list with the corresponding item from the *new_items* list. If an item in the *old_items* list does not exist in the specified List *widget*, the corresponding item in *new_items*[1] is skipped. If *item_count* is smaller than the number of *old_items* or *new_items*, only the first *item_count* items are replaced. A new item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItems`() is a convenience routine that allows you to replace particular items in a list. The routine replaces items by manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item appears as selected. You should only use this routine if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to replace items with this routine, you have to create compound strings for all of the old and new items. The routine uses a linear search to locate the items to be replaced.

See Also

`XmListAddItem`(1), `XmListAddItemUnselected`(1), `XmListReplaceItemsPos`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplaceItemsUnselected`(1), `XmListReplacePositions`(1), `XmList`(2).

1.Erroneously given as *new_list* in 1st and 2nd edition.

Name

XmListReplaceItemsPos – replace specified items in a list.

Synopsis

#include <Xm/List.h>

void XmListReplaceItemsPos (Widget *widget*, XmString **new_items*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
new_items	Specifies a list of the new items.
item_count	Specifies the number of items that are to be replaced.
position	Specifies the position at which to replace items.

Description

`XmListReplaceItemsPos`() replaces a consecutive number of items in the list with items from the *new_items* list. The first item that is replaced is located at the specified *position* and each subsequent item is replaced by the corresponding item from *new_items*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. If *item_count* is smaller than the number of *new_items*, only the first *item_count* items are replaced. If the number of items between *position* and the end of the list is less than *item_count*, the routine replaces all of the items up through the last item in the list. A new item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItemsPos`() is a convenience routine that allows you to replace a contiguous sequence of items in a list. The routine replaces items by manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item appears as selected. You should only use this routine if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to replace items with this routine, you have to create compound strings for all of the new items. The routine does not have to search for the items, so it avoids the linear searches that are used by `XmListReplaceItems`().

See Also

`XmListAddItem`(1), `XmListAddItemUnselected`(1), `XmListReplaceItems`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplaceItemsUnselected`(1), `XmListReplacePositions`(1), `XmList`(2).

Name

XmListReplaceItemsPosUnselected – replace specified items in a list.

Synopsis

```
#include <Xm/List.h>

void XmListReplaceItemsPosUnselected (  Widget      widget,
                                        XmString    *new_items,
                                        int         item_count,
                                        int         position)
```

Inputs

widget	Specifies the List widget.
new_items	Specifies a list of the new items.
item_count	Specifies the number of items that are to be replaced.
position	Specifies the position at which to replace items.

Availability

Motif 1.2 and later.

Description

`XmListReplaceItemsPosUnselected`() replaces a consecutive number of items in the list with items from the *new_items* list. The first item that is replaced is located at the specified *position* and each subsequent item is replaced by the corresponding item from *new_items*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. If *item_count* is smaller than the number of *new_items*, only the first *item_count* items are replaced. If the number of items between *position* and the end of the list is less than *item_count*, the routine replaces all of the items up through the last item in the list. A new item does not appear selected, even if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItemsPosUnselected`() is a convenience routine that allows you to replace a contiguous sequence of items in a list. The routine replaces items by modifying the array of compound strings specified through the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item does not appear as selected. In order to replace items with this routine, you have to create compound strings for all of the new items. The routine does not have to search for the items, so it avoids the linear searches that are used by `XmListReplaceItemsUnselected`().

See Also

XmListAddItem(1), XmListAddItemUnselected(1), XmListReplaceItems(1), XmListReplaceItemsPos(1), XmListReplaceItemsUnselected(1), XmListReplacePositions(1), XmList(2).

Name

XmListReplaceItemsUnselected – replace specified items in a list.

Synopsis

```
#include <Xm/List.h>

void XmListReplaceItemsUnselected (  Widget     widget,
                                     XmString   *old_items,
                                     int        item_count,
                                     XmString   *new_items)
```

Inputs

widget	Specifies the List widget.
old_items	Specifies a list of the items that are to be replaced.
item_count	Specifies the number of items that are to be replaced.
new_items	Specifies a list of the new items.

Availability

Motif 1.2 and later.

Description

`XmListReplaceItemsUnselected`() replaces the first occurrence of each item in the *old_items* list with the corresponding item from the *new_items* list. If an item in the *old_items* list does not exist in the specified List *widget*, the corresponding item in *new_items*[1] is skipped. If *item_count* is smaller than the number of *old_items* or *new_items*, only the first *item_count* items are replaced. A new item does not appear selected, even if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItemsUnselected`() is a convenience routine that allows you to replace particular items in a list. The routine replaces items by modifying the array of compound strings specified through the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item does not appear as selected. In order to replace items with this routine, you have to create compound strings for all of the old and new items. The routine uses a linear search to locate the items to be replaced.

See Also

`XmListAddItem(1), XmListAddItemUnselected(1), XmListReplaceItems(1), XmListReplaceItemsPos(1), XmListReplaceItemsPosUnselected(1),`

1.Erroneously given as *new_list* in 1st and 2nd editions.

XmListReplacePositions(1), XmList(2).

Name

XmListReplacePositions – replace items at the specified positions in a list.

Synopsis

#include <Xm/List.h>

void XmListReplacePositions (Widget *widget*, int **position_list*, XmString **item_list*, int *item_count*)

Inputs

widget	Specifies the List widget.
position_list	Specifies a list of positions at which to replace items.
item_list	Specifies a list of the new items.
item_count	Specifies the number of items that are to be replaced.

Availability

Motif 1.2 and later.

Description

`XmListReplacePositions`() replaces the items that appear at the positions specified in *position_list* with the corresponding items from *item_list*. A position value of 1 indicates the first item, a value of 2 indicates the second item, and so on. If the list does not have the specified position, a warning message is displayed. If *item_count* is smaller than the number of positions in *position_list*, only the first *item_count* items are replaced. A new item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListReplacePositions`() is a convenience routine that allows you to replace items at particular positions in a list. The routine replaces items by modifying the array of compound strings specified through the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item appears as selected. You should only use this routine if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to replace items with this routine, you have to create compound strings for all of the new items. The routine does not have to search for the items, so it avoids the linear searches that are used by `XmListReplaceItems`().

See Also

`XmListAddItem`(1), `XmListAddItemUnselected`(1), `XmListReplaceItems`(1), `XmListReplaceItemsPos`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplaceItemsUnselected`(1), `XmList`(2).

Name

XmListSelectItem – select an item from a list.

Synopsis

#include <Xm/List.h>

void XmListSelectItem (Widget *widget*, XmString *item*, Boolean *notify*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be selected.
notify	Specifies whether or not the selection callback is invoked.

Description

`XmListSelectItem`() highlights and selects the first occurrence of the specified *item* in the list. If the XmNselectionPolicy resource of the list is XmMULTIPLE_SELECT, the routine toggles the selection state of *item*. For any other selection policy, `XmListSelectItem`() replaces the currently selected item(s) with *item*. The XmNselectedItems resource specifies the current selection of the list. If notify is True, `XmListSelectItem`() invokes the selection callback for the current selection policy.

Usage

`XmListSelectItem`() is a convenience routine that allows you to select an item in a list. The routine selects the item by modifying the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. In order to select an item with this routine, you have to create a compound string for the item. The routine uses a linear search to locate the item to be selected. XmListSelectItem() only allows you to select a single item; there are no routines for selecting multiple items. If you need to select more than one item, use `XtSetValues`() to set XmNselectedItems and XmNselectedItemCount.

The *notify* parameter indicates whether or not the selection callbacks for the current selection policy are invoked. You can avoid redundant code by setting this parameter to True. If you are calling `XmListSelectItem`() from a selection callback routine, you probably want to set the parameter to False to avoid the possibility of an infinite loop. Calling `XmListSelectItem`() with *notify* set to True causes the callback routines to be invoked in a way that is indistinguishable from a user-initiated selection action.

See Also

`XmListDeselectAllItems`(1), `XmListDeselectItem`(1), `XmListDeselectPos`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListSelectPos – select an item at the specified position from a list.

Synopsis

#include <Xm/List.h>

void XmListSelectPos (Widget *widget*, int *position*, Boolean *notify*)

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item that is to be selected.
notify	Specifies whether or not the selection callback is invoked.

Description

`XmListSelectPos`() highlights and selects the item at the specified *position* in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. If the XmNselectionPolicy resource of the list is XmMULTIPLE_SELECT, the routine toggles the selection state of the item. For any other selection policy, `XmListSelectPos`() replaces the currently selected item with the specified item. The XmNselectedItems resource lists the current selection of the list. If notify is True, `XmListSelectPos`() invokes the selection callback for the current selection policy.

Usage

`XmListSelectPos`() is a convenience routine that allows you to select an item at a particular position in a list. The routine selects the item by modifying the array of compound strings specified through the XmNselectedItems and XmNselectedItemCount resources. Since you are specifying the position of the item to be selected, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListSelectItem`(). `XmListSelectPos`() only allows you to select a single item; there are no routines for selecting multiple items. If you need to select more than one item, use XtSetValues() to set XmNselectedItems and XmNselectedItemCount.

The *notify* parameter indicates whether or not the selection callbacks for the current selection policy are invoked. You can avoid redundant code by setting this parameter to True. If you are calling XmListSelectPos() from a selection callback routine, you probably want to set the parameter to False to avoid the possibility of an infinite loop. Calling `XmListSelectPos`() with *notify* set to True causes the callback routines to be invoked in a way that is indistinguishable from a user-initiated selection action.

See Also

XmListDeselectAllItems(1), XmListDeselectItem(1), XmListDeselectPos(1), XmListGetSelectedPos(1), XmListPosSelected(1), XmListSelectItem(1), XmList(2).

Name

XmListSetAddMode – set add mode in a list.

Synopsis

#include <Xm/List.h>

void XmListSetAddMode (Widget *widget*, Boolean *mode*)

Inputs

widget	Specifies the List widget.
mode	Specifies whether to set add mode on or off.

Description

`XmListSetAddMode`() sets the state of add mode when the XmNselectionPolicy is XmEXTENDED_SELECT. If *mode* is True, add mode is turned on; if *mode* is False, add mode is turned off. When a List widget is in add mode, the user can move the location cursor without disturbing the current selection.

Usage

XmListSetAddMode() provides a way to change the state of add mode in a list. The distinction between normal mode and add mode is only important for making keyboard-based selections. In normal mode, the location cursor and the selection move together, while in add mode, the location cursor and the selection can be separate.

See Also

`XmListGetKbdItemPos`(1), `XmListSetKbdItemPos`(1), `XmList`(2).

Name

XmListSetBottomItem – set the last visible item in a list.

Synopsis

#include <Xm/List.h>

void XmListSetBottomItem (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is made the last visible item.

Description

`XmListSetBottomItem`() scrolls the List *widget* so that the first occurrence of the specified *item* appears as the last visible item in the list.

Usage

`XmListSetBottomItem`() provides a way to make sure that a particular *item* is visible in a list. The routine changes the viewable portion of the list so that the specified *item* is displayed at the bottom of the viewport. If there is more than one occurrence of the *item* in the list, the routine uses the first occurrence. In order to use this routine, you have to create a compound string for the *item*. The routine uses a linear search to locate the *item*.

See Also

`XmListSetBottomPos`(1), `XmListSetHorizPos`(1), `XmListSetItem`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetBottomPos – set the last visible item in a list.

Synopsis

#include <Xm/List.h>

void XmListSetBottomPos (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item that is made the last visible item.

Description

`XmListSetBottomPos`() scrolls the List *widget* so that the item at the specified *position* appears as the last visible item in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list.

Usage

`XmListSetBottomPos`() provides a way to make sure that an item at a particular position is visible in a list. The routine changes the viewable portion of the list so that the item at the specified *position* is displayed at the bottom of the viewport. Since you are specifying the position of the item, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListSetBottomItem`().

Example

The following routine shows how to make sure that an item at a given position in a list is visible:

```
void MakePosVisible (Widget list_w, int item_no)
{
    int top, visible;

    XtVaGetValues (list_w, XmNtopItemPosition, &top, XmNvisibleItem-
    Count, &visible, NULL);

    if (item_no < top)
        XmListSetPos (list_w, item_no);
    else if (item_no >= top+visible)
        XmListSetBottomPos (list_w, item_no);
}
```

See Also

`XmListSetBottomItem`(1), `XmListSetHorizPos`(1), `XmListSetItem`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetHorizPos – set the horizontal position of a list.

Synopsis

#include <Xm/List.h>

void XmListSetHorizPos (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the horizontal position.

Description

`XmListSetHorizPos`() scrolls the list to the specified horizontal *position*. If XmNlistSizePolicy is set to XmCONSTANT or XmRESIZE_IF_POSSIBLE and the horizontal scroll bar is visible, `XmListSetHorizPos`() sets the XmNvalue resource of the horizontal scroll bar to the specified *position* and updates the visible area of the list.

Usage

When a list item is too long to fit horizontally inside the viewing area of a List widget, the widget either expands horizontally or adds a horizontal scroll bar, depending on the value of the XmNlistSizePolicy resource. Calling `XmListSetHorizPos`() is equivalent to the user moving the horizontal scroll bar to the specified location.

See Also

`XmListSetBottomItem`(1), `XmListSetBottomPos`(1), `XmListSetItem`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetItem – set the first visible item in a list.

Synopsis

#include <Xm/List.h>

void XmListSetItem (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is made the first visible item.

Description

`XmListSetItem`() scrolls the List *widget* so that the first occurrence of the specified *item* appears as the first visible item in the list.

Usage

`XmListSetItem`() provides a way to make sure that a particular *item* is visible in a list. The routine changes the viewable portion of the list so that the specified *item* is displayed at the top of the viewport. Using this routine is equivalent to setting the XmNtopItemPosition resource. If there is more than one occurrence of the *item* in the list, the routine uses the first occurrence. In order to use this routine, you have to create a compound string for the *item*. The routine uses a linear search to locate the *item*.

See Also

`XmListSetBottomItem`(1), `XmListSetBottomPos`(1), `XmListSetHorizPos`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetKbdItemPos – set the position of the location cursor in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListSetKbdItemPos (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position where the location cursor is set.

Returns

True on success or False if there is not item at position or the list is empty.

Availability

Motif 1.2 and later.

Description

`XmListSetKbdItemPos`() sets the location cursor at the specified *position*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. The routine does not check the selection state of the item at the specified location.

Usage

`XmListSetKbdItemPos`() provides a way to change which item in a list has the keyboard focus. The routine is useful if you need to make sure that particular item has the keyboard focus at a given time, such as when the list first receives the keyboard focus.

See Also

`XmListGetKbdItemPos`(1), `XmListSetAddMode`(1), `XmList`(2).

Name

XmListSetPos – sets the first visible item in a list.

Synopsis

#include <Xm/List.h>

void XmListSetPos (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item that is made the first visible item.

Description

`XmListSetPos`() scrolls the List widget so that the item at the specified *position* appears as the first visible item in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list.

Usage

`XmListSetPos`() provides a way to make sure that an item at a particular location is visible in a list. The routine changes the viewable portion of the list so that the item at the specified position is displayed at the top of the viewport. Using this routine is equivalent to setting the XmNtopItemPosition resource. Since you are specifying the position of the item, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListSetItem`().

Example

The following routine shows how to make sure that an item at a given position in a list is visible:

```
void MakePosVisible (Widget list_w, int item_no)
{
    int top, visible;

    XtVaGetValues (list_w, XmNtopItemPosition, &top, XmNvisibleItem-
    Count, &visible, NULL);

    if (item_no < top)
        XmListSetPos (list_w, item_no);
    else if (item_no >= top+visible)
        XmListSetBottomPos (list_w, item_no);
}
```

See Also

`XmListSetBottomItem(1), XmListSetBottomPos(1),`

XmListSetHorizPos(1), XmListSetItem(1), XmList(2).

Name

XmListUpdateSelectedList – update the list of selected items in a list.

Synopsis

#include <Xm/List.h>

void XmListUpdateSelectedList (Widget *widget*)

Inputs

widget Specified the List widget.

Availability

Motif 1.2 and later.

Description

`XmListUpdateSelectedList`() updates the array of compound strings specified through the XmNselectedItems resource. The routine frees the current selected array, and then traverses the array of compound strings specified by the XmNitems resource, adding each currently selected item to the XmNselectedItems list.

Usage

`XmListUpdateSelectedList`() provides a way to update the list of selected items in a list. This routine is useful if the actual items that are selected are not synchronized with the value of the XmNselectedItems resource. This situation might arise if you are using internal list functions and modifying internal data structures. If you are using the defined list routines, the situation should never occur.

See Also

`XmListDeselectAllItems`(1), `XmListDeselectItem`(1), `XmListDeselectPos`(1), `XmListGetSelectedPos`(1), `XmListPosSelected`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmList`(2).

Name

XmListYToPos – get the position of the item at the specified y-coordinate in a list.

Synopsis

#include <Xm/List.h>

int XmListYToPos (Widget *widget*, Position *y*)

Inputs

widget	Specifies the List widget.
y	Specifies the y-coordinate.

Returns

The position of the item at the specified y-coordinate.

Availability

Motif 1.2 and later.

Description

`XmListYToPos`() retrieves the position of the item at the specified *y*-coordinate in the list. The *y*-coordinate is specified in the coordinate system of the list. A returned value of 1 indicates the first item, a value of 2 indicates the second item, and so on. The value 0 (zero) specifies that there is no item at the specified location.

As of Motif 1.2, a return value of 0 (zero) indicates the first item, a value of 1 indicates the second item, and so on. In Motif 1.2.3 and earlier, the value that is returned may not be a valid position in the list, so an application should check the value with respect to the value of XmNitemCount before using it. In Motif 1.2.4 and later, the returned position may not exceed the value of XmNitemCount.

Usage

`XmListYToPos`() provides a way to translate a y-coordinate into a list position. This routine is useful if you are processing events that report a pointer position and you need to convert the location of the event into an item position.

See Also

`XmListPosToBounds`(1), `XmList`(2).

Name

XmMainWindowSep1, XmMainWindowSep2, XmMainWindowSep3 – get the widget ID of a MainWindow Separator.

Synopsis

```
#include <Xm/MainW.h>

Widget XmMainWindowSep1 (Widget widget)
Widget XmMainWindowSep2 (Widget widget)
Widget XmMainWindowSep3 (Widget widget)
```

Inputs

widget Specifies the MainWindow widget.

Returns

The widget ID of the particular MainWindow Separator.

Availability

In Motif 2.0 and later, these routines are marked as deprecated.

Description

`XmMainWindowSep1`() returns the widget ID of the MainWindow widget's first Separator, which is located directly below the MenuBar. `XmMainWindowSep2`() returns the widget ID of the second Separator in the Main Window, which is between the Command and ScrolledWindow widgets. `XmMainWindowSep3`() returns the widget ID of the MainWindow's third Separator, which is located just above the message window. The three Separator widgets in a MainWindow are visible only when the XmNshowSeparator resource is set to True.

Usage

`XmMainWindowSep1`(), `XmMainWindowSep2`(), and `XmMainWindowSep3`() provide access to the three Separator widgets that can be displayed by a MainWindow widget. With the widget IDs, you can change the visual attributes of the individual Separators.

In Motif 2.0 and later, the function `XtNameToWidget`() is the preferred method of obtaining the MainWindow components. You should pass *widget* as the first parameter, and "Separator1", "Separator2", or "Separator3" as the second parameter to this procedure.

See Also

`XmMainWindowSetAreas`(1), `XmMainWindow`(2), `XmScrolledWindow`(2).

Name

XmMainWindowSetAreas – specify the children for a MainWindow.

Synopsis

```
#include <Xm/MainW.h>

void XmMainWindowSetAreas ( Widget   widget,
                            Widget   menu_bar,
                            Widget   command_window,
                            Widget   horizontal_scrollbar,
                            Widget   vertical_scrollbar,
                            Widget   work_region)
```

Inputs

widget	Specifies the MainWindow widget.
menu_bar	Specifies the widget ID of the MenuBar.
command_window	Specifies the widget ID of the command window.
horizontal_scrollbar	Specifies the widget ID of the horizontal ScrollBar.
vertical_scrollbar	Specifies the widget ID of the vertical ScrollBar.
work_region	Specifies the widget ID of the work window.

Availability

In Motif 2.0 and later, the procedure is marked as deprecated.

Description

`XmMainWindowSetAreas`() sets up the standard regions of the MainWindow *widget* for an application. The MainWindow must be created before the routine is called. `XmMainWindowSetAreas`() specifies the MenuBar, the work window, the command window, and the horizontal and vertical ScrollBars for the MainWindow. If an application does not have one of these regions, the corresponding argument can be specified as NULL. Each region may have child widgets, and this routine determines which of those children will be actively managed by the MainWindow.

Usage

Each of the MainWindow regions is associated with a MainWindow resource; `XmMainWindowSetAreas`() sets the associated resources. The associated resources that correspond to the last five arguments to the routine are XmNmenuBar, XmNcommand, XmNhorizontalScrollBar, XmNverticalScrollBar, and XmNworkWindow. `XmMainWindowSetAreas`() does not provide a way to set up the message area; this region must be set up by specifying the XmNmessageWindow resource.

If an application does not call `XmMainWindowSetAreas`(), the widget may still set some of the standard regions. When a MenuBar child is added to a MainWindow, if XmNmenuBar has not been set, it is set to the MenuBar child. When a Command child is added to a MainWindow, if XmNcommand has not been set, it is set to the Command child. If ScrollBars are added as children, the XmNhorizontalScrollBar and XmNverticalScrollBar resources may be set if they have not already been specified. Any child that is not one of these types is used for the XmNworkWindow. If you want to be certain about which widgets are used for the different regions, it is wise to call `XmMainWindowSetAreas`() explicitly.

In Motif 2.0 and later, `XmMainWindowSetAreas`(), is deprecated. The programmer should use `XtSetValues`() in order to specify the XmNcommandWindow, XmNmenuBar, XmNworkWindow, XmNhorizontalScrollBar, and XmNverticalScrollBar resources of the MainWindow. `XmMainWindowSetAreas`() does not handle the XmNmessageWindow resource in any case.

Example

The following code fragment shows how to set some of the regions of a MainWindow:

```
Widget top, main_w, menubar, command_w, text_w, scrolled_text_w;
Arg     args[4];

main_w = XtVaCreateManagedWidget("main_w", xmMainWindowWidgetClass, top, NULL);
menubar = XmCreateMenuBar (main_w, "menubar", NULL, 0);
XtManageChild (menubar);

XtSetArg (args[0], XmNrows, 24);
XtSetArg (args[1], XmNcolumns, 80);
XtSetArg (args[2], XmNeditable, False);
XtSetArg (args[3], XmNeditMode, XmMULTI_LINE_EDIT);
text_w = XmCreateScrolledText (main_w, "text_w", args, 4);
XtManageChild (text_w);

scrolled_text_w = XtParent (text_w);
command_w = XmCreateText (main_w, "command_w", (Arg *) 0, 0);
XtManageChild (command_w);

#if     (XmVERSION > 1)
XtVaSetValues (main_w,
                XmNmenuBar,             menubar,
                XmNcommandWindow,       command_w,
                XmNhorizontalScrollBar, NULL,
                XmNverticalScrollBar,   NULL,
```

```
                XmNworkWindow,          scrolled_text_w,
                0);
#else /* XmVERSION > 1 */
XmMainWindowSetAreas (main_w, menubar, command_w, NULL, NULL,
scrolled_text_w);
#endif /* XmVERSION > 1 */
```

See Also

`XmMainWindowSep`(1), `XmMainWindow`(2), `XmScrolledWindow`(2).

Name

XmMapSegmentEncoding – get the compound text encoding format for a font list element tag.

Synopsis

char * XmMapSegmentEncoding (char *_fontlist_tag_)

Inputs

_fontlist_tag_ Specifies the compound string font list element tag.

Returns

A character string that contains a copy of the compound text encoding format or NULL if the font list element tag is not found in the registry.

Availability

Motif 1.2 and later.

Description

`XmMapSegmentEncoding`() retrieves the compound text encoding format associated with the specified _fontlist_tag_. The toolkit stores the mappings between compound text encodings and font list elements tags in a registry. `XmMapSegmentEncoding`() searches the registry for a compound text encoding format associated with the specified _fontlist_tag_ and returns a copy of the format. If _fontlist_tag_ is not in the registry, the routine returns NULL. `XmMapSegmentEncoding`() allocates storage for the returned character string; the application is responsible for freeing the storage using XtFree().

Usage

Compound text is an encoding that is designed to represent text from any locale. Compound text strings identify their encoding using embedded escape sequences. The compound text representation was standardized for X11R4 for use as a text interchange format for interclient communication.

`XmCvtXmStringToCT`() converts a compound string into compound text by using the font list tag of each compound string segment to select a compound text format from the registry for the segment. `XmMapSegmentEncoding`() provides a way for an application to determine the compound text format that would be used for a particular font list element tag.

See Also

`XmCvtXmStringToCT`(1), `XmRegisterSegmentEncoding`(1).

Name

XmMenuPosition – position a popup menu.

Synopsis

#include <Xm/RowColumn.h>

void XmMenuPosition (Widget *menu*, XButtonPressedEvent **event*)

Inputs

menu Specifies the PopupMenu.

event Specifies the event that was passed to the action procedure managing the PopupMenu.

Description

`XmMenuPosition`() positions a popup menu, using the values of the x_root and y_root fields from the specified *event*. An application must call this routine before managing the popup menu, except when the application is positioning the menu itself.

Usage

The *event* parameter for `XmMenuPosition`() is defined to be of type XButtonPressedEvent *; using another type of event might lead to toolkit problems. The x_root and y_root fields in the *event* structure are used to position the menu at the location of the mouse button press. You can modify these fields to position the menu at another location.

In Motif 2.0 and later, a menu whose XmNpopupEnabled resource is XmPOPUP_AUTOMATIC or XmPOPUP_AUTOMATIC_RECURSIVE has an installed event handler which calls `XmMenuPosition`() directly without the need for an application to intervene in posting the menu.

Example

The following routine shows the use of an event handler to post a popup menu.

```
void PostIt (Widget w, XtPointer client_data, XEvent *event, Boolean *dispatch)
{
    Widget                popup = (Widget) client_data;
    XButtonPressedEvent   *bevent = (XButtonPressedEvent *) event;

    if ((bevent->type != ButtonPress) && (bevent->button != 3))
        return;

    XmMenuPosition (popup, bevent);
    XtManageChild (popup);
}
```

```
...
extern Widget some_widget;      /* Where the menu is posted */
extern Widget my_menu;          /* The menu to post */

XtAddEventHandler(some_widget, ButtonPressMask, False, PostIt, (XtPointer)
my_menu) ;
```

See Also

XmRowColumn(2), XmPopupMenu(2).

Name

XmMessageBoxGetChild – get the specified child of a MessageBox widget.

Synopsis

#include <Xm/MessageB.h>

Widget XmMessageBoxGetChild (Widget *widget*, unsigned char *child*)

Inputs

widget Specifies the MessageBox widget.
child Specifies the child of the MessageBox widget. Pass one of the values from the list below.

Returns

The widget ID of the specified child of the MessageBox.

Availability

As of Motif 2.0, the toolkit abstract child fetch routines are marked for deprecation. You should give preference to `XtNameToWidget`(), except when fetching the MessageBox default button.

Description

`XmMessageBoxGetChild`() returns the widget ID of the specified *child* of the MessageBox *widget*.

Usage

The *child* values XmDIALOG_CANCEL_BUTTON, XmDIALOG_HELP_BUTTON, and XmDIALOG_OK_BUTTON specify the action buttons in the *widget*. A *child* value of XmDIALOG_DEFAULT_BUTTON specifies the current default button. The value XmDIALOG_SYMBOL_LABEL specifies the label used to display the message symbol, while XmDIALOG_MESSAGE_LABEL specifies the message label. XmDIALOG_SEPARATOR specifies the separator that is positioned between the message and the action buttons. For more information on the different children of the MessageBox, see the manual page in Section 2, *Motif and Xt Widget Classes*.

Widget Hierarchy

As of Motif 2.0, most Motif composite child fetch routines are marked as deprecated. However, since it is not possible to fetch the XmDIALOG_DEFAULT_BUTTON child using a public interface except through `XmMessageBoxGetChild`(), the routine should not be considered truly deprecated. For consistency with the preferred new style, when fetching all other child values, consider giving preference to the Intrinsics routine `XtNameToWidget`(), passing one of the following names as the second parameter:

“Cancel”	(XmDIALOG_CANCEL_BUTTON)
“OK”	(XmDIALOG_OK_BUTTON)
“Separator”	(XmDIALOG_SEPARATOR)
“Help”	(XmDIALOG_HELP_BUTTON)
“Symbol”	(XmDIALOG_SYMBOL_LABEL)
“Message”	(XmDIALOG_MESSAGE_LABEL)

Structures

The possible values for child are:

XmDIALOG_CANCEL_BUTTON	XmDIALOG_OK_BUTTON
XmDIALOG_DEFAULT_BUTTON	XmDIALOG_SEPARATOR
XmDIALOG_HELP_BUTTON	
XmDIALOG_SYMBOL_LABEL	
XmDIALOG_MESSAGE_LABEL	

See Also

`XmBulletinBoard`(2), XmBulletinBoardDialog(2), `XmErrorDialog`(2), `XmInformationDialog`(2), `XmManager`(2), `XmMessageBox`(2), `XmMessageDialog`(2), `XmQuestionDialog`(2), `XmTemplateDialog`(2), `XmWarningDialog`(2), `XmWorkingDialog`(2).

Name

XmMultiListDeselectItems–clear the selection state of items of a MultiList.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListDeselectItems (Widget *widget*, XmString *item*, int *column*)

Inputs

widget	Specifies the MultiList widget.
item	Specifies XmString to use as selection key.
column	Specifies a column number to match.

Description

XmMultiListDeselectItems() clears the selection state of the MultiList widget by matching column entries to *item*.

Usage

The *column* specifies the column number in the MultiList starting from 0 or XmANY_COLUMN.

See Also

`XmMultiList`(2).

Name

XmMultiListDeselectRow – clear the selection state of a specified row of a MultiList widget.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListDeselectRow (Widget *widget*, int *row*)

Inputs

widget	Specifies the MultiList widget.
child	Specifies a row to select.

Description

XmMultiListDeselectRow() clears the selection state of the row of a MultiList widget.

Usage

The *row* specifies the row number in the MultiList starting from 0.

See Also

`XmMultiList`(2).

Name

XmMultiListGetSelectedRowArray – returns NULL-terminated array of pointers to selected rows of MultiList.

Synopsis

#include <Xm/MultiList.h>

int * XmMultiListGetSelectRowArray (Widget *widget*, int **num_rows*)

Inputs

widget Specifies the MultiList widget.

Outputs

num_row Returns the number of returned rows.

Returns

NULL-terminated array of pointers to selected rows of MultiList.

Description

`XmMultiListGetSelectedRowArray`() returns NULL_terminated array of pointers to selected rows of MultiList. `XmMultiListGetSelectedRowArray`() allocated storage for the returned array; the application is responsible for freeing this storage using `XtFree`().

Usage

`XmMultiListGetSelectedRowArray`() is a convenience routine that provides a way to obtain numbers of selected rows in a list.

See Also

`XmMultiList`(2).

Name

XmMultiListGetSelectedRows — An Extended List function that returns the rows that currently are selected.

Synopsis

```
#include <Xm/MultiList.h>
XmMultiListRowInfo ** XmMultiListGetSelectedRows(Widget widget);
```

Inputs

widget	Specifies the ID of the Extended list widget.

Outputs

Xm18RowInfo **	Contains a NULL terminated array of Xm18RowInfo pointers.

Availability

Motif 2.2 and later.

Description

This function return an a NULL terminated array of Xm18RowInfo pointers. The calling routine is responsible for freeing the returned pointer with XtFree(). The function will return NULL if no elements are selected.

Usage

Use this routine to find the rows that currently are selected in an extended list.

See Also

`XmMultiList(2)`.

Name

XmMultiListMakeRowVisible – make row of a MultiList widget visible.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListMakeRowVisible (Widget *widget*, int *row*)

Inputs

widget	Specifies the MultiList widget.
row	Specifies a row to be made visible.

Description

XmMultiListMakeRowVisible() scrolls the MultiList to make the specified row visible.

Usage

The *row* specifies the row number in the MultiList starting from 0.

See Also

`XmMultiList`(2).

Name

XmMultiListSelectAllItems – set the selection state on all rows of a MultiList.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListSelectAllItems (Widget *w*, Boolean *notify*)

Inputs

widget Specifies the MultiList widget.
notify Specifies whether to call XmNsingleSelectionCallback for each item in a list.

Description

XmMultiListSelectAllItems() sets the selection state of the MultiList widget on all rows.

See Also

`XmMultiList`(2).

Name

XmMultiListSelectItems – set the selection state of items of a MultiList.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListSelectItems(Widget *widget*, XmString *item*, int *column*, Boolean *notify*)

Inputs

widget	Specifies the MultiList widget.
item	Specifies XmString to use as a selection key.
column	Specifies a column number to match.
notify	Specifies whether to call XmNsingleSelectionCallback.

Description

XmMultiListSelectItems() sets the selection state of the MultiList widget by matching column entries to *item*.

Usage

The *column* specifies the column number in the MultiList starting from 0 or XmANY_COLUMN.

See Also

`XmMultiList`(2).

Name

XmMultiListSelectRow –set the selection state of a specified row of a MultiList widget.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListSelectRow (Widget *widget*, int *row*, Boolean *notify*)

Inputs

widget	Specifies the MultiList widget.
child	Specifies a row to select.
notify	Specifies whether to call XmNsingleSelectionCallback for the row.

Description

XmMultiListSelectRow() toggles the selection state of the row of the MultiList widget.

Usage

The *row* specifies the row number in the MultiList starting from 0.

See Also

`XmMultiList`(2).

Name

XmMultiListToggleRow–toggle the selection state of a specified row of a Multi-List widget.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListToggleRow (Widget *widget*, short *row*)

Inputs

widget	Specifies the MultiList widget.
child	Specifies a row of the MultiList widget.

Description

XmMultiListToggleRow() returns the selection state of the MultiList widget.

Usage

The *row* specifies the row number in the MultiList starting from 0.

See Also

`XmMultiList`(2).

Name

XmMultiListUnselectAllItems — An Extended List function that deselects all rows of the list

Synopsis

#include <Xm/MultiList.h>

void XmMultiListUnselectAllItems(Widget *widget*);

Inputs

widget Specifies the ID of the Extended list widget

Availability

Motif 2.2 and later.

Description

XmMultiListUnselectAllItems() unhightlights all of the selected items in the specified *widget.*

Usage

XmMultiListUnselectAllItems() is a convience routine that allows you to deselect all items in a list.

See Also

`XmMultiList(2)`.

Name

XmMultiListUnselectItem — An Extended List function that deselects the specified item of the list.

Synopsis

#include <Xm/MultiList.h>

void XmMultiListUnselectItem(Widget *widget*, Xm18RowInfo **row_info*);

Inputs

widget	Specifies the ID of the Extended list widget
row_info	Specifies the pointer to the row which is to be unselected.

Availability

Motif 2.2 and later.

Description

XmMultiListUnselectItem unselects the row designated by row_info of the passed extended list widget.

Usage

XmMultiListUnselectItem() is a convience routine that allows you to deselect an item in a list.

See Also

`XmMultiList(2)`

Name

XmNotebookGetPageInfo – return information about a Notebook page.

Synopsis

#include <Xm/Notebook.h>

XmNotebookPageStatus XmNotebookGetPageInfo (Widget *widget*,
int *page_number*,
XmNotebookPageInfo **page_info*)

Inputs

widget Specifies the Notebook widget.
page_number Specifies a logical page number.

Outputs

page_info Returns a structure into which the requested page information is placed.

Returns

The status of the search for the requested information.

Availability

Motif 2.0 and later.

Description

`XmNotebookGetPageInfo`() returns information associated with a logical page of the Notebook.

The Notebook searches through the list of its children, looking for those which are associated with the logical page number specified by *page_number*. The Notebook principally searches for page children, but collects data in passing on any status area child with a matching logical number, or major and minor tab children whose logical page number does not exceed *page_number*. The function returns within the *page_info* structure the data collected for each of the child widget types.

If the requested *page_number* is greater than the value of the Notebook XmNlast-PageNumber resource, or less than the Notebook XmNfirstPageNumber value, the function returns XmPAGE_INVALID.

Otherwise, if exactly one matching page child is found, the function returns XmPAGE_FOUND. If more than one matching page child is found, the routine returns XmPAGE_DUPLICATED. For no matching page child, the return value is XmPAGE_EMPTY.

Usage

XmNotebookGetPageInfo performs a linear search through the children of the Notebook for widgets whose XmNpageNumber constraint resource matches the requested *page_number*. If a matching child is found with the XmNnotebook-ChildType resource set to XmPAGE, the widget ID is stored within the *page_widget* element of the *page_info* structure. If a matching child is of type XmSTATUS_AREA, the widget ID is placed in the *status_area_widget* element. If during the search a child widget is found which is of type XmMAJOR_TAB, and the logical page number of the child does not exceed *page_number*, the widget ID is stored within the *major_tab_widget* element. Again, if a child widget is found of type XmMINOR_TAB, and the logical page number of the child does not exceed *page_number*, the widget ID is stored within the *minor_tab_widget* element of page_info.

The *page_widget*, *status_area_widget*, *major_tab_widget*, and *minor_tab_widget* elements of the *page_info* structure are set during the search as each Notebook child is compared, even if no XmPAGE child is found, or if *page_number* exceeds the Notebook first and last page resources. An element of the *page_info* structure can be NULL if no child of the associated type is found with a logical page number which meets the matching criteria.

The Notebook automatically sorts children into ascending logical page order, and the search is terminated as soon as any child has a logical page number which exceeds the requested *page_number*.

Structures

XmNotebookPageInfo is defined as follows:

```
typedef struct {
    int         page_number;          /* the requested page number      */
    Widget      page_widget;          /* any matching page widget       */
    Widget      status_area_widget;   /* any matching status area widget */
    Widget      major_tab_widget;     /* the nearest major tab widget   */
    Widget      minor_tab_widget;     /* the nearest minor tab widget   */
} XmNotebookPageInfo;
```

A XmNotebookPageStatus can have one of the following values:

XmPAGE_FOUND	XmPAGE_INVALID
XmPAGE_EMPTY	XmPAGE_DUPLICATED

See Also

`XmNotebook`(2).

Name

XmObjectAtPoint – determine the child nearest to a point.

Synopsis

#include <Xm/Xm.h>

Widget XmObjectAtPoint (Widget *widget*, Position *x*, Position *y*)

Inputs

widget	Specifies a composite widget.
x	Specifies an X coordinate relative to the widget left side.
y	Specifies an Y coordinate relative to the widget top side.

Returns

The widget most closely associated with the coordinate x, y.

Availability

Motif 2.0 or later.

Description

`XmObjectAtPoint`() searches the list of children of *widget*, and returns the widget ID of the child associated with the *x*, *y* coordinate. *x* and *y* are interpreted as pixel values, relative to the top left of the Manager widget.

Usage

`XmObjectAtPoint`() calls the object_at_point method associated with a Manager widget, in order to determine the child of the Manager most closely associated with the coordinate specified by *x* and *y*. Each widget class may override the object_at_point method inherited from Manager, to redefine what is meant by "associated".

The default Manager class method returns the last managed gadget which contains the coordinate.

The DrawingArea overrides the default method, and performs a simple linear search for the first managed child, widget or gadget, which contains the coordinate.

The Container overrides the object_at_point method, by searching through the list of logical child nodes, using any XmQTpointIn trait held by each child to determine a logical match with the coordinate. If no XmQTpointIn is held by the child, the Container simply checks whether the coordinate is within the child dimensions. The IconGadget holds the XmQTpointIn trait, although neither this fact nor the trait itself is otherwise documented.

See Also

`XmContainer`(2), `XmDrawingArea`(2), `XmGadget`(2),

XmIconGadget(2), XmManager(2).

Name

XmOptionButtonGadget – get the CascadeButtonGadget in an option menu

Synopsis

#include <Xm/RowColumn.h>

Widget XmOptionButtonGadget (Widget *option_menu*)

Inputs

option_menu Specifies the option menu.

Returns

The widget ID of the internal CascadeButtonGadget.

Description

`XmOptionButtonGadget`() returns the widget ID for the internal CascadeButtonGadget that is created when the specified *option_menu* widget is created. An option menu is a RowColumn widget containing two gadgets: a CascadeButtonGadget that displays the current selection and posts the submenu and a LabelGadget that displays the XmNlabelString resource.

Usage

`XmOptionButtonGadget`() provides a way for an application to access the internal CascadeButtonGadget that is part of an option menu. Once you have retrieved the gadget, you can alter its appearance. In Motif 1.2, you can also specify resources for the gadget using the widget name OptionButton.

See Also

`XmOptionLabelGadget`(1), `XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmOptionMenu`(2), `XmRowColumn`(2).

Name

XmOptionLabelGadget – get the LabelGadget in an option menu.

Synopsis

#include <Xm/RowColumn.h>

Widget XmOptionLabelGadget (Widget *option_menu*)

Inputs

option_menu Specifies the option menu.

Description

`XmOptionLabelGadget`() returns the widget ID for the internal LabelGadget that is created when the specified *option_menu* widget is created. An option menu is a RowColumn widget containing two gadgets: a LabelGadget that displays the XmNlabelString resource, and a CascadeButtonGadget that displays the current selection and posts the submenu.

Usage

`XmOptionLabelGadget`() provides a way for an application to access the internal LabelGadget that is part of an option menu. Once you have retrieved the gadget, you can alter its appearance. In Motif 1.2, you can also specify resources for the gadget using the widget name OptionLabel.

See Also

`XmOptionButtonGadget`(1), `XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmOptionMenu`(2), `XmRowColumn`(2).

Name

XmPanedGetPanes – A Paned function that retrieves the panes in the widget.

Synopsis

#include <Xm/Paned.h>

void XmPanedGetPanes (Widget *widget*, WidgetList **panes_returned*, int **num_panes_returned*)

Inputs

widget	Specifies the widget ID of the Paned window.
panes_returned	Specifies the list of panes in the Paned window widget.
num_panes_returned	Specifies the number of panes in the Paned window widget.

Description

`XmPanedGetPanes`() retrieves the panes in the widget. Because the Paned widget adds children other than the panes, these values are not the same as those retrieved with the XmNchildren and XmNnumChildren resources.

Usage

`XmPanedGetPanes`() is a convenience routine that retrieves the panes in the widget.

See Also

`XmPaned`(2).

Name

XmParseMappingCreate – create a parse mapping.

Synopsis

XmParseMapping XmParseMappingCreate (Arg **arg_list*, Cardinal *arg_count*)

Inputs

arg_list Specifies an argument list, consisting of resource name/value pairs.
arg_count Specifies the number of arguments in arg_list.

Returns

An allocated parse mapping.

Availability

Motif 2.0 and later.

Description

`XmParseMappingCreate`() creates a parse mapping, which is an entry in a parse table. A parse mapping consists minimally of a match pattern, and a substitution pattern or procedure, which can be used by string parsing functions in order to compare against and subsequently transform text. A parse mapping is created through a resource style argument list, where *arg_list* is an array of resource name/value pairs, and *arg_count* is the number of such pairs.

Usage

A parse table is an array of parse mappings. `XmParseMappingCreate`() creates a parse mapping using a resource style parameter list. The parse table can subsequently be passed to `XmStringParseText`() in order to filter or modify an input string.

`XmParseMappingCreate`() allocates storage associated with the returned parse mapping object. It is the responsibility of the programmer to free the allocated memory by a call to `XmParseMappingFree`() at the appropriate moment.

Example

The following code fragment creates a parse mapping which performs a simple swap of occurrences of two characters within an input string:

```
char *swapover ( char   *input,     /* input string */
                 char   *a,         /* only first character in array used */
                 char   *b)         /* only first character in array used */
{
    XmString            tmp;
    XmParseMapping      parse_mapping;
```

```
XmParseTable        parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
Cardinal            parse_table_index = 0;
Arg                 argv[4];
Cardinal            argc   = 0;
char                *output = (char *) 0;

/* create a XmParseMapping object to swap *a with *b */

argc = 0;
tmp = XmStringCreateLocalized (a);
XtSetArg (argv[argc], XmNincludeStatus,     XmINSERT);
argc++;
XtSetArg (argv[argc], XmNsubstitute,        tmp);
argc++;
XtSetArg (argv[argc], XmNpattern,           b);
argc++;
XtSetArg (argv[argc], XmNpatternType,       XmCHARSET_TEXT);
argc++;
parse_mapping = XmParseMappingCreate (argv, argc);
parse_table[parse_table_index++] = parse_mapping;
XmStringFree (tmp);

/* create a XmParseMapping object to swap *b with *a */
argc = 0;
tmp = XmStringCreateLocalized (b);
XtSetArg (argv[argc], XmNincludeStatus,     XmINSERT);
argc++;
XtSetArg (argv[argc], XmNsubstitute,        tmp);
argc++;
XtSetArg (argv[argc], XmNpattern,           a);
argc++;
XtSetArg (argv[argc], XmNpatternType,       XmCHARSET_TEXT);
argc++;
parse_mapping = XmParseMappingCreate (argv, argc);
parse_table[parse_table_index++] = parse_mapping;
XmStringFree (tmp);

/* substitute using the XmParseMapping. */

tmp = XmStringParseText ((XtPointer) input, NULL, NULL,
                         XmCHARSET_TEXT,
                         parse_table, parse_table_index, NULL);
XmParseTableFree (parse_table, parse_table_index);
```

```
    /* convert XmString to String */
    if (tmp != (XmString) 0) {
        output = (char *) XmStringUnparse (tmp, NULL,
                                           XmCHARSET_TEXT,
                                           XmCHARSET_TEXT, NULL,
                                           0, XmOUTPUT_ALL);[1]
        XmStringFree (tmp);
    }

    return output;
}
```

See Also

`XmParseMappingFree(1), XmParseMappingGetValues(1), XmParseMappingSetValues(1), XmParseTableFree(1), XmStringParseText(1), XmStringUnparse(1), XmParseMapping(2).`

1.The code sample in the 2nd edition used `XmStringGetLtoR`() to convert the compound string. `XmStringGetLtoR`() is deprecated as of Motif 2.0.

Name

XmParseMappingFree – free the memory used by a parse mapping.

Synopsis

void XmParseMappingFree (XmParseMapping *parse_mapping*)

Inputs

parse_mapping Specifies a parse mapping.

Availability

Motif 2.0 and later.

Description

`XmParseMappingFree`() deallocates storage used by the specified parse mapping object.

Usage

The XmParseMapping type is opaque, and represents an entry in a parse table, which can be used for transforming text. A parse mapping is created by `XmParseMappingCreate`(), which allocates storage for the object represented by the type, and it is the responsibility of the programmer to reclaim the memory when the parse mapping is no longer required.

It is important to call `XmParseMappingFree`() rather than `XtFree`() upon redundant parse mappings, otherwise compound strings internally referenced by the object are not deallocated.

See Also

`XmParseMappingCreate`(1), `XmParseMappingGetValues`(1), `XmParseMappingSetValues`(1), `XmParseTableFree`(1), `XmStringParseText`(1), `XmParseMapping`(2).

Name

XmParseMappingGetValues – fetch resources from a parse mapping object.

Synopsis

```
void XmParseMappingGetValues ( XmParseMapping   parse_mapping,
                               Arg              *arg_list,
                               Cardinal         arg_count)
```

Inputs

parse_mapping Specifies a parse mapping object.

arg_count Specifies the number of arguments in the list arg_list.

Outputs

arg_list Specifies the argument list of name/value pairs that contain the resource names and addresses into which the resource values are to be stored.

Availability

Motif 2.0 and later.

Description

`XmParseMappingGetValues`() fetches selected attributes from *parse_mapping*. The set of attributes retrieved is specified through the resource list *arg_list*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *arg_count*.

Usage

If the XmNsubstitute attribute of the parse mapping is retrieved, the procedure returns a copy of the internal value. It is the responsibility of the programmer to recover the allocated space at a suitable point by calling `XmStringFree`().

Example

The following code illustrates fetching the values from an XmParseMapping:

```
XtPointer       pattern;
XmTextType      pattern_type;
XmString        substitute;
XmParseProc     parse_proc;
XtPointer       client_data;
XmIncludeStatus include_status;
Arg             argv[6];
Cardinal        argc = 0;

/* construct a resource-style argument list for all XmParseMapping values */
XtSetArg (argv[argc], XmNpattern,     &pattern);      argc++;
XtSetArg (argv[argc], XmNpatternType, &pattern_type); argc++;
```

```
XtSetArg (argv[argc], XmNsubstitute,          &substitute);      argc++;
XtSetArg (argv[argc], XmNinvokeParseProc,     &parse_proc);      argc++;
XtSetArg (argv[argc], XmNclientData,          &client_data);     argc++;
XtSetArg (argv[argc], XmNincludeStatus,       &include_status);  argc++;

/* fetch the values. parse_mapping here is an unspecified XmParseMapping */
XmParseMappingGetValues (parse_mapping, argv, argc);
...
/* XmParseMappingGetValues returns a copy of the XmNsubstitute value */
/* which must be freed when no longer required by the application    */
XmStringFree (substitute);
```

See Also

`XmParseMappingCreate`(1), `XmParseMappingFree`(1), `XmParseMappingSetValues`(1), `XmParseTableFree`(1), `XmParseMapping`(2).

Name

XmParseMappingSetValues – sets resources for a parse mapping object.

Synopsis

```
void XmParseMappingSetValues (  XmParseMapping    parse_mapping,
                                Arg               *arg_list,
                                Cardinal          arg_count)
```

Inputs

parse_mapping	Specifies a parse mapping object.
arg_list	Specifies the list of name/value pairs containing resources to be modified.
arg_count	Specifies the number of arguments in the list *arg_list*.

Availability

Motif 2.0 and later.

Description

`XmParseMappingSetValues`() sets selected attributes within *parse_mapping*. The set of attributes which is modified is specified through the resource list *arg_list*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *arg_count*.

Usage

If the XmNsubstitute attribute of the parse mapping is set, the procedure internally takes a copy of the supplied value. It is the responsibility of the programmer to recover the allocated space at a suitable point by calling `XmStringFree`().

Example

The following skeleton code illustrates changing the values of a parse mapping:

```
XmIncludeStatus map_tab (  XtPointer          *in_out,
                           XtPointer          text_end,       /* unused
*/
                           XmTextType         type,           /* unused
*/
                           XmStringTag        tag,            /* unused
*/
                           XmParseMapping     entry,          /* unused
*/
                           int                pattern_length, /* unused
*/
                           XmString           *str_out,
```

```
                                XtPointer       call_data)      /* unused
*/
{
    /* Insert an XmString Tab component into the output stream */
    *str_out = XmStringComponentCreate
    (XmSTRING_COMPONENT_TAB, 0, NULL);
    *in_out = (*in_out + 1);

    return XmINSERT;

}

/* change a parse mapping to invoke the above parse procedure */
void set_parse_tab_mapping (XmParseMapping parse_mapping)
{
    Arg         argv[4];
    Cardinal    argc = 0;

    /* construct resource-style argument list for XmParseMapping values */
    XtSetArg (argv[argc], XmNpattern,           "\t");
    argc++;
    XtSetArg (argv[argc], XmNpatternType,       XmCHARSET_TEXT);
    argc++;
    XtSetArg (argv[argc], XmNincludeStatus,     XmINVOKE);
    argc++;
    XtSetArg (argv[argc], XmNinvokeParseProc,   map_tab);
    argc++;

    /* change the values */
    XmParseMappingSetValues (parse_mapping, argv, argc);
}
```

See Also

`XmParseMappingCreate`(1), `XmParseMappingFree`(1), `XmParseMappingGetValues`(1), `XmParseTableFree`(1), `XmParseMapping`(2),

Name

XmParseTableFree – free the memory used by a parse table.

Synopsis

void XmParseTableFree (XmParseTable *parse_table*, Cardinal *parse_count*)

Inputs

parse_table	Specifies a parse table.
parse_count	Specifies the number of entries in the parse table.

Availability

Motif 2.0 and later.

Description

`XmParseTableFree`() deallocates storage used by the specified *parse_table*. In addition, the function deallocates storage used by any parse mapping elements of the table. *parse_count* indicates the number of mapping elements within the table.

Usage

A parse table is an array of XmParseMapping objects. The XmParseMapping is an opaque type, which is used when transforming text. Each parse mapping object allocates memory in addition to any memory allocated by the parse table array. It is important to call `XmParseTableFree`() rather than `XtFree`() when deallocating storage associated with a parse table, otherwise objects constituent within the array, and compound strings internally referenced by the parse mapping objects, are not deallocated. The function should be called when a parse table is no longer needed.

Example

```
/* Allocate a parse table */
XmParseTable        parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
Cardinal            parse_table_index = 0;
XmParseMapping      parse_mapping;
Arg                 argv[MAX_ARGS];
Cardinal            argc = 0;

/* Create a XmParseMapping object */
argc = 0;
...
parse_mapping = XmParseMappingCreate (argv, argc);

/* Insert into parse table */
parse_table[parse_table_index++] = parse_mapping;
```

```
/* Create another XmParseMapping object */
argc = 0;
...
parse_mapping = XmParseMappingCreate (argv, argc);

/* Insert into parse table */
parse_table[parse_table_index++] = parse_mapping;

/* Use the XmParseTable. */
tmp = XmStringParseText ((XtPointer) input, NULL, NULL,
                         XmCHARSET_TEXT, parse_table,
                         parse_table_index, NULL);

/* Free the parse table: this also frees the parse mappings */
XmParseTableFree (parse_table, parse_table_index);
```

See Also

`XmParseMappingCreate`(1), `XmParseMappingFree`(1), `XmParseMappingGetValues`(1), `XmParseMappingSetValues`(1), `XmParseMapping`(2).

Name

XmPrintPopupPDM – notify the Print Display Manager.

Synopsis

#include <Xm/Print.h>

XtEnum XmPrintPopupPDM (Widget *print_shell*, Widget *video_shell*)

Inputs

print_shell	Specifies a PrintShell widget.
video_shell	Specifies the widget on whose behalf the PDM dialog is required.

Returns

Returns XmPDM_NOTIFY_SUCCESS if the PDM was notified, XmPDM_NOTIFY_FAIL otherwise.

Availability

Motif 2.1 and later.

Note that not all operating system vendors incorporate the XmPrintShell within the native Motif toolkit.[1]

Description

`XmPrintPopupPDM`() sends a notification to start a Print Display Manager for the application. The notification is issued to either the display associated with *print_shell*, or the display of *video_shell*, depending upon the value of the environment variable XPDMDISPLAY. XPDMDISPLAY can only be set to "print" or "video". If the value is "print", the notification is sent to the display of *print_shell*, and similarly the value "video" sends the notification to the display of *video_shell*. If the notification could be sent, the function returns XmPDM_NOTIFY_SUCCESS, otherwise the return value is XmPDM_NOTIFY_FAIL.

Usage

`XmPrintPopupPDM`() is a convenience function which issues a notification through the X selection mechanisms in order to start a Print Dialog Manager. The notification is issued asynchronously: the return value XmPDM_NOTIFY_SUCCESS indicates that the message has successfully been issued, not that any PDM is now initialized. In order to track the status of the PDM, the programmer registers an XmNpdmNotificationCallback with the widget *print_shell*, which must be an instance of the PrintShell widget class. To ensure that the contents of the video_shell is not modified whilst the PDM is ini-

1.Sun Solaris being a case in point.

tializing, XmPrintPopupPDM() creates an input-only window over the top of *video_shell*, and the window is only removed when the PDM indicates that it is present, or if the selection XmIPDM_START times out. The timeout period is set at two minutes.

See Also

XmPrintSetup(1), XmPrintToFile(1), XmRedisplayWidget(1), XmPrintShell(2).

Name

XmPrintSetup – create a Print Shell widget.

Synopsis

```
#include <Xm/Print.h>

Widget XmPrintSetup ( Widget     video_widget,
                      Screen     *print_screen,
                      String     name
                      ArgList    arg_list,
                      Cardinal   arg_count)
```

Inputs

video_widget	Specifies a widget from which video application data is fetched.
print_screen	Specifies the screen on which the PrintShell is created.
name	Specifies the name of the created PrintShell.
arg_list	Specifies an argument list of name/value pairs that contain resources for the PrintShell.
arg_count	Specifies the number of arguments in the list arg_list.

Returns

The created PrintShell, or NULL if no ApplicationShell can be found from *video_widget*,

Availability

Motif 2.1 and later.

Note that not all operating system vendors incorporate the PrintShell in their native toolkit.[1]

Description

`XmPrintSetup`() creates a PrintShell widget with the given *name* on the screen *print_screen*. The new PrintShell is returned to the application. Resources which configure the new print shell are supplied through an array of structures which contain name/value pairs. The array of resources is *arg_list*, and the number of items in the array is *arg_count*.

Usage

`XmPrintSetup`() creates a new ApplicationShell on the screen specified by *print_screen*, and thereafter creates a PrintShell as a popup child. The new ApplicationShell is created with the same name and class as the ApplicationShell from which *video_widget* is descended. The XmNmappedWhenManaged resource of

1.For example, Sun Solaris includes the headers, but does not compile the widget into the Motif library.

the PrintShell is set to False under the assumption that subsequent notification of the start of a job or page is the correct time to map the widget. The print shell is finally realized, and returned.

See Also

`XmPrintPopupPDM`(1), `XmPrintToFile`(1), `XmRedisplayWidget`(1), `XmPrintShell`(2).

Name

XmPrintToFile – save X Print Server data to file.

Synopsis

```
#include <Xm/Print.h>

XtEnum XmPrintToFile (  Display         *display,
                        String          file_name,
                        XPFinishProc    finish_proc,
                        XPointer        client_data)
```

Inputs

display	Specifies the print connection to the X server.
file_name	Specifies the name of the file to contain the print output.
finish_proc	Specifies a procedure called when printing is finished.
client_data	Specifies application data to be passed to finish_proc.

Returns

True if printing can be initiated, otherwise False.

Availability

Motif 2.1 and later.

Note that not all operating system vendors incorporate the XmPrintShell in their native toolkits.[1]

Description

`XmPrintToFile`() is a convenience function which provides a simple interface onto the X Print mechanisms, in order to save print data to the file *file_name*. Printing takes place asynchronously, and the programmer receives notification of the status of the printing task by supplying *finish_proc*, which is called when the task is finished. The *display* parameter is the print connection to the X server, and is used to deduce an application name and class.

Usage

If `XmPrintToFile`() cannot open the file *file_name* for writing, create a pipe, or fork off a child process, the procedure returns False. An application name and class is deduced using the *display* parameter, and these are used by the child process, which creates a new application context, and opens a new display connection using the same name and class as the application process. Data is retrieved from the X server through a call to `XpGetDocumentData`(). The parent process does not wait for the child to complete, but returns immediately

1.For example, Sun Solaris supply the widget headers, but do not compile the component into the Motif library.

after initiating the child process. The return value True therefore does not mean that the print task is complete, merely that the task is initiated.

The application is notified of task completion by supplying an XPFinishProc. The *status* parameter passed to the finish procedure when the task is completed is set to XPGetDocFinished on successful completion. If for any reason the child process fails to print the data, the file *file_name* is both closed and removed. The file is closed in any case prior to calling the XPFinishProc.

`XpStartJob`() must be called by the application before `XmPrintToFile`() can be called.

Structures

An XPFinishProc is specified as follows:

```
typedef void (*XPFinishProc)( Display          *display,
                              XPContext        context,
                              XPGetDocStatus   status,
                              XPointer         client_data);
```

If status is XPGetDocFinished, the print task has completed successfully.

See Also

`XmPrintPopupPDM`(1), `XmPrintSetup`(1), `XmRedisplayWidget`(1), `XmPrintShell`(2).

Name

XmProcessTraversal – set the widget that has the keyboard focus.

Synopsis

Boolean XmProcessTraversal (Widget *widget*, XmTraversalDirection *direction*)

Inputs

widget	Specifies the widget whose hierarchy is to be traversed.
direction	Specifies the direction in which to traverse the hierarchy. Pass one of the values from the list below.

Returns

True on success or False otherwise.

Description

`XmProcessTraversal`() causes the input focus to change to another widget under application control, rather than as a result of keyboard traversal events from a user. *widget* specifies the widget whose hierarchy is traversed up to the shell widget. If that shell has the keyboard focus, `XmProcessTraversal`() changes the keyboard focus immediately. If that shell does not have the focus, the routine does not have an effect until the shell receives the focus.

The *direction* argument specifies the nature of the traversal to be made. In each case, the routine locates the hierarchy that contains the specified widget and then performs the action that is particular to the *direction*. If the new setting succeeds, `XmProcessTraversal`() returns True. The routine returns False if the keyboard focus policy is not XmEXPLICIT, if no traversable items exist, or if the arguments are invalid.

Usage

For XmTRAVERSE_CURRENT, if the tab group that contains *widget* is inactive, it is made the active tab group. If *widget* is in the active tab group, it is given the keyboard focus; if *widget* is the active tab group, the first traversable item in it is given the keyboard focus. For XmTRAVERSE_UP, XmTRAVERSE_DOWN, XmTRAVERSE_LEFT, and XmTRAVERSE_RIGHT, in the hierarchy that contains *widget*, the item in the specified *direction* from the active item is given the keyboard focus. For XmTRAVERSE_NEXT and XmTRAVERSE_PREV, in the hierarchy that contains *widget*, the next and previous items in child order from the active item are given keyboard focus. For XmTRAVERSE_HOME, in the hierarchy that contains widget, the first traversable item is given the keyboard focus. For XmTRAVERSE_NEXT_TAB_GROUP and XmTRAVERSE_PREV_TAB_GROUP, in the hierarchy that contains *widget*,

the next and previous tab groups from the active tab group are given the keyboard focus.

In Motif 2.0 and later, new XmTraversalDirection values XmTRAVERSE_GLOBALLY_FORWARD and XmTRAVERSE_GLOBALLY_BACKWARD are provided in order to implement the XmDisplay resource XmNenableButtonTab. If enabled, for XmTRAVERSE_GLOBALLY_FORWARD navigation proceeds to the next (or downwards, depending upon orientation) item within the current tab group, unless the current location is the last item in the group, when navigation is into the next tab group. Similarly, for XmTRAVERSE_GLOBALLY_BACKWARD navigation proceeds to the previous (or upwards) item in the current tab group, unless the current location is the first item in the group, when navigation is into the previous tab group. The interpretation of the *direction* values XmTRAVERSE_GLOBALLY_FORWARD and XmTRAVERSE_GLOBALLY_BACKWARD is reversed where XmNlayoutDirection is XmRIGHT_TO_LEFT.

`XmProcessTraversal`() does not allow traversal to widgets in different shells or widgets that are not mapped. Calling `XmProcessTraversal`() inside a XmNfocusCallback causes a segmentation fault.

Example

The following code fragments shows the use of `XmProcessTraversal`() as a callback routine for a text widget. When the user presses the Return key, the keyboard focus is advanced to the next input area:

```
Widget form, label, text;

form = XtVaCreateWidget ("form", xmFormWidgetClass, parent,
                         XmNorientation, XmHORIZONTAL,
                         NULL);
label = XtVaCreateManagedWidget ("label", xmLabelGadgetClass, form,
                         XmNleftAttachment,
                         XmATTACH_FORM,
                         XmNtopAttachment,
                         XmATTACH_FORM,
                         XmNbottomAttachment,
                         XmATTACH_FORM,
                         NULL);
text = XtVaCreateManagedWidget ("text", xmTextWidgetClass, form,
                         XmNleftAttachment,
                         XmATTACH_WIDGET,
                         XmNleftWidget,          label,
```

```
                                XmNtopAttachment,
                                XmATTACH_FORM,
                                XmNrightAttachment,
                                XmATTACH_FORM,
                                XmNbottomAttachment,
                                XmATTACH_FORM,
                                NULL);
    XtAddCallback (text, XmNactivateCallback,
                        XmProcessTraversal, (XtPointer)
                        XmTRAVERSE_NEXT_TAB_GROUP);
    XtManageChild (form);
```

Structures

The possible values for direction are:

XmTRAVERSE_CURRENT	XmTRAVERSE_NEXT
XmTRAVERSE_UP	XmTRAVERSE_PREV
XmTRAVERSE_DOWN	XmTRAVERSE_HOME
XmTRAVERSE_LEFT	
XmTRAVERSE_NEXT_TAB_GROUP	
XmTRAVERSE_RIGHT	
XmTRAVERSE_PREV_TAB_GROUP	
XmTRAVERSE_GLOBALLY_FORWARD	
XmTRAVERSE_GLOBALLY_BACKWARD	

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1), `XmGetVisibility`(1), `XmIsTraversable`(1).

Name

XmRedisplayWidget – force widget exposure for printing.

Synopsis

#include <Xm/Print.h>

void XmRedisplayWidget (Widget *widget*)

Inputs

widget Specifies the widget to redisplay.

Availability

Motif 2.1 and later.

Note that not all operating system vendors compile the XmPrintShell into their native Motif toolkits.[1]

Description

`XmRedisplayWidget`() forces widget to redisplay itself by invoking the expose method of the *widget*. The routine is a convenience function which hides the internals of the X11R6 Xp mechanisms, which use *widget* exposure in order to implement printing.

Usage

`XmRedisplayWidget`() constructs a region which corresponds precisely to the location and area occupied by a widget. The expose method of the widget is called directly using the region in order to redisplay the widget. `XmRedisplayWidget`() is synchronous in effect. Asynchronous printing is performed by creating a PrintShell, and specifying XmNstartJobCallback, XmNendJobCallback, and XmNpageSetupCallback procedures which are invoked in response to X Print events as they arrive.

`XmRedisplayWidget`() is not multi-thread safe, nor is the *widget* parameter fully validated: it is implicitly assumed to be the descendant of a PrintShell.

1.Sun Solaris supplied the widget headers, but the widget itself is compiled out of the Motif library.

Example

The following code synchronously prints the contents of a text widget:

```
Widget    app_shell, app_text;
Screen    print_screen;
Display   print_display;
Widget    print_shell, print_form, print_text;
short     rows;
int       lines, pages, page;
char      *data;
...
/* create a connection to the X Print server */
print_shell = XmPrintSetup (app_shell, print_screen, "PrintShell", NULL, 0);

/* create a suitable print hierarchy */
print_form = XmCreateForm (print_shell,...);
print_text = XmCreateText (print_form,...);

/* configure and manage the print hierarchy */
...
/* copy the video text to the print text     */
/* what is copied depends upon whether it is */
/* contents and/or visuals that are printed */
...
data = XmTextGetString (app_text);
XmTextSetString (print_text, data);
XtFree (data);
...
/* start a print job */
print_display = XtDisplay (print_shell);
XpStartJob (print_display, XPSpool);

/* deduce number of logical pages in the print text widget */
XtVaGetValues (print_text, XmNrows, &rows, XmNtotalLines, &lines, 0);

for (page = 0, pages = lines / rows; page < pages; page++) {
    /* start of page notification */
    XpStartPage (print_display, XtWindow (print_shell), False);

    /* force the print text to expose itself */
    XmRedisplayWidget (print_text);

    /* end of page notification */
    XpEndPage (print_display);

    /* scroll to next page */
```

```
    XmTextScroll (print_text, rows);
}

/* end of print job notification */
XpEndJob (print_display);
...
```

See Also

`XmPrintPopupPDM`(1), `XmPrintSetup`(1), `XmPrintToFile`(1), `XmPrintShell`(2).

Name

XmRegisterSegmentEncoding – register a compound text encoding format for a font list element tag.

Synopsis

char *XmRegisterSegmentEncoding (char **fontlist_tag*, char **ct_encoding*)

Inputs

fontlist_tag	Specifies the compound string font list element tag.
ct_encoding	Specifies the compound text character set.

Returns

The old compound text encoding format for a previously-registered font list element tag or NULL for a new font list element tag.

Availability

Motif 1.2 and later.

Description

`XmRegisterSegmentEncoding`() registers the specified compound text encoding format *ct_encoding* for the specified *fontlist_tag*. Both *fontlist_tag* and *ct_encoding* must be NULL-terminated ISO8859-1 strings. If the font list tag is already associated with a compound text encoding format, registering the font list tag again overwrites the previous entry and the routine returns the previous compound text format. If the font list tag is has not been registered before, the routine returns NULL. If *ct_encoding* is NULL, the font list tag is unregistered. If *ct_encoding* is the reserved value XmFONTLIST_DEFAULT_TAG, the font list tag is mapped to the code set of the current locale. `XmRegisterSegmentEncoding`() allocates storage if the routine returns a character string; the application is responsible for freeing the storage using `XtFree`().

Usage

Compound text is an encoding that is designed to represent text from any locale. Compound text strings identify their encoding using embedded escape sequences. The compound text representation was standardized for X11R4 for use as a text interchange format for interclient communication.

`XmCvtXmStringToCT`() converts a compound string into compound text. The routine uses the font list tag of each compound string segment to select a compound text format for the segment. A mapping between font list tags and compound text encoding formats is stored in a registry. `XmRegisterSegmentEncoding`() provides a way for an application to map particular font list element tags to compound text encoding formats.

See Also

XmCvtXmStringToCT(1), XmMapSegmentEncoding(1).

Name

XmRemoveFromPostFromList – make a menu inaccessible from a widget.

Synopsis

#include <Xm/RowColumn.h>

void XmRemoveFromPostFromList (Widget *menu*, Widget *widget*)

Inputs

menu	Specifies a menu widget
widget	Specifies the widget which no longer posts menu.

Availability

In Motif 2.0 and later, the functional prototype is removed from RowColumn.h, although there is otherwise no indication that the procedure is obsolete. [1]

Description

`XmRemoveFromPostFromList`() is the inverse of the procedure `XmAddToPostFromWidget`(). The menu hierarchy associated with *menu* is made inaccessible from *widget*.

Usage

If the type of menu is XmMENU_PULLDOWN, the XmNsubMenuId resource of widget is set to NULL. If the type of menu is XmMENU_POPUP, event handlers presumably added to widget by XmAddToPostFromWidget() in order to post the menu are removed.

No check is made to ensure that the XmNsubMenuId resource of widget is originally set to menu before clearing the value. Passing the wrong menu into the procedure can therefore have unwanted effects. There are implicit assumptions that widget is a CascadeButton or CascadeButtonGadget when menu is XmMENU_PULLDOWN, and that widget is not a Gadget when menu is XmMENU_POPUP. These are not checked by the procedure.

See Also

`XmAddToPostFromList`(1), `XmGetPostedFromWidget`(1), `XmPopupMenu`(2), `XmPulldownMenu`(2), `XmRowColumn`(2).

1.This is true of Motif 2.1.10, although the header reference is restored in the OpenMotif 2.1.30.

Name

XmRemoveProtocolCallback – remove client callback from a protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmRemoveProtocolCallback (  Widget          shell,
                                 Atom            property,
                                 Atom            protocol,
                                 XtCallbackProc  callback,
                                 XtPointer       closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocol	Specifies the protocol atom.
callback	Specifies the procedure that is to be removed.
closure	Specifies any client data that is passed to the callback.

Description

`XmRemoveProtocolCallback`() removes the specified *callback* from the list of callback procedures that are invoked when the client message corresponding to *protocol* is received.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. To communicate using a protocol, a client sends a ClientMessage event containing a *property* and *protocol*, and the receiving client responds by calling the associated protocol *callback* routine. `XmRemoveProtocolCallback`() allows you to unregister one of these callback routines. The inverse routine is `XmAddProtocolCallback`().

See Also

`XmAddProtocolCallback`(1), `XmInternAtom`(1), `XmRemoveWMProtocolCallback`(1), `VendorShell`(2).

Name

XmRemoveProtocols – remove protocols from the protocol manager.

Synopsis

#include <Xm/Protocols.h>

void XmRemoveProtocols (Widget *shell*, Atom *property*, Atom **protocols*, Cardinal *num_protocols*)

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocols	Specifies a list of protocol atoms.
num_protocols	Specifies the number of atoms in protocols.

Description

`XmRemoveProtocols`() removes the specified *protocols* from the protocol manager and deallocates the internal tables for the protocols. If the specified *shell* is realized and at least one of the *protocols* is active, the routine also updates the handlers and the *property*. The inverse routine is `XmAddProtocols`().

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. `XmRemoveProtocols`() allows you eliminate protocols that can be understood by your application. The inverse routine is XmAddProtocols().

See Also

`XmAddProtocols`(1), `XmInternAtom`(1), `XmRemoveWMProtocols`(1), `VendorShell`(2).

Name

XmRemoveTabGroup – remove a widget from a list of tab groups.

Synopsis

void XmRemoveTabGroup (Widget *tab_group*)

Inputs

tab_group Specifies the widget to be removed.

Availability

In Motif 1.1, `XmRemoveTabGroup`() is obsolete. It has been superseded by setting XmNnavigationType to XmNONE.

Description

`XmRemoveTabGroup`() removes the specified *tab_group* widget from the list of tab groups associated with the widget hierarchy. This routine is retained for compatibility with Motif 1.0 and should not be used in newer applications. If traversal behavior needs to be changed, this should be done by setting the XmNnavigationType resource directly.

Usage

A tab group is a group of widgets that can be traversed using the keyboard rather than the mouse. Users move from widget to widget within a single tab group by pressing the arrow keys. Users move between different tab groups by pressing the Tab or Shift-Tab keys. The inverse routine is XmAddTabGroup().

See Also

`XmAddTabGroup`(1), `XmGetTabGroup`(1), `XmManager`(2), `XmPrimitive`(2).

Name

XmRemoveWMProtocolCallback – remove client callbacks from a XA_WM_PROTOCOLS protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmRemoveWMProtocolCallback (  Widget          shell,
                                   Atom            protocol,
                                   XtCallbackProc  callback,
                                   XtPointer       closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
protocol	Specifies the protocol atom.
callback	Specifies the procedure that is to be removed.
closure	Specifies any client data that is passed to the callback.

Description

`XmRemoveWMProtocolCallback`() is a convenience routine that calls `XmRemoveProtocolCallback`() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. To communicate using a protocol, a client sends a ClientMessage event containing a *property* and *protocol*, and the receiving client responds by calling the associated protocol *callback* routine. `XmRemoveWMProtocolCallback`() allows you to unregister one of these *callback* routines with the window manager *protocol* property. The inverse routine is `XmAddWMProtocolCallback`().

See Also

`XmAddProtocolCallback`(1), `XmAddWMProtocolCallback`(1), `XmInternAtom`(1), `XmRemoveProtocolCallback`(1), `VendorShell`(2).

Name

XmRemoveWMProtocols – remove the XA_WM_PROTOCOLS protocols from the protocol manager.

Synopsis

#include <Xm/Protocols.h>

void XmRemoveWMProtocols (Widget *shell*, Atom **protocols*, Cardinal *num_protocols*)

Inputs

shell	Specifies the widget associated with the protocol property.
protocols	Specifies a list of protocol atoms.
num_protocols	Specifies the number of atoms in protocols.

Description

`XmRemoveWMProtocols`() is a convenience routine that calls `XmRemoveProtocols`() with property set to XA_WM_PROTOCOL, the window manager protocol property. The inverse routine is `XmAddWMProtocols`().

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. `XmRemoveWMProtocols`() allows you to remove this protocol so that it is no longer understood by your application. The inverse routine is `XmAddWMProtocols`().

See Also

`XmAddProtocols`(1), `XmAddWMProtocols`(1), `XmInternAtom`(1), `XmRemoveProtocols`(1), `VendorShell`(2).

Name

XmRenderTableAddRenditions – add renditions to a render table.

Synopsis

```
XmRenderTable XmRenderTableAddRenditions (
        XmRenderTable old_table,
        XmRendition *new_renditions,
        Cardinal new_rendition_count,
        XmMergeMode merge_mode)
```

Inputs

old_table	Specifies a render table.
new_renditions	Specifies an array of renditions to merge with the render table.
new_rendition_count	Specifies the number of renditions in the array.
merge_mode	Specifies the action to take if entries have the same tag.

Returns

The newly allocated merged render table.

Availability

Motif 2.0 and later.

Description

A render table is a set of renditions which can be used to specify the way in which XmStrings are drawn. `XmRenderTableAddRenditions`() creates a new render table by merging the list of renditions specified by *new_renditions* into the renditions contained within *old_table*. If a rendition with the same tag is found in both *old_table* and *new_renditions*, *merge_mode* is used to give precedence. The new render table is returned.

If *old_table* is NULL, a new render table is allocated which contains only the renditions of *new_renditions*. If *new_renditions* is NULL or *new_rendition_count* is zero, the *old_table* is returned unmodified. If a rendition within *old_table* has the same tag as one within *new_renditions*, *merge_mode* determines how to resolve the conflict. If *merge_mode* is XmMERGE_REPLACE, the rendition within *old_table* is ignored, and the rendition within *new_renditions* is added to the new table. If the mode is XmMERGE_SKIP, the new table contains the rendition from *old_table*, and that from *new_renditions* is ignored. If the mode is XmMERGE_NEW, the rendition

within *new_renditions* is used, except that where any resources of the rendition are unspecified, the value is copied from the matching rendition from the *old_table*. A resource is unspecified if the value is XmAS_IS or NULL. Lastly, if the mode is XmMERGE_OLD, it is the *old_table* rendition which is added to the new table, and any unspecified resources are taken from the new rendition.

Usage

The reference count for the original table is decremented and deallocated where necessary, and a newly allocated render table containing the merged data is returned. It is the responsibility of the programmer to reclaim the allocated memory for the returned render table by calling `XmRenderTableFree`() at a suitable point.

Example

The following specimen code creates a set of renditions and merges them into an unspecified render table:

```
XmRendition         new_renditions[2];
XmRenderTable       new_table;
Arg                 argv[4];
Cardinal            argc = 0;
Pixel               fg =...;
Pixel               bg =...;

XtSetArg (argv[argc], XmNfontName,      "fixed");
argc++;
XtSetArg (argv[argc], XmNfontType,      XmFONT_IS_FONT);
argc++;
XtSetArg (argv[argc], XmNloadModel,     XmLOAD_DEFERRED);
argc++;
new_renditions[0] = XmRenditionCreate (widget,
XmFONTLIST_DEFAULT_TAG, argv, argc);

argc = 0;
XtSetArg (argv[argc], XmNrenditionBackground, bg); argc++;
XtSetArg (argv[argc], XmNrenditionForeground, fg); argc++;
new_renditions[1] = XmRenditionCreate (widget, "colors", argv, argc);
new_table = XmRenderTableAddRenditions (old_table, new_renditions, 2,
XmMERGE_REPLACE);)
```

See Also

XmRenderTableCopy(1), XmRenderTableFree(1), XmRenderTableGetRendition(1), XmRenderTableGetRenditions(1), XmRenderTableGetTags(1), XmRenderTableRemoveRenditions(1), XmRenditionCreate(1), XmRenditionFree(1), XmRenditionRetrieve(1), XmRenditionUpdate(1), XmRendition(2).

Name

XmRenderTableCopy – copy a render table.

Synopsis

XmRenderTable XmRenderTableCopy (XmRenderTable *old_table*, XmStringTag **tags*, int *tag_count*)

Inputs

old_table	Specifies the table containing the renditions to be copied.
tags	Specifies an array of tags. Renditions with matching tags are copied.
tag_count	Specifies the number of items within the tags array.

Returns

A new render table containing renditions with matching tags, or NULL.

Availability

Motif 2.0 and later.

Description

An XmRenderTable is an array of XmRendition objects, which are used to render compound strings. `XmRenderTableCopy`() creates a newly allocated render table by copying renditions from an existing table, *old_table*. An array of tags can be supplied which acts as a filter: only those renditions from *old_table* which have a matching XmNtag resource are copied. The number of items within any tags array is specified through *tag_count*. If *tags* is NULL, all of the renditions within *old_table* are copied. If *old_table* is NULL, the function returns NULL.

Usage

The function allocates storage for the returned render table, including storage for each of the newly copied renditions. It is the responsibility of the programmer to reclaim the memory at an appropriate point by calling `XmRenderTableFree`().

In Motif 2.0 and later, the XmRenderTable supersedes the XmFontList, which is now considered obsolete. For backwards compatibility, the XmFontList opaque type is implemented through the render table.

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableFree`(1), `XmRenderTableGetRendition`(1), `XmRenderTableGetRenditions`(1), `XmRenderTableGetTags`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRenditionRetrieve`(1),

XmRenditionUpdate(1), XmRendition(2).

Name

XmRenderTableCvtFromProp – convert from a string representation into a render table.

Synopsis

XmRenderTable XmRenderTableCvtFromProp (Widget *widget*, char **property*, unsigned int *length*)

Inputs

widget	Specifies a destination widget in a data transfer.
property	Specifies the render table in string representation format.
length	Specifies the number of bytes in the property string.

Returns

The converted render table.

Availability

Motif 2.0 and later.

Description

`XmRenderTableCvtFromProp`() converts a string representation of a render table into an XmRenderTable. The string representation to be converted is given by *property*, and the size of the string in bytes is *length*.

Usage

Typically, the procedure is used within the destination callback of widget when it is the target of a data transfer. The inverse function `XmRenderTableCvtToProp`() is called by the convert procedures of the source of the data transfer. `XmRenderTableCvtFromProp`() returns allocated memory, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmRenderTableFree`().

See Also

`XmRenderTableCvtToProp`(1), `XmRenderTableFree`(1), `XmRendition`(2).

Name

XmRenderTableCvtToProp – convert a render table into a string representation.

Synopsis

unsigned int XmRenderTableCvtToProp (Widget *widget*,
XmRenderTable *render_table*,
char
***property_return*)

Inputs

widget	Specifies a source widget for the render table.
render_table	Specifies the render table to convert.

Outputs

property_return	Returns the string representation of the converted render table.

Returns

The number of bytes in the converted string representation.

Availability

Motif 2.0 and later.

Description

`XmRenderTableCvtToProp`() converts an XmRenderTable render_table into a string representation at the address specified by *property_return*. The length of the converted string is returned.

Usage

Typically, the procedure is used within the convert callback of widget when it is the source of a data transfer. The procedure returns allocated memory within property_return, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XtFree`().

The standard built-in conversion routines within the Uniform Transfer Model internally call `XmRenderTableCvtToProp`() when asked to convert the _MOTIF_RENDER_TABLE selection.

See Also

`XmRenderTableCvtFromProp`(1), `XmRendition`(2).

Name

XmRenderTableFree – free the memory used by a render table.

Synopsis

void XmRenderTableFree (XmRenderTable *table*)

Inputs

table Specifies the render table to free.

Availability

Motif 2.0 and later.

Description

`XmRenderTableFree`() is a convenience function which deallocates space used by the render table table.

Usage

Render tables, and the renditions which they contain, are reference counted. It is important to call `XmRenderTableFree`() on a render table rather than `XtFree`() so that each rendition in the table is properly deallocated. Motif caches and shares render tables and the renditions which they contain, and so an improper XtFree() would not respect any sharing currently in place. `XmRenderTableFree`() does not actually free the render table until the reference count is zero.

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableCopy`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRendition`(2).

Name

XmRenderTableGetRendition – search a render table for a matching rendition.

Synopsis

XmRendition XmRenderTableGetRendition (XmRenderTable *table*, XmStringTag *tag*)

Inputs

table	Specifies the render table to search.
tag	Specifies the tag with which to find a rendition.

Returns

A Rendition which matches tag, otherwise NULL.

Availability

Motif 2.0 and later.

Description

`XmRenderTableGetRendition`() is a convenience function which searches *table*, and returns the rendition which matches *tag*.

Usage

`XmRenderTableGetRendition`() performs a linear search through the renditions contained within *table*, comparing the XmNtag resource value with the search string given by *tag*. If no match is found, any XmNnoRenditionCallback[1] callbacks registered with the XmDisplay object are invoked, supplying the table as the *render_table* element of the XmDisplayCallbackStruct passed to the callbacks. If the callbacks modify the *render_table* element, the linear search is restarted. A copy of any matching rendition is returned, otherwise NULL.

`XmRenderTableGetRendition`() allocates space for the returned rendition, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmRenditionFree`().

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableGetRenditions`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionFree`(1), `XmRendition`(2).

1.Erroneously given as XmNnoRendition in 2nd edition.

Name

XmRenderTableGetRenditions – search a render table for matching renditions.

Synopsis

```
XmRendition *XmRenderTableGetRenditions ( XmRenderTable   table,
                                          XmStringTag     *tags,
                                          Cardinal        tag_count)
```

Inputs

table	Specifies the render table to search.
tags	Specifies an array of tags for which matching renditions are required.
tag_count	Specifies the number of items in tags.

Returns

The array of renditions which have matching tags.

Availability

Motif 2.0 and later.

Description

`XmRenderTableGetRenditions`() searches *table* for all renditions which have a tag that matches an entry within the list *tags*. If the *table* is NULL, or if *tags* is NULL, or if *tag_count* is zero, the function returns NULL. Otherwise, the function returns an allocated array of matching rendition objects.

Usage

`XmRenderTableGetRenditions`() iterates through a set of *tags*, comparing in turn each tag with the group of renditions contained within a render table.

If no match is found when comparing a tag, any XmNnoRenditionCallback[1] callbacks registered with the XmDisplay object are invoked, supplying the table as the *render_table* element of the XmDisplayCallbackStruct passed to the callbacks. If the callbacks modify the *render_table* element, the linear search is restarted for that tag.

The documentation states that the function returns an allocated array, renditions being copied into the array at the same index of the matching tag within the tags array. For example, if the third tag in *tags* matches a rendition, that rendition is copied into the third element of the returned array. If any tag in the *tags* list does not match any rendition in the table, that slot in the returned array is set to NULL.

The sources, however, do not match the documentation: renditions are copied into the array in the order which they are matched, ignoring any slots which do

1.Erroneously given as XmNnoRendition in 2nd edition.

not match. Thus if the first tag in *tags* results in a NULL match, any rendition found from the second tag is placed into the first slot. If the number of matched renditions is less than the number of supplied *tags*, then memory for the returned array is reallocated to match the number of found renditions. In the absence of a XmNnoRenditionCallback callback, it is not possible to deduce the size of the returned rendition array.

The function allocates space for both the returned rendition array and the constituent renditions, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmRenditionFree`() on each of the elements in the returned array, and subsequently `XtFree`() on the array itself.

Example

The following specimen code illustrates the basic outline of a call to `XmRenderTableGetRenditions`():

```
XmRendition     *match_renditions;
XmStringTag     tags[MAX_TAGS];
int             i;

tags[0] = XmFONTLIST_DEFAULT_TAG;
tags[1] = XmS; /* "" */

...
/* search an unspecified render table */
match_renditions = XmRenderTableGetRenditions (render_table, tags,
MAX_TAGS);

/* use the matched set of renditions */
...

/* free the returned space */
if (match_renditions != NULL) {
    /* ASSUMPTION: XtNumber (match_renditions) == MAX_TAGS */
    /* Not a valid assumption if a tag does not match     */

    for (i = 0; i < MAX_TAGS; i++) {
        XmRenditionFree (match_renditions[i]);
    }

XtFree (match_renditions);
}
```

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableGetRendition`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionFree`(1), `XmRendition`(2).

Name

XmRenderTableGetTags – fetch the list of rendition tags from a render table.

Synopsis

int XmRenderTableGetTags (XmRenderTable *table*, XmStringTag **_tag_list*)

Inputs

table Specifies the render table.

Outputs

tag_list Returns the list of rendition tags.

Returns

The number of tags within the returned tag_list.

Availability

Motif 2.0 and later.

Description

`XmRenderTableGetTags`() is a convenience function which iterates through a render *table*, collecting all the tags from the individual renditions within the table, and returning them to the programmer. The number of tags placed at the address *tag_list* by the function is returned.

Usage

`XmRenderTableGetTags`() allocates an array, and places in the array a copy of the XmNtag resource for each rendition within the table. The array is returned at the address specified by the *tag_list* parameter. If the *table* is NULL, *tag_list* is initialized to NULL, and the function returns zero. It is the responsibility of the programmer to reclaim the space by calling `XtFree`() on each of the items within the allocated array, and then subsequently calling `XtFree`() on the array itself.

Example

The following specimen code illustrates the basic outline of a call to `XmRenderTableGetTags`():

```
XmStringTag *tags;
int         count, i;

/* fetch the tags from an unspecified render table */
count = XmRenderTableGetTags (render_table, &tags);

/* use the tags */
...

/* free the returned space */
```

```
if (tags != (XmStringTag *) 0) {
      for (i = 0; i < count; i++) {
            XtFree (tags[i]);
      }

      XtFree (tags);
}
```

See Also

`XmRenditionFree`(1), `XmRendition`(2).

Name

XmRenderTableRemoveRenditions – copy a render table, excluding specified renditions.

Synopsis

XmRenderTable XmRenderTableRemoveRenditions (XmRenderTable *old_table*,
XmStringTag **tags*,
int *tag_count*)

Inputs

old_table	Specifies a render table.
tags	Specifies an array of rendition tags. Any rendition which matches an item in the array is not copied from old_table.
tag_count	Specifies the number of items in the tags array.

Returns

A new render table with matching renditions removed.

Availability

Motif 2.0 and later.

Description

`XmRenderTableRemoveRenditions`() creates a new render table by copying from *old_table* only those renditions which do not have a tag matching items within the array *tags*. If *tags* is NULL, or if *tag_count* is zero, or if no renditions are removed, the function returns the *old_table* unmodified. Otherwise, *old_table* is deallocated, and the reference counts for any excluded renditions are decremented, before the function returns the newly allocated render table.

Usage

A rendition is not copied into the returned table if it has a XmNtag resource value the same as any item within the tags list. When the returned render table differs from the original *old_table* parameter, the function allocates space for the new table, and it is the responsibility of the programmer to reclaim the space by calling `XmRenderTableFree`().

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableFree`(1), `XmRendition`(2).

Name

XmRenditionCreate – create a rendition object.

Synopsis

XmRendition XmRenditionCreate (Widget *widget*, XmStringTag *tag*, Arg **arglist*, Cardinal *argcount*)

Inputs

widget	Specifies a widget.
tag	Specifies a tag for the rendition object.
arglist	Specifies an argument list, consisting of resource name/value pairs.
argcount	Specifies the number of arguments in arglist.

Returns

The new rendition object.

Availability

Motif 2.0 and later.

Description

`XmRenditionCreate`() creates a new rendition object, which can be used as an entry in a render table used for rendering XmStrings. *widget* is used to find a connection to the X server and an application context. *tag* is used as the XmNtag resource of the new rendition object. Resources for the new object are supplied in the *arglist* array.

Usage

The implementation of XmRendition is through a pseudo widget: although not a true widget, the object has resources and a resource style interface for setting and fetching values of the rendition. Typically, a rendition is merged into an existing render table through the function `XmRenderTableAddRenditions`(). Compound strings are rendered by successively matching tags within the compound string with the XmNtag resources of renditions in the table, and then using the resources of matched renditions to display the string components.

`XmRenditionCreate`() allocates storage for the returned rendition object. It is the responsibility of the programmer to reclaim the storage at a suitable point by calling `XmRenditionFree`(). Renditions are reference counted, and it is important to call `XmRenditionFree`() rather than `XtFree`() in order to maintain the references.

Example

The following specimen code creates a pair of renditions and merges them into an unspecified render table:

```
XmRendition       new_renditions[2];
```

```
XmRenderTable    new_table;
Arg              argv[4];
Cardinal         argc = 0;
Pixel            fg =...;
Pixel            bg =...;

/* create a rendition with fonts specified */
argc = 0;
XtSetArg (argv[argc], XmNfontName,        "fixed");
argc++;
XtSetArg (argv[argc], XmNfontType,        XmFONT_IS_FONT);
argc++;
XtSetArg (argv[argc], XmNloadModel,       XmLOAD_DEFERRED);
argc++;
new_renditions[0] = XmRenditionCreate (widget,
XmFONTLIST_DEFAULT_TAG, argv, argc);

/* create a rendition with line style specified */
argc = 0;
XtSetArg (argv[argc], XmNrenditionBackground,     bg);
argc++;
XtSetArg (argv[argc], XmNrenditionForeground,     fg);
argc++;
XtSetArg (argv[argc], XmNunderlineType,           XmSINGLE_LINE);
argc++;
XtSetArg (argv[argc], XmNstrikethruType,          XmSINGLE_LINE);
argc++;
new_renditions[1] = XmRenditionCreate (widget, "lineStyle", argv, argc);

/* merge into an unspecified render table */
new_table = XmRenderTableAddRenditions (old_table, new_renditions, 2,
XmMERGE_REPLACE);
```

See Also

`XmRenderTableAddRenditions`(1), `XmRenditionFree`(1), `XmRenditionRetrieve`(1), `XmRenditionUpdate`(1), `XmRendition`(2).

Name

XmRenditionFree – free the memory used by a rendition.

Synopsis

void XmRenditionFree (XmRendition *rendition*)

Inputs

rendition Specifies the rendition that is to be freed.

Availability

Motif 2.0 and later.

Description

`XmRenditionFree`() deallocates storage used by the specified *rendition*. The routine does not free any XFontSet or XFontStruct data structures associated with the rendition object.

Usage

`XmRenditionFree`() frees the storage used by the rendition object, but does not free font data structures associated with the XmNfont resource of the object. It is important to call `XmRenditionFree`() rather than `XtFree`() because Motif reference counts rendition objects. `XmRenditionFree`() decrements the reference count for the rendition; the rendition is not actually freed until the reference count reaches 0 (zero).

See Also

`XmRenditionCreate`(1), `XmRendition`(2).

Name

XmRenditionRetrieve – fetch rendition object resources.

Synopsis

void XmRenditionRetrieve (XmRendition *rendition*, Arg **arg_list*, Cardinal *arg_count*)

Inputs

rendition	Specifies the rendition whose resources are fetched.
arg_count	Specifies the number of arguments in arg_list.

Outputs

arg_list	Specifies an argument list, consisting of resource name/value pairs.

Availability

Motif 2.0 and later.

Description

`XmRenditionRetrieve`() fetches selective resource values of a *rendition* object. The set of resources retrieved is specified through the resource list *arg_list*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *arg_count*.

Usage

`XmRenditionRetrieve`() directly returns the values of the rendition resources, and not copies of them. The programmer should not inadvertently modify a returned value, but should take a copy of any pointer-valued resource which is to be changed. For example, the XmNtag and XmNfontName resources should be copied into a separate address space before modifying or manipulating the values.

If the XmNloadModel of the rendition object is XmLOAD_DEFERRED, and the font specified by the XmNfont resource is NULL, but the XmNfontName value is not NULL, and if the programmer has specified that the font is to be retrieved within *arg_list*, then `XmRenditionRetrieve`() automatically changes the load model to XmLOAD_IMMEDIATE and directly calls a procedure to load the font indicated by XmNfontName before returning the requested resource values.

Example

The following specimen code illustrates fetching resources from an unspecified rendition object:

```
Pixel       bg;
Pixel       fg;
XtPointer   font;
```

```
String          font_name;
XmFontType      font_type;
unsigned char   load_model;
unsigned char   strike_type;
XmTabList       tab_list;
XmStringTag     tag;
unsigned char   ul_type;
Arg             av[10];
Cardinal        ac = 0;

XtSetArg (av[ac], XmNrenditionForeground,   &fg);          ac++;
XtSetArg (av[ac], XmNrenditionBackground,   &bg);          ac++;
XtSetArg (av[ac], XmNfont,                  &font);        ac++;
XtSetArg (av[ac], XmNfontName,              &font_name);   ac++;
XtSetArg (av[ac], XmNfontType,              &font_type);   ac++;
XtSetArg (av[ac], XmNloadModel,             &load_model);  ac++;
XtSetArg (av[ac], XmNstrikethruType,        &strike_type); ac++;
XtSetArg (av[ac], XmNtabList,               &tab_list);    ac++;
XtSetArg (av[ac], XmNtag,                   &tag);         ac++;
XtSetArg (av[ac], XmNunderlineType,         &ul_type);     ac++;

XmRenditionRetrieve (rendition, av, ac);
```

See Also

`XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRenditionUpdate`(1), `XmRendition`(2).

Name

XmRenditionUpdate – set rendition object resources.

Synopsis

void XmRenditionUpdate (XmRendition *rendition*, Arg **arg_list*, Cardinal *arg_count*)

Inputs

rendition	Specifies the rendition whose resources are to be changed.
arg_list	Specifies an argument list, consisting of resource name/value pairs.
arg_count	Specifies the number of arguments within arg_list.

Availability

Motif 2.0 and later.

Description

`XmRenditionUpdate`() is a convenience function which sets the resources for a *rendition* object. The attributes to change are specified through an array of name/value pairs, similar to the resource-style interface of `XtSetValues`().

Usage

Modifying the value of the XmNfontName resource initially resets the XmNfont resource to NULL, irrespective of whether the load model for the new font is XmLOAD_IMMEDIATE or XmLOAD_DEFERRED.

Example

The following specimen code illustrates setting resources for an unspecified rendition object:

```
Pixel       bg =...;
Pixel       fg =...;
Arg         av[10];
Cardinal    ac = 0;

XtSetArg (av[ac], XmNrenditionForeground,       fg);
ac++;
XtSetArg (av[ac], XmNrenditionBackground,       bg);
ac++;
XtSetArg (av[ac], XmNfontType,                  XmFONT_IS_FONT);
ac++;
XtSetArg (av[ac], XmNfontName,                  "fixed");
ac++;
XtSetArg (av[ac], XmNloadModel,                 XmLOAD_DEFERRED);
ac++;
```

```
XtSetArg (av[ac], XmNstrikethruType,
ac++;
                                    XmSINGLE_LINE);
XtSetArg (av[ac], XmNunderlineType,
ac++;
                                    XmSINGLE_LINE);

XmRenditionUpdate (rendition, av, ac);
```

See Also

`XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRenditionRetrieve`(1), `XmRendition`(2).

Name

XmRepTypeAddReverse – install the reverse converter for a representation type.

Synopsis

#include <Xm/RepType.h>

void XmRepTypeAddReverse (XmRepTypeId *rep_type_id*)

Inputs

rep_type_id Specifies the ID number of the representation type.

Availability

Motif 1.2 and later.

Description

`XmRepTypeAddReverse`() installs a reverse converter for a previously registered representation type. The reverse converter converts numerical representation type values to string values. The *rep_type_id* argument specifies the ID number of the representation type. If the representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeAddReverse`() provides a way for an application to install a converter that converts numeric values to their string values.

See Also

`XmRepTypeGetId`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeGetId – get the ID number of a representation type.

Synopsis

#include <Xm/RepType.h>

XmRepTypeId XmRepTypeGetId (String *rep_type*)

Inputs

rep_type Specifies the string name of a representation type.

Returns

The ID number of the representation type or XmREP_TYPE_INVALID if the representation type is not registered.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetId`() retrieves the ID number of the specified representation type *rep_type* from the representation type manager. The *rep_type* string is the string name of a representation type that has been registered with `XmRepTypeRegister`(). `XmRepTypeGetId`() returns the ID number if the representation type has been registered. This value is used in other representation type manager routines to identify a particular type. Otherwise, the routine returns XmREP_TYPE_INVALID.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetId`() provides a way for an application get the ID of a representation type, which can be used to identify the type to other representation manager routine.

See Also

`XmRepTypeGetNameList`(1), `XmRepTypeGetRecord`(1), `XmRepTypeGetRegistered`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeGetNameList – get the list of value names for a representation type.

Synopsis

#include <Xm/RepType.h>

String * XmRepTypeGetNameList (XmRepTypeId *rep_type_id*, Boolean *use_uppercase_format*)

Inputs

rep_type_id Specifies the ID number of the representation type.
use_uppercase_format Specifies whether or not the names are in uppercase characters.

Returns

A pointer to an array of value names.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetNameList`() retrieves the list of value names associated with the specified *rep_type_id*. The routine returns a pointer to a NULL-terminated list of value names for the representation type, where each value name is a NULL-terminated string. If *use_uppercase_format* is True, the value names are in uppercase characters with Xm prefixes. Otherwise, the value names are in lowercase characters without Xm prefixes. `XmRepTypeGetNameList`() allocates storage for the returned data. The application is responsible for freeing the storage using `XtFree`() on each of the elements in the returned array, and subsequently upon the array pointer itself.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetNameList`() provides a way for an application to get the named values for a particular representation type.

See Also

`XmRepTypeGetId`(1), `XmRepTypeGetRecord`(1), `XmRepTypeGetRegistered`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeGetRecord – get information about a representation type.

Synopsis

#include <Xm/RepType.h>

XmRepTypeEntry XmRepTypeGetRecord (XmRepTypeId *rep_type_id*)

Inputs

rep_type_id Specifies the ID number of the representation type.

Returns

A pointer to a representation type entry structure.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetRecord`() retrieves information about the representation type specified by *rep_type_id*. The routine returns a XmRepTypeEntry, which is a pointer to a representation type entry structure. This structure contains information about the value names and values for the enumerated type. `XmRepTypeGetRecord`() allocates storage for the returned data. The application is responsible for freeing the storage using `XtFree`().

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetRecord`() provides a way for an application to retrieve information about a particular representation type.

Structures

The XmRepTypeEntry is defined as follows:

```
typedef struct {
    String          rep_type_name;      /* name of representation type     */
    String          *value_names;       /* array of value names            */
    unsigned char   *values;            /* array of numeric values         */
    unsigned char   num_values;         /* number of values                */
    Boolean         reverse_installed;  /* reverse converter installed flag */
    XmRepTypeId     rep_type_id;        /* representation type ID          */
```

} XmRepTypeEntryRec, *XmRepTypeEntry, XmRepTypeListRec, *XmRep-TypeList;

See Also

`XmRepTypeGetId(1), XmRepTypeGetNameList(1), XmRepTypeGetRegistered(1), XmRepTypeRegister(1).`

Name

XmRepTypeGetRegistered – get the registered representation types.

Synopsis

#include <Xm/RepType.h>

XmRepTypeList XmRepTypeGetRegistered (void)

Returns

A pointer to the registration list of representation types.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetRegistered`() retrieves the whole registration list for the representation type manager. The routine returns a copy of the registration list, which contains information about all of the registered representation types. The registration list is an array of XmRepTypeList structures, where each structure contains information about the value names and values for a single representation type. The end of the registration list is indicated by a NULL pointer in the *rep_type_name* field. `XmRepTypeGetRegistered` allocates storage for the returned data. The application is responsible for freeing this storage using `XtFree`(). The list of value names (the value of the *value_names* field), the list of values (the value of the *values* field), and the array of structures all need to be freed.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetRegistered`() *p*rovides a way for an application to get information about all of the registered representation types.

Example

The following code fragment shows the use of `XmRepTypeGetRegistered`() to print the value names and values of all of the registered representation types:

```
XmRepTypeList replist; int i;
replist = XmRepTypeGetRegistered();
```

```
while (replist->rep_type_name != NULL) {
    printf ("Representation type name: %s\n", replist->rep_type_name);
    printf ("Value names and associated values: \n");

    for (i = 0; i < replist->num_values; i++) {
        printf ("%s: ", replist->value_names[i]);
        printf ("%d\n", replist->values[i]);
    }

    replist++;
    XtFree ((char *)replist->values);
    XtFree ((char *)replist->value_names);
}

XtFree ((char *)replist);
```

Structures

The XmRepTypeList is defined as follows:

```
typedef struct {
    String          rep_type_name;      /* name of representation type
    */
    String          *value_names;       /* array of value names        */
    unsigned char   *values;            /* array of numeric values     */
    unsigned char   num_values;         /* number of values            */
    Boolean         reverse_installed;  /* reverse converter installed flag
    */
    XmRepTypeId     rep_type_id;        /* representation type ID      */
} XmRepTypeEntryRec, *XmRepTypeEntry, XmRepTypeListRec, *XmRep-
TypeList;
```

See Also

`XmRepTypeGetRecord`(1), `XmRepTypeGetNameList`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeInstallTearOffModelConverter – install the resource converter for the RowColumn XmNtearOffModel resource.

Synopsis

#include <Xm/RepType.h>

void XmRepTypeInstallTearOffModelConverter (void)

Availability

Motif 1.2 and later. In Motif 2.0 and later, the converter for the XmNtearOff-Model resource is internally installed, and this function is obsolete.

Description

`XmRepTypeInstallTearOffModelConverter`() installs the resource converter for the RowColumn XmNtearOffModel resource. This resource controls whether or not PulldownMenus and PopupMenus in an application can be torn off. Once the converter is installed, the value of XmNtearOffModel can be specified in a resource file.

Usage

In Motif 1.2, a RowColumn that is configured as a PopupMenu or a Pulldown-Menu supports tear-off menus. When a menu is torn off, it remains on the screen after a selection is made so that additional selections can be made. A menu pane that can be torn off contains a tear-off button at the top of the menu. The XmNtearOffModel resource controls whether or not tear-off functionality is available for a menu. This resource can take the values XmTEAR_OFF_ENABLED or XmTEAR_OFF_DISABLED.

In Motif 1.2, the resource converter for XmNtearOffModel is not installed by default. Some existing applications depend on receiving a callback when a menu is mapped; since torn-off menus are always mapped, these applications might fail if a user is allowed to enable tear-off menus from a resource file. `XmRepTy-peInstallTearOffModelConverter`() registers the converter that allows the resource to be set from a resource file.

See Also

`XmRowColumn`(2).

Name

XmRepTypeRegister – register a representation type resource.

Synopsis

```
#include <Xm/RepType.h>

XmRepTypeId XmRepTypeRegister (  String         rep_type,
                                 String         *value_names,
                                 unsigned char  *values,
                                 unsigned char  num_values)
```

Inputs

rep_type	Specifies the string name for the representation type.
value_names	Specifies an array of value names for the representation type. IP values 1i Specifies an array of values for the representation type.
num_values	Specifies the number of items in value_names and values.

Returns

The ID number of the representation type.

Availability

Motif 1.2 and later.

Description

`XmRepTypeRegister`() registers a representation type with the representation type manager. The representation type manager provides resource conversion facilities for enumerated values. `XmRepTypeRegister`() installs a resource converter that converts string values to numerical representation type values. The strings in the *value_names* array specify the value names for the representation type. The strings are specified in lowercase characters, with underscore characters separating words and without Xm prefixes.

If the *values* argument is NULL, the order of the strings in the *value_names* array determines the numerical values for the enumerated type. In this case, the names are assigned consecutive values starting with 0 (zero). If *values* is non-NULL, it is used to assign values to the names. Each name in the *value_names* array is assigned the corresponding value in the *values* array, so it is possible to have nonconsecutive values or duplicate names for the same value.

`XmRepTypeRegister`() returns the ID number that is assigned to the representation type. This value is used in other representation type manager routines to identify a particular type. A representation type can only be registered once. If a type is registered more than once, the behavior of the representation type manager is undefined.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeRegister`() provides a way for an application to register representation types for application-specific resources or for new widget classes.

See Also

`XmRepTypeAddReverse`(1), `XmRepTypeGetId`(1), `XmRepTypeGetNameList`(1), `XmRepTypeGetRecord`(1), `XmRepTypeGetRegistered`(1), `XmRepTypeValidValue`(1).

Name

XmRepTypeValidValue – determine the validity of a numerical value for a representation type.

Synopsis

```
#include <Xm/RepType.h>

Boolean XmRepTypeValidValue (  XmRepTypeId   rep_type_id,
                               unsigned char test_value,
                               Widget        enable_default_warning)
```

Inputs

rep_type_id	Specifies the ID number of the representation type.
test_value	Specifies the value that is to be tested.
enable_default_warning	Specifies a widget that is used to generate a default warning message.

Returns

True if the specified value is valid or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmRepTypeValidValue`() checks the validity of the specified *test_value* for the representation type specified by *rep_type_id*. The routine returns True if the value is valid. Otherwise, it returns False. If the *enable_default_warning* parameter is non-NULL, `XmRepTypeValidValue`() uses the specified widget to generate a default warning message if the value is invalid. If *enable_default_warning* is NULL, no default warning message is provided.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeValidValue`() provides a way for an application to check if a value is valid for a particular representation type.

See Also

`XmRepTypeGetId`(1), `XmRepTypeRegister`(1).

Name

XmResolveAllPartOffsets – ensure upward-compatible widgets and applications.

Synopsis

```
void XmResolveAllPartOffsets (  WidgetClass     widget_class,
                                XmOffsetPtr     *offset,
                                XmOffsetPtr     *constraint_offset)
```

Inputs

widget_class Specifies the widget class pointer.

Outputs

offset Returns the widget offset record.

constraint_offset Returns the constraint offset record.

Description

`XmResolveAllPartOffsets`() ensures that an application or a widget will be upwardly compatible with the records in a widget structure. In other words, if the size of a widget structure changes in the future, this routine can be used to calculate the locations of the new offsets. This routine and `XmResolvePartOffsets`() are similar. During the creation of a widget, both routines modify the widget structure by allocating an array of offset values. XmResolvePartOffsets() affects only the widget instance record, while `XmResolveAllPartOffsets`() affects the widget instance and constraint records.

Usage

If you are subclassing a Motif widget, you should use `XmResolveAllPartOffsets`() and `XmResolvePartOffsets`() to ensure that your widget will be compatible with future releases of the toolkit.

See Also

`XmResolvePartOffsets`(1).

Name

XmResolvePartOffsets – ensure upward-compatible widgets and applications.

Synopsis

void XmResolvePartOffsets (WidgetClass *widget_class*, XmOffsetPtr **offset*)

Inputs

widget_class Specifies the widget class pointer.

Outputs

offset Returns the widget offset record.

Description

`XmResolvePartOffsets`() ensures that an application or a widget will be upwardly compatible with the records in a widget structure. In other words, if the size of a widget structure changes in the future, this routine can be used to calculate the locations of the new offsets. This routine and `XmResolveAllPartOffsets`() are similar. During the creation of a widget, both routines modify the widget structure by allocating an array of offset values. `XmResolvePartOffsets`() affects only the widget instance record, while `XmResolveAllPartOffsets`() affects the widget instance and constraint records.

Usage

If you are subclassing a Motif widget, you should use `XmResolvePartOffsets`() and `XmResolveAllPartOffsets`() to ensure that your widget will be compatible with future releases of the toolkit.

See Also

`XmResolveAllPartOffsets`(1).

Name

XmScaleGetValue – get the slider value for a Scale widget.

Synopsis

#include <Xm/Scale.h>

void XmScaleGetValue (Widget *widget*, int **value_return*)

Inputs

widget	Specifies the Scale widget.

Outputs

value_return	Returns the current slider position for the Scale.

Description

`XmScaleGetValue`() returns the current position of the slider within the specified Scale *widget*.

Usage

`XmScaleGetValue`() is a convenience routine that returns the value of the XmNvalue resource for the Scale widget. Calling the routine is equivalent to calling `XtGetValues`() for that resource, although `XmScaleGetValue`() accesses the value through the widget instance structure rather than through `XtGetValues`().

See Also

`XmScaleSetValue`(1), `XmScale`(2).

Name

XmScaleSetTicks – set tick marks for a Scale widget.

Synopsis

```
#include <Xm/Scale.h>

void XmScaleSetTicks ( Widget      widget,
                       int         big_every,
                       Cardinal    num_med,
                       Cardinal    num_small,
                       Dimension   size_big,
                       Dimension   size_med,
                       Dimension   size_small)
```

Inputs

widget	Specifies a scale widget.
big_every	Specifies the number of scale values between large ticks.
num_med	Specifies the number of medium-sized ticks between the large tick marks.
num_small	Specifies the number of small-sized ticks between the medium-sized tick marks.
size_big	Specifies the size of the large ticks.
size_med	Specifies the size of the medium ticks.
size_small	Specifies the size of the small ticks.

Availability

Motif 2.0 and later.

Description

XmScaleSetTicks() places tick marks along the edges of a Scale widget. Ticks may be of three types: big, medium, and small, and the size (in pixels) of each type is specified by size_big, size_med, and size_small respectively. The location of each big tick is given by big_every, which simply specifies the number of scale values between each big tick. The number of medium-sized ticks between each big tick is given by num_med, and the number of small-sized ticks between each medium-sized tick is num_small.

Usage

XmScaleSetTicks() is a convenience function which places tick marks along the edge of a Scale by creating a series of SeparatorGadget children at evenly spaced intervals. If size_big is zero, XmScaleSetTicks() simply returns. If size_med or size_small is zero, num_med and num_small are forced to zero respectively. The number of medium and small tick marks required may be zero, but the number of large tick marks must not be less than 2.

SeparatorGadgets are created with the names "BigTic", "MedTic", and "SmallTic", the XmNseparatorType resource of each is forced to XmSINGLE_LINE. XmScaleSetTicks() does not delete any existing ticks when invoked on any particular Scale, neither does the Scale recalculate proper positions for the tick marks if the scale orientation is changed after tick marks are added. In each case, existing tick marks must be erased and subsequently redrawn or re-specified.

Example

The following code ensures that any tick marks are erased before adding new ticks to a Scale:

```
#include <Xm/Scale.h>
#include <Xm/SeparatoG.h>

void ScaleEraseSetTicks ( Widget      scale,
                          int         big_every,
                          Cardinal    num_med,
                          Cardinal    num_small,
                          Dimension   size_big,
                          Dimension   size_med,
                          Dimension   size_med)
{
    WidgetList    children     = (WidgetList) 0;
    Cardinal      num_children = (Cardinal) 0;
    int           i;
    String        name;

    /* fetch scale children. */
    XtVaGetValues (scale, XmNchildren, &children, XmNnumChildren,
    &num_children, 0)

    /* destroy old ticks.                */
    /* some optimization to reuse correctly   */
    /* placed ticks might be in order here... */
    for (i = 0; i < num_children; i++) {
        if (XmIsSeparatorGadget (children[i])) {
            if ((name = XtName (children[i])) != (String) 0) {
                if ((strcmp (name, "BigTic") == 0) ||
                    (strcmp (name, "MedTic") == 0) ||
                    (strcmp (name, "SmallTic") == 0)) {
                    XtDestroyWidget (children[i]);}
            }
        }
    }
```

```
    /* create new ticks. */
    XmScaleSetTicks (scale, big_every, num_med, num_small, size_big,
    size_med, size_small);
}
```

See Also

`XmScaleSetValues`(1). `XmScale`(2), `XmSeparatorGadget`(2).

Name

XmScaleSetValue – set the slider value for a Scale widget.

Synopsis

#include <Xm/Scale.h>

void XmScaleSetValue (Widget *widget*, int *value*)

Inputs

widget	Specifies the Scale widget.
value	Specifies the value of the slider.

Description

`XmScaleSetValue`() sets the current position of the slider to value in the specified Scale *widget*. The *value* must be in the range XmNminimum to XmNmaximum.

Usage

`XmScaleSetValue`() is a convenience routine that sets the value of the XmNvalue resource for the Scale widget. Calling the routine is equivalent to calling `XtSetValues`() for that resource, although `XmScaleSetValue`() accesses the value through the widget instance structure rather than through `XtSetValues`().

See Also

`XmScaleGetValue`(1), `XmScale`(2).

Name

XmScrollBarGetValues – get information about the current state of a ScrollBar widget.

Synopsis

```
#include <Xm/ScrollBar.h>

void XmScrollBarGetValues (  Widget    widget,
                             int       *value_return,
                             int       *slider_size_return,
                             int       *increment_return,
                             int       *page_increment_return)
```

Inputs

widget	Specifies the ScrollBar widget.

Outputs

value_return	Returns the current slider position.
slider_size_return	Returns the current size of the slider.
increment_return	Returns the current increment and decrement level.
page_increment_return	Returns the current page increment and decrement level.

Description

`XmScrollBarGetValues`() returns the current state information for the specified ScrollBar *widget*. This information consists of the position and size of the slider, as well as the increment and page increment values.

Usage

`XmScrollBarGetValues`() is a convenience routine that returns the values of the XmNvalue, XmNsliderSize, XmNincrement, and XmNpageIncrement resources for the ScrollBar widget. Calling the routine is equivalent to calling `XtGetValues`() for those resources, although `XmScrollBarGetValues`() accesses the values through the widget instance structure rather than through `XtGetValues`().

See Also

`XmScrollBarSetValues`(1), `XmScrollBar`(2).

Name

XmScrollBarSetValues – set the current state of a ScrollBar widget.

Synopsis

```
#include <Xm/ScrollBar.h>

void XmScrollBarSetValues (  Widget     widget,
                             int        value,
                             int        slider_size,
                             int        increment,
                             int        page_increment,
                             Boolean    notify)
```

Inputs

widget	Specifies the ScrollBar widget.
value	Specifies the slider position.
slider_size	Specifies the size of the slider.
increment	Specifies the increment and decrement level.
page_increment	Specifies the page increment and decrement level.
notify	Specifies whether or not the value changed callback is invoked.

Description

`XmScrollBarSetValues`() sets the current state of the specified ScrollBar *widget*. The position of the slider is set to *value*, which must be in the range XmNminimum to XmNmaximum minus XmNsliderSize. The size of the slider is set to *slider_size*, which must be between 1 and the size of the scroll region. The increment and page increment values are set to *increment* and *page_increment*, respectively.

If *notify* is True, `XmScrollBarSetValues`() invokes the XmNvalueChangedCallback for the ScrollBar when the state is set.

Usage

`XmScrollBarSetValues`() is a convenience routine that sets the values of the XmNvalue, XmNsliderSize, XmNincrement, and XmNpageIncrement resources for the ScrollBar widget. Calling the routine is equivalent to calling `XtSetValues`() for those resources, although `XmScrollBarSetValues`() accesses the values through the widget instance structure rather than through `XtSetValues`().

The *notify* parameter indicates whether or not the value changed callbacks for the ScrollBar are invoked. You can avoid redundant code by setting this parameter to True. If you are calling `XmScrollBarSetValues`() from a value changed

callback routine, you probably want to set the parameter to False to avoid the possibility of an infinite loop. Calling `XmScrollBarSetValues`() with *notify* set to True causes the callback routines to be invoked in a way that is indistinguishable from a user-initiated adjustment to the ScrollBar.

See Also

`XmScrollBarGetValues`(1), `XmScrollBar`(2).

Name

XmScrolledWindowSetAreas – specify the children for a scrolled window.

Synopsis

```
#include <Xm/ScrolledW.h>

void XmScrolledWindowSetAreas (Widget   widget,
                               Widget   horizontal_scrollbar,
                               Widget   vertical_scrollbar,
                               Widget   work_region)
```

Inputs

widget	Specifies the ScrolledWindow widget.
horizontal_scrollbar	Specifies the widget ID of the horizontal ScrollBar.
vertical_scrollbar	Specifies the widget ID of the vertical ScrollBar.
work_region	Specifies the widget ID of the work window.

Availability

In Motif 2.0 and later, `XmScrolledWindowSetAreas`() is obsolete.

Description

`XmScrolledWindowSetAreas`() sets up the standard regions of a ScrolledWindow widget for an application. The ScrolledWindow must be created before the routine is called. `XmScrolledWindowSetAreas`() specifies the horizontal and vertical ScrollBars and the work window region. If a particular ScrolledWindow does not have one of these regions, the corresponding argument can be specified as NULL.

Usage

Each of the ScrolledWindow regions is associated with a ScrolledWindow resource; `XmScrolledWindowSetAreas`() sets the associated resources. The resources that correspond to the last three arguments to the routine are XmNhorizontalScrollBar, XmNverticalScrollBar, and XmNworkWindow, respectively.

If an application does not call `XmScrolledWindowSetAreas`(), the widget may still set some of the standard regions. If ScrollBars are added as children, the XmNhorizontalScrollBar and XmNverticalScrollBar resources may be set if they have not already been specified. Any child that is not a ScrollBar is used for the XmNworkWindow. If you want to be certain about which widgets are used for the different regions, it is wise to call `XmScrolledWindowSetAreas`() explicitly.

In Motif 2.0 and later, the function is obsolete, and the programmer should specify the XmNhorizontalScrollBar, XmNverticalScrollBar, and XmNworkWindow

resources directly through a call to `XtSetValues`(). Although ostensibly maintained for backwards compatibility, the implementation of `XmScrolledWindowSetAreas`() in Motif 2.0 and later is not Motif 1.2 compatible. In Motif 1.2, supplying a NULL value for any of the scrollbar or work window parameters directly sets the internal component to NULL. In Motif 2.0 and later, supplying a NULL value causes that parameter to be ignored, leaving the internal component intact.

Example

The following code fragment shows how to set the regions of a ScrolledWindow:

```
Widget      toplevel, scrolled_w, drawing_a, vsb, hsb;
int         view_width, view_height;

scrolled_w = XtVaCreateManagedWidget ("scrolled_w", xmScrolledWindow-
WidgetClass, toplevel,
                                        XmNscrollingPolicy,
                                        XmAPPLICATION_DEFINED,
                                        XmNvisualPolicy, XmVARIA-
                                        BLE,
                                        NULL);
drawing_a = XtVaCreateManagedWidget ("drawing_a", xmDrawingAreaWidg-
etClass, scrolled_w,
                                        XmNwidth, view_width,
                                        XmNheight, view_height,
                                        NULL);
vsb = XtVaCreateManagedWidget ("vsb", xmScrollBarWidgetClass, scrolled_w,
                                    XmNorientation, XmVERTICAL,
                                    NULL);
hsb = XtVaCreateManagedWidget ("hsb", xmScrollBarWidgetClass, scrolled_w,
                                    XmNorientation, XmHORIZONTAL,
                                    NULL);

XmScrolledWindowSetAreas (scrolled_w, hsb, vsb, drawing_a);
```

See Also

`XmScrolledWindow`(2).

Name

XmScrollVisible – make an obscured child of a ScrolledWindow visible.

Synopsis

```
#include <Xm/ScrolledW.h>

void XmScrollVisible (  Widget      scrollw_widget,
                        Widget      widget,
                        Dimension   left_right_margin,
                        Dimension   top_bottom_margin)
```

Inputs

scrollw_widget	Specifies the ScrolledWindow widget.
widget	Specifies the widget ID of the widget that is to be made visible.
left_right_margin	Specifies the distance between the widget and the left or right edge of the viewport if the ScrolledWindow is scrolled horizontally.
top_bottom_margin	Specifies the distance between the widget and the top or bottom edge of the viewport if the ScrolledWindow is scrolled vertically.

Availability

Motif 1.2 and later.

Description

`XmScrollVisible`() scrolls the specified ScrolledWindow *scrollw_widget* so that the obscured or partially obscured *widget* becomes visible in the work area viewport. *widget* must be a descendent of *scrollw_widget*. The routine repositions the work area of the ScrolledWindow and sets the margins between the widget and the viewport boundaries based on *left_right_margin* and *top_bottom_margin* if necessary.

Usage

`XmScrollVisible`() provides a way for an application to ensure that a particular child of a ScrolledWindow is visible. In order for the routine to work, the XmNscrollingPolicy of the ScrolledWindow widget must be set to XmAUTOMATIC. This routine is designed to be used in the XmNtraverseObscureCallback for a ScrolledWindow.

See Also

`XmScrolledWindow`(2).

Name

XmSelectionBoxGetChild – get the specified child of a SelectionBox widget.

Synopsis

#include <Xm/SelectioB.h>

Widget XmSelectionBoxGetChild (Widget *widget*, unsigned char *child*)

Inputs

widget Specifies the SelectionBox widget.

child Specifies the child of the SelectionBox widget. Pass one of the values from the list below.

Returns

The widget ID of the specified child of the SelectionBox.

Availability

As of Motif 2.0, the toolkit abstract child fetch routines are marked for deprecation. You should give preference to `XtNameToWidget`(), except when fetching the SelectionBox default button or work area.

Description

`XmSelectionBoxGetChild`() returns the widget ID of the specified child of the SelectionBox widget.

Usage

XmDIALOG_APPLY_BUTTON, XmDIALOG_CANCEL_BUTTON, XmDIALOG_HELP_BUTTON, and XmDIALOG_OK_BUTTON specify the action buttons in the *widget*. XmDIALOG_DEFAULT_BUTTON specifies the current default button. XmDIALOG_LIST and XmDIALOG_LIST_LABEL specify the list and its label. XmDIALOG_TEXT and XmDIALOG_SELECTION_LABEL specify the selection text entry area and its label. XmDIALOG_SEPARATOR specifies the separator and XmDIALOG_WORK_AREA specifies any work area child that has been added to the SelectionBox. For more information on the different children of the SelectionBox, see the manual page in Section 2, *Motif and Xt Widget Classes*.

Widget Hierarchy

As of Motif 2.0, most Motif composite child fetch routines are marked as deprecated. However, since it is not possible to fetch the XmDIALOG_DEFAULT_BUTTON or XmDIALOG_WORK_AREA children using a public interface except through XmSelectionBoxGetChild(), the routine should not be considered truly deprecated. For consistency with the preferred new style, when fetching all other child values, consider giving preference to the

Intrinsics routine XtNameToWidget(), passing one of the following names as the second parameter:

“Apply”	(XmDIALOG_APPLY_BUTTON)
“Cancel”	(XmDIALOG_CANCEL_BUTTON)
“OK”	(XmDIALOG_OK_BUTTON)
“Separator”	(XmDIALOG_SEPARATOR)
“Help”	(XmDIALOG_HELP_BUTTON)
“Symbol”	(XmDIALOG_SYMBOL_LABEL)
“Message”	(XmDIALOG_MESSAGE_LABEL)
“*ItemsList”[1]	(XmDIALOG_LIST)
“Items”	(XmDIALOG_LIST_LABEL)
“Selection”	(XmDIALOG_SELECTION_LABEL)
“Text”	(XmDIALOG_TEXT)

Structures

The possible values for child are:

XmDIALOG_APPLY_BUTTON
XmDIALOG_OK_BUTTON
XmDIALOG_CANCEL_BUTTON
XmDIALOG_SELECTION_LABEL
XmDIALOG_DEFAULT_BUTTON XmDIALOG_SEPARATOR
XmDIALOG_HELP_BUTTON XmDIALOG_TEXT
XmDIALOG_LIST
XmDIALOG_WORK_AREA
XmDIALOG_LIST_LABEL

See Also

`XmPromptDialog`(2), `XmSelectionBox`(2).

1.The “*” is important: the List is not a direct child of the SelectionBox, but of a ScrolledList.

Name

XmSetColorCalculation – set the procedure that calculates default colors.

Synopsis

XmColorProc XmSetColorCalculation (XmColorProc *color_proc*)

Inputs

color_proc Specifies the procedure that is used for color calculation.

Returns

The previous color calculation procedure.

Description

`XmSetColorCalculation`() sets the procedure called by `XmGetColors`()[1] that calculates the default foreground, top and bottom shadow, and selection colors. The procedure calculates these colors based on the background color that has been passed to the procedure. If *color_proc* is NULL, this routine restores the default color calculation procedure. `XmSetColorCalculation`() returns the color calculation procedure that was in use when the routine was called. Both `XmGetColors`() and `XmChangeColor`() use the color calculation procedure.

Usage

Motif widgets rely on the use of shadowed borders to create their three-dimensional appearance. The top and bottom shadow colors are lighter and darker shades of the background color; these colors are reversed to make a component appear raised out of the screen or recessed into the screen. The select color is a slightly darker shade of the background color that indicates that a component is selected. The foreground color is either black or white, depending on which color provides the most contrast with the background color. `XmSetColorCalculation`() sets the procedure that calculates these colors. Use `XmGetColorCalculation`() to get the default color calculation procedure.

In Motif 2.0 and later, per-screen color calculation procedures are supported: if the XmNcolorCalculationProc resource of the XmScreen object associated with a given widget is not NULL, the procedure specified by the resource is used to calculate color in preference to any procedure which may have been specified by `XmSetColorCalculation`().

Procedures

The XmColorProc has the following syntax:

typedef void (*XmColorProc) (XColor *bg_color, /* specifies the background color */

1.Erroneously missing from 1st and 2nd editions.

```
                                XColor      *fg_color,    /* returns the fore-
ground color */
                                XColor      *sel_color,   /* returns the select
color */
                                XColor      *ts_color,    /* returns the top
shadow color */
                                XColor      *bs_color)    /* returns the bottom
shadow color */
```

An XmColorProc takes five arguments. The first argument, bg_color, is a pointer to an XColor structure that specifies the background color. The red, green, blue, and pixel fields in the structure contain valid values. The rest of the arguments are pointers to XColor structures for the colors that are to be calculated. The procedure fills in the red, green, and blue fields in these structures.

See Also

`XmChangeColor`(1), `XmGetColorCalculation`(1), `XmGetColors`(1). `XmScreen`(2).

Name

XmSetFontUnit – set the font unit values.

Synopsis

void XmSetFontUnit (Display **display*, int *font_unit_value*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

font_unit_value Specifies the value for both horizontal and vertical font units.

Availability

In Motif 1.2 and later, `XmSetFontUnit`() is obsolete. It has been superseded by setting the Screen resources XmNhorizontalFontUnit and XmNverticalFontUnit.

Description

`XmSetFontUnit`() sets the value of the horizontal and vertical font units for all of the screens on the display. This routine is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

Font units are a resolution-independent unit of measurement that are based on the width and height characteristics of a particular font. The default horizontal and vertical font unit values are based on the XmNfont resource, which in Motif 1.2, is a resource of the Screen object. An application can override these default values by calling `XmSetFontUnit`(). The values should be set before any widgets that use resolution-independent data are created.

See Also

`XmConvertUnits`(1), `XmSetFontUnits`(1), `XmGadget`(2), `XmManager`(2), `XmPrimitive`(2), `XmScreen`(2).

Name

XmSetFontUnits – set the font unit values.

Synopsis

void XmSetFontUnits (Display **display*, int *h_value*, int *v_value*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

h_value Specifies the value for horizontal font units.

v_value Specifies the value for vertical font units.

Availability

In Motif 1.2 and later, `XmSetFontUnits`() is obsolete. It has been superseded by setting the Screen resources XmNhorizontalFontUnit and XmNverticalFontUnit.

Description

`XmSetFontUnits`() sets the value of the horizontal and vertical font units to *h_value* and *v_value* respectively. The routine sets the font units for all of the screens on the display. This routine is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

Font units are a resolution-independent unit of measurement that are based on the width and height characteristics of a particular font. The default horizontal and vertical font unit values are based on the XmNfont resource, which in Motif 1.2 and later, is a resource of the Screen object. An application can override these default values by calling `XmSetFontUnits`(). The values should be set before any widgets that use resolution-independent data are created.

See Also

`XmConvertUnits`(1), `XmSetFontUnit`(1), `XmGadget`(2), `XmManager`(2), `XmPrimitive`(2), `XmScreen`(2).

Name

XmSetMenuCursor – set the current menu cursor.

Synopsis

void XmSetMenuCursor (Display **display*, Cursor *cursorId*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

cursorId Specifies the cursor ID for the menu cursor.

Availability

In Motif 1.2 and later, `XmSetMenuCursor`() is obsolete. It has been superseded by setting the Screen resource XmNmenuCursor.

Description

`XmSetMenuCursor`() sets the menu cursor for an application. The routine sets the cursor for all screens on the specified *display.* The specified cursor is shown whenever the application is using a Motif menu on the specified *display.* This routine is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

The menu cursor is the pointer shape that is used whenever a menu is posted. This cursor can be different from the normal pointer shape. In Motif 1.2 and later, the new Screen object has a resource, XmNmenuCursor, that specifies the menu cursor. `XmSetMenuCursor`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

See Also

`XmGetMenuCursor`(1), `XmScreen`(2).

Name

XmSetProtocolHooks – set prehooks and posthooks for a protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmSetProtocolHooks ( Widget          shell,
                          Atom            property,
                          Atom            protocol,
                          XtCallbackProc  prehook,
                          XtPointer       pre_closure,
                          XtCallbackProc  posthook,
                          XtPointer       post_closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocol	Specifies the protocol atom.
prehook	Specifies the procedure to invoke before the client callbacks.
pre_closure	Specifies any client data that is passed to the prehook.
posthook	Specifies the procedure to invoke after the client callbacks.
post_closure	Specifies any client data that is passed to the posthook.

Description

`XmSetProtocolHooks`() allows pre- and post-procedures to be invoked in addition to the regular callback procedures that are performed when the Motif window manager sends a protocol message. The prehook procedure is invoked before calling the procedures on the client's callback list, whereas the posthook procedure is invoked after calling the procedures on the client's callback list. This routine gives shells more control flow, since callback procedures aren't necessarily executed in any particular order.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. To communicate using a protocol, a client sends a ClientMessage event containing a *property* and *protocol*, and the receiving client responds by calling the associated protocol callback routine. `XmSetProtocolHooks`() gives an application more control over the flow of callback procedures, since callbacks are not necessarily invoked in any particular order.

See Also

`XmAddProtocolCallback`(1), `XmRemoveProtocolCallback`(1), `XmSetWMProtocolHooks`(1), `VendorShell`(2).

Name

XmSetWMProtocolHooks – set prehooks and posthooks for the XA_WM_PROTOCOLS protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmSetWMProtocolHooks ( Widget          shell,
                            Atom            protocol,
                            XtCallbackProc  prehook,
                            XtPointer       pre_closure,
                            XtCallbackProc  posthook,
                            XtPointer       post_closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
protocol	Specifies the protocol atom.
prehook	Specifies the procedure to invoke before the client callbacks.
pre_closure	Specifies any client data that is passed to the prehook.
posthook	Specifies the procedure to invoke after the client callbacks.
post_closure	Specifies any client data that is passed to the posthook.

Description

`XmSetWMProtocolHooks`()[1] is a convenience routine that calls `XmSetProtocolHooks`() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. To communicate using a protocol, a client sends a ClientMessage event containing a property and *protocol*, and the receiving client responds by calling the associated protocol callback routine. `XmSetWMProtocolHooks`() gives an application more control over the flow of callback procedures, since callbacks are not necessarily invoked in any particular order.

See Also

`XmAddWMProtocolCallback`(1), `XmInternAtom`(1), `XmRemoveWMProtocolCallback`(1), `XmSetProtocolHooks`(1), `VendorShell`(2).

1.Erroneously given as XmSetXmProtocolHooks() in 1st and 2nd editions.

Name

XmSimpleSpinBoxAddItem – add an item to a SimpleSpinBox.

Synopsis

#include <Xm/SSpinB.h>

void XmSimpleSpinBoxAddItem (Widget *widget*, XmString *item*, int *position*)

Inputs

widget	Specifies a SimpleSpinBox widget.
item	Specifies an item to add.
position	Specifies the position at which to add the new item.

Availability

Motif 2.1 and later.

Description

`XmSimpleSpinBoxAddItem`() adds an *item* to a SimpleSpinBox *widget* at a given *position* within the list of values which the widget may display. If *position* is zero, or if *position* is greater than the number of items in the list, the *item* is appended to the list of values.

Usage

`XmSimpleSpinBoxAddItem`() is a convenience routine that adds an *item* to the list of items which a SimpleSpinBox may display. In order to add an item to the SimpleSpinBox, a compound string must be created. `XmSimpleSpinBox-AddItem`() adds the *item* to the SimpleSpinBox by manipulating the XmNvalues, XmNnumValues, and XmNposition resources of the widget. If the XmNspinBoxChildType resource of the widget is not XmSTRING, or if the item is NULL, the procedure simply returns without modifying the array of values.

The SimpleSpinBox widget takes a copy of the supplied item; the programmer is responsible for freeing the compound string at an appropriate point by calling `XmStringFree`().

Example

The following procedure simply appends an item onto the end of a SimpleSpinBox list:

```
void SimpleSpinBoxAppend (Widget spinb, char *item)
{
   XmString xms = XmStringGenerate ((XtPointer)
   value,
                                    XmFONTLIST_DEFA
                                    ULT_TAG,
```

```
                                                XmCHARSET_TEXT,
                                                NULL);
        XmSimpleSpinBoxAddItem (spinb, xms, 0);
        XmStringFree (xms);
    }
```

See Also

XmSimpleSpinBoxDeletePos(1), XmSimpleSpinBoxSetItem(1), XmStringFree(1), XmSimpleSpinBox(2).

Name

XmSimpleSpinBoxDeletePos – delete an item at the specified position from a SimpleSpinBox.

Synopsis

#include <Xm/SSpinB.h>

void XmSimpleSpinBoxDeletePos (Widget *widget*, int *position*)

Inputs

widget	Specifies a SimpleSpinBox widget.
position	Specifies the position at which to delete an item.

Availability

Motif 2.1 and later.

Description

`XmSimpleSpinBoxDeletePos`() deletes an item at a given *position* from a SimpleSpinBox widget. A value of 1 indicates the first item, 2 is the second item, and so on. The last item in the list can be specified by passing a *position* of zero.

Usage

`XmSimpleSpinBoxDeletePos`() is a convenience function which deletes an item from the set of values associated with a SimpleSpinBox. The function directly manipulates the XmNvalues, XmNnumValues, and XmNposition resources of the widget. If the XmNspinBoxChildType resource of the widget is not XmSTRING, the function simply returns without modifying the array of values.

See Also

`XmSimpleSpinBoxAddItem`(1), `XmSimpleSpinBoxSetItem`(1), `XmSimpleSpinBox`(2).

Name

XmSimpleSpinBoxSetItem – set an item in a SimpleSpinBox.

Synopsis

#include <Xm/SSpinB.h>

void XmSimpleSpinBoxSetItem (Widget *widget*, XmString *item*)

Inputs

widget	Specifies a SimpleSpinBox widget.
item	Specifies the item to set.

Availability

Motif 2.1 and later.

Description

`XmSimpleSpinBoxSetItem`() makes an item in a SimpleSpinBox widget the current value.

Usage

`XmSimpleSpinBoxSetItem`() is a convenience routine that selects one of the SimpleSpinBox values. The *item* must exist within the XmNvalues array of the *widget*, otherwise a warning message is displayed. The function modifies the XmNposition resource of the widget if the item is found. No check is performed to ensure that the XmNspinBoxChildType resource of the SimpleSpinBox is XmSTRING.

See Also

`XmSimpleSpinBoxAddItem`(1), `XmSimpleSpinBoxDeletePos`(1), `XmSimpleSpinBox`(2).

Name

XmSpinBoxValidatePosition – validate the current value of a SpinBox.

Synopsis

#include <Xm/SpinB.h>

int XmSpinBoxValidatePosition (Widget *text_field*, int **position_value*)

Inputs

text_field Specifies a text field child of a SpinBox widget.

Outputs

position_value Returns the position of the current value.

Returns

The status of the validation.

Availability

Motif 2.1 and later.

Description

`XmSpinBoxValidatePosition`() checks that the *text_field* child of a SpinBox has a valid position value, and places the validated value of the text_field at the address *position_value*. If the position is valid, the function returns XmVALID_VALUE. Otherwise the function returns XmCURRENT_VALUE, XmMAXIMUM_VALUE, XmMINIMUM_VALUE, or XmINCREMENT_VALUE, depending upon a comparison of the current position and other constraint resources of the *text_field*.

Usage

`XmSpinBoxValidatePosition`() can be used to ensure that the user has entered a valid value into an editable textual child of a SpinBox. If *text_field* is NULL, or if *text_field* does not hold the XmQTaccessTextual trait, or if the XmNspinBoxChildType of this widget is not XmNUMERIC the function returns XmCURRENT_VALUE. The current value of the text field is fetched as a floating point number, then converted into an integer using the XmNdecimalPoints resource: digits after the decimal place are simply truncated. The current value is subsequently compared against the XmNminimumValue and XmNmaximumValue resources: if less than XmNminimumValue, *position_value* is set to the value of XmNminimumValue, and the function returns XmMINIMUM_VALUE, or if the current value is more than XmNmaximumValue, *position_value* is set to the value of XmNmaximumValue, and the function returns XmMAXIMUM_VALUE. Lastly, the function checks that the current value falls between XmNminimumValue and XmNmaximumValue on an

interval specified by the XmNincrementValue resource. That is, the current value is a member of the set:

```
{
        XmNminimumValue,
        XmNminimumValue + XmNincrementValue,
        XmNminimumValue + (2 * XmNincrementValue),
        XmNminimumValue + (3 * XmNincrementValue),
        ...
        XmNminimumValue + (n * XmNincrementValue),
        ...
        XmNmaximumValue
}
```

If the current value does not fall within the set, the *position_value* is set to the nearest item in the set which is not more than the current value, and the function returns XmINCREMENT_VALUE. If all checks pass, the *position_value* is set to the current value, and the function returns XmVALID_VALUE.

The SpinBox does not modify the contents of *text_field* when performing the validation.

Structures

The returned status has the following values:

XmCURRENT_VALUE	XmINCREMENT_VALUE
XmMAXIMUM_VALUE	XmMINIMUM_VALUE
XmVALID_VALUE	

See Also

`XmSpinBox`(2).

Name

XmStringBaseline – get the baseline spacing for a compound string.

Synopsis

Dimension XmStringBaseline (XmFontList *fontlist*, XmString *string*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Returns

The distance, in pixels, from the top of the character box to the baseline of the first line of text.

Availability

In Motif 2.0 and later, the XmFontList is obsolete. It is superseded by the XmRenderTable, to which it has become an alias.

Description

`XmStringBaseline`() returns the distance, in pixels, from the top of the character box to the baseline of the first line of text in *string*. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringBaseline`() provides information that is useful if you need to render a compound string. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringComponentCreate`(1), `XmStringExtent`(1), `XmStringHeight`(1), `XmStringWidth`(1), `XmRendition`(2).

Name

XmStringByteCompare – compare two compound strings byte-by-byte.

Synopsis

Boolean XmStringByteCompare (XmString *string1*, XmString *string2*)

Inputs

string1	Specifies a compound string.
string2	Specifies another compound string.

Returns

True if the two compound strings are byte-by-byte identical or False otherwise.

Description

`XmStringByteCompare`() compares the compound strings *string1* and *string2* byte by byte. If the strings are equivalent, it returns True; otherwise it returns False. If two compound strings are created with `XmStringCreateLocalized`() in the same language environment, using the same character string, the strings are byte-for-byte equal. Similarly, if two compound strings are created with `XmStringCreate`() using the same font list element tag and character string, the strings are equal.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringByteCompare`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

When a compound string is placed into a widget, the string is sometimes converted to an internal format, which provides faster processing but strips out redundant information. As a result, when an application retrieves the compound string from the widget by calling `XtGetValues`(), the returned string does not necessarily match the original string byte-for-byte. This situation occurs most often with Label widgets and its subclasses.

See Also

`XmStringComponentCreate`(1), `XmStringCompare`(1).

Name

XmStringByteStreamLength – calculates the length of a byte stream.

Synopsis

unsigned int XmStringByteStreamLength (unsigned char *_string_)

Inputs

string Specifies a string in byte stream format.

Returns

The length, in bytes, of the string.

Availability

Motif 2.0 and later.

Description

`XmStringByteStreamLength`() calculates and returns the length of a byte stream *string* in bytes, including any header information. The *string* is presumed to be a compound string which has been converted into byte stream format.

Usage

Since the returned value includes the size of the stream header, the function returns a non-zero value even if *string* is NULL. The function is primarily used as part of data transfer operations, for example in transferring compound string tables to and from the clipboard or other widgets.

See Also

`XmCvtXmStringToByteStream`(1),
`XmCvtByteStreamToXmString`(1).

Name

XmStringCompare – compare two compound strings.

Synopsis

Boolean XmStringCompare (XmString *string1*, XmString *string2*)

Inputs

string1	Specifies a compound string.
string2	Specifies another compound string.

Returns

True if the two compound strings are semantically equivalent or False otherwise.

Description

`XmStringCompare`() compares the compound strings *string1* and *string2* semantically. If the strings are equivalent, it returns True; otherwise it returns False. `XmStringCompare`() is similar to `XmStringByteCompare`() but less restrictive. Two compound string are semantically equivalent if they have the same text components, font list element tags, directions, and separators.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCompare`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringcomponentCreate`(1), `XmStringByteCompare`(1).

Name

XmStringComponentCreate – create a compound string consisting of a single component.

Synopsis

```
XmString XmStringComponentCreate (  XmStringComponentType  type,
                                    unsigned int           length,
                                    XtPointer              value)
```

Inputs

type	Specifies the type of component to create.
length	Specifies the length, in bytes, of value.
value	Specifies the value of the component.

Returns

A new compound string, or NULL.

Availability

Motif 2.0 and later.

Description

`XmStringComponentCreate`() creates a new compound string consisting of a component of the type specified by *type*, which contains the given *value*.

Usage

If *type* is not a valid component type, or if *length* is greater than zero and *value* is NULL, then the function returns NULL. Otherwise, the function returns an allocated compound string. It is the responsibility of the programmer to reclaim the utilized space at an appropriate point by calling `XmStringFree`().

Structures

The string component *type* can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_TAB
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN
XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP

XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

Example

The following code illustrates basic compound string creation by concatenating elements from an array of strings:

```
XmString create_xmstring_from_array (char **array, int count, Boolean tab)
{
    XmString                txt, sep;
    XmString                xms = (XmString) 0;
    XmStringComponentType   sep_type;
    int                     i;

    if (tab) {
        sep_type = XmSTRING_COMPONENT_TAB;
    }
    else {
        sep_type = XmSTRING_COMPONENT_SEPARATOR;
    }

    for (i = 0; i < count; i++) {
        txt = XmStringComponentCreate
                                        (XmSTRING_COMPONENT_TE
                                        XT, strlen (array[i]), (XtPointer)
                                        array[i]);
        xms = XmStringConcatAndFree (xms, txt);

        if (i < count) {
            /* another item after this... */
            sep = XmStringComponentCreate (sep_type, 0, NULL);
            xms = XmStringConcatAndFree (xms, sep);
        }
    }

    /* caller must free this */
    return xms;
}
```

See Also

`XmStringConcatAndFree`(1), `XmStringFree`(1).

Name

XmStringConcat – concatenate two compound strings.

Synopsis

XmString XmStringConcat (XmString *string1*, XmString *string2*)

Inputs

string1	Specifies a compound string.
string2	Specifies another compound string.

Returns

A new compound string.

Description

`XmStringConcat`() returns the compound string formed by appending *string2* to *string1*, leaving the original compound strings unchanged. Storage for the result is allocated within the routine and should be freed by calling `XmString-Free`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringConcat`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringComponentCreate`(1), `XmStringCopy`(1), `XmStringNConcat`(1), `XmStringNCopy`(1).

Name

XmStringConcatAndFree – concatenate two compound strings.

Synopsis

XmString XmStringConcatAndFree (XmString *string1*, XmString *string2*)

Inputs

string1 Specifies a compound string.
string2 Specifies another compound string.

Returns

A new compound string.

Availability

Motif 2.0 and later.

Description

`XmStringConcatAndFree`() is similar to `XmStringConcat`() in that each returns a compound string formed by appending *string2* to *string1*. `XmString-ConcatAndFree`() differs from `XmStringConcat`() by freeing the original compound strings. Storage for the result is allocated within the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, and locale components.

Example

The following code constructs a simple compound string out of piecemeal sub-components:

```
XmString xms;
XmString xms_temp;

xms = XmStringGenerate((XtPointer) “Multiple”,
                       XmFONTLIST_DEFAULT_TAG,
                       XmCHARSET_TEXT,
                       NULL);
xms_temp = XmStringComponentCreate
                                    (XmSTRING_COMPO
                                    NENT_TAB, 0,
                                    NULL);
xms = XmStringConcatAndFree(xms, xms_temp);
```

```
xms_temp = XmStringGenerate((XtPointer) “Column”,
                            XmFONTLIST_DEFAULT_TAG
                            ,
                            XmCHARSET_TEXT,
                            NULL);
xms = XmStringConcatAndFree (xms, xms_temp);
xms_temp = XmStringComponentCreate
                                  (XmSTRING_COMPO
                                  NENT_TAB, 0,
                                  NULL);
xms = XmStringConcatAndFree (xms, xms_temp);
xms_temp = XmStringGenerate((XtPointer) “Format”,
                            XmFONTLIST_DEFAULT_TAG
                            ,
                            XmCHARSET_TEXT,
                            NULL);
xms = XmStringConcatAndFree (xms, xms_temp);
```

See Also

XmStringComponentCreate(1), XmStringCopy(1), XmStringConcat(1), XmStringNConcat(1), XmStringNCopy(1).

Name

XmStringCopy – copy a compound string.

Synopsis

XmString XmStringCopy (XmString *string*)

Inputs

string Specifies a compound string.

Returns

A new compound string.

Description

`XmStringCopy`() copies the compound string *string* and returns the copy, leaving the original compound string unchanged. Storage for the result is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCopy`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringComponentCreate`(1), `XmStringConcat`(1), `XmStringNConcat`(1), `XmStringNCopy`(1).

Name

XmStringCreate – create a compound string.

Synopsis

XmString XmStringCreate (char *_text_, XmStringCharSet _tag_)

Inputs

text	Specifies the text component of the compound string.
tag	Specifies the font list element tag.

Returns

A new compound string.

Description

`XmStringCreate`() creates a compound string containing two components: a text component composed of *text* and the font list element tag specified by *tag*. *text* must be a NULL-terminated string. *tag* can have the value XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCreate`() allows you to create a compound string composed of a font list element tag and a text component.

In Motif 1.1, compound strings use character set identifiers rather than font list element tags. The character set identifier for a compound string can have the value XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

`XmStringCreate`() creates a compound string with no specified direction. The default direction may be taken from the XmNstringDirection resource of the parent of the widget that contains the compound string. If you need a string with a direction other than the default direction, use `XmStringDirectionCreate`() to create a direction string and concatenate it with the compound string containing the text.

Example

The following code fragment shows how to create compound strings using `XmStringCreate`():

```
Widget      toplevel;
XmString    s1, s2, s3, text, tmp;
String      string1 = "This is a string", string2 = "that contains three", string3 =
"separate fonts.";

s1 = XmStringCreate (string1, "tag1");
s2 = XmStringCreate (string2, "tag2");
s3 = XmStringCreate (string3, XmFONTLIST_DEFAULT_TAG);

tmp = XmStringConcatAndFree (s1, s2);
text = XmStringConcatAndFree (tmp, s3);

XtVaCreateManagedWidget ("widget_name", xmLabelWidgetClass, toplevel,
                                        XmNlabelString, text, NULL);
XmStringFree (text);
```

See Also

`XmStringBaseline`(1), `XmStringByteCompare`(1), `XmStringCompare`(1), `XmStringConcat`(1), `XmStringComponentCreate`(1), `XmStringCopy`(1), `XmStringCreateLocalized`(1), `XmStringCreateLtoR`(1), `XmStringCreateSimple`(1), `XmStringDirectionCreate`(1), `XmStringDraw`(1), `XmStringDrawImage`(1), `XmStringDrawUnderline`(1), `XmStringEmpty`(1), `XmStringExtent`(1), `XmStringFree`(1), `XmStringFreeContext`(1), `XmStringGetLtoR`(1), `XmStringGetNextComponent`(1), `XmStringGetNextSegment`(1), `XmStringHasSubstring`(1), `XmStringHeight`(1), `XmStringInitContext`(1), `XmStringLength`(1), `XmStringLineCount`(1), `XmStringNConcat`(1), `XmStringNCopy`(1), `XmStringPeekNextComponent`(1), `XmStringSegmentCreate`(1), `XmStringSeparatorCreate`(1), `XmStringWidth`(1).

Name

XmStringCreateLocalized – create a compound string in the current locale.

Synopsis

XmString XmStringCreateLocalized (String *text*)

Inputs

text Specifies the text component of the compound string.

Returns

A new compound string.

Availability

Motif 1.2 and later.

Description

`XmStringCreateLocalized`() creates a compound string containing two components: a text component composed of *text* and the font list element tag XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. *text* must be a NULL-terminated string. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCreateLocalized`() creates the identical compound string that would result from calling XmStringCreate with XmFONTLIST_DEFAULT_TAG as the font list entry tag.

Example

The following program shows how to create a compound string in the current locale and use it as the label for a PushButton:

```
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>

String fallbacks[] = { "*fontList:9x15=tag", NULL };

main (int argc, char *argv[])
{
    Widget          toplevel, rowcol;
    XtAppContext    app;
```

```
    XmString        text;

    XtSetLanguageProc (NULL, (XtLanguageProc) NULL, NULL);

    toplevel = XtVaAppInitialize (&app, argv[0], NULL, 0, &argc, argv, fall-
    backs, NULL);
    rowcol = XtVaCreateWidget ("rowcol", xmRowColumnWidgetClass,
    toplevel, NULL);

    text = XmStringCreateLocalized ("Testing, testing...");
    XtVaCreateManagedWidget ("pb", xmPushButtonWidgetClass, rowcol,
                                        XmNlabelString, text, NULL);

    XmStringFree (text);
    XtManageChild (rowcol);
    XtRealizeWidget (toplevel);
    XtAppMainLoop (app);
}
```

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringFree`(1).

Name

XmStringCreateLtoR – create a compound string.

Synopsis

XmString XmStringCreateLtoR (char **text*, XmStringCharSet *tag*)

Inputs

text	Specifies the text component of the compound string.
tag	Specifies the font list element tag.

Returns

A new compound string.

Availability

In Motif 2.0 and later, this function is obsolete, and is replaced by the function `XmStringGenerate`().

Description

`XmStringCreateLtoR`() creates a compound string containing two components: a text component composed of *text* and the font list element tag specified by *tag*. text must be a NULL-terminated string. In addition, `XmStringCreateLtoR`() searches for newline characters (\n) in *text*. Each time a newline is found, the characters up to the newline are placed into a compound string segment followed by a separator component. The routine does not add a separator component to the end of the compound string. The default direction of the string is left to right and the assumed encoding is 8-bit characters rather than 16-bit characters.

tag can have the value XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element *tag*, a string direction, and a *text* component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCreateLtoR`() allows you to create a compound string composed of a font list element *tag* and a multi-line *text* component.

In Motif 1.1, compound strings use character set identifiers rather than font list element tags. The character set identifier for a compound string can have the value XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringFree`(1). `XmStringGenerate`(1).

Name

XmStringCreateSimple – create a compound string in the current language environment.

Synopsis

XmString XmStringCreateSimple (char *_text_)

Inputs

text Specifies the text component of the compound string.

Returns

A new compound string.

Availability

In Motif 1.2, `XmStringCreateSimple`() is obsolete. It has been superseded by `XmStringCreateLocalized`().

Description

`XmStringCreateSimple`() creates a compound string containing two components: a text component composed of _text_ and a character set identifier derived from the LANG environment variable or from a vendor-specific default, which is usually ISO8859-1. _text_ must be a NULL-terminated string. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. In Motif 1.1, compound strings use character set identifiers rather than font list element tags. `XmStringCreateSimple`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringCreateLocalized`(1), `XmStringFree`(1).

Name

XmStringDirectionCreate – create a compound string containing a direction component.

Synopsis

XmString XmStringDirectionCreate (XmStringDirection *direction*)

Inputs

direction Specifies the value of the direction component. Pass either XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L or XmSTRING_DIRECTION_DEFAULT.

Returns

A new compound string.

Description

`XmStringDirectionCreate`() creates a compound string containing a single component, which is a direction component with the specified *direction* value. If the *direction* is XmSTRING_DIRECTION_DEFAULT, the widget where the compound string is rendered controls the direction. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDirectionCreate`() allows you to create a string direction component that can be concatenated with a compound string containing other components.

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringFree`(1).

Name

XmStringDirectionToDirection – converts a string direction to a direction.

Synopsis

XmDirection XmStringDirectionToDirection (XmStringDirection *string_direction*)

Inputs

string_direction Specifies the string direction to be converted.

Returns

The converted direction.

Availability

Motif 2.0 and later.

Description

`XmStringDirectionToDirection`() converts an XmStringDirection value specified by *string_direction* into an XmDirection value.

Usage

`XmStringDirectionToDirection`() converts between the XmStringDirection and XmDirection data types. If *string_direction* is XmSTRING_DIRECTION_LEFT_TO_RIGHT, the function returns XmLEFT_TO_RIGHT. If *string_direction* is XmSTRING_DIRECTION_RIGHT_TO_LEFT, the function returns XmRIGHT_TO_LEFT. Otherwise, the function returns XmDIRECTION_DEFAULT.

See Also

`XmStringDirectionCreate`(1).

Name

XmStringDraw – draw a compound string.

Synopsis

```
void XmStringDraw ( Display       *display,
                    Window        window,
                    XmFontList    fontlist,
                    XmString      string,
                    GC            gc,
                    Position      x,
                    Position      y,
                    Dimension     width,
                    unsigned char alignment,
                    unsigned char layout_direction,
                    XRectangle    *clip)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies the window where the string is drawn.
fontlist	Specifies the font list for drawing the string.
string	Specifies a compound string.
gc	Specifies the graphics context that is used to draw the string.
x	Specifies the x-coordinate of the rectangle that will contain the string.
y	Specifies the y-coordinate of the rectangle that will contain the string.
width	Specifies the width of the rectangle that will contain the string.
alignment	Specifies the alignment of the string in the rectangle. Pass one of the following values: XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, or XmALIGNMENT_END.
layout_direction	Specifies the layout direction of the string segments. Pass XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L, or XmSTRING_DIRECTION_DEFAULT.
clip	Specifies an clip rectangle that restricts the area where the string will be drawn.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringDraw`() draws the compound string specified by string by rendering the foreground pixels for each character. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDraw`() provides a means of rendering a compound string that is analogous to the Xlib string rendering routines. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget.

In Motif 1.2 or later, if a segment of a compound string is associated with a font list entry that is a font set, the font member of the *gc* is left in an undefined state by the underlying call to `XmbDrawString`(). If a segment of the compound string is not associated with a font set, the gc must contain a valid font member. The *gc* must be created using `XtAllocateGC`(); graphics contexts created with `XtGetGC`() are not valid.

See Also

`XmStringDrawImage`(1), `XmStringDrawUnderline`(1), `XmRendition`(2).

Name

XmStringDrawImage – draw a compound string.

Synopsis

```
void XmStringDrawImage ( Display        *display,
                         Window         window,
                         XmFontList     fontlist,
                         XmString       string,
                         GC             gc,
                         Position       x,
                         Position       y,
                         Dimension      width,
                         unsigned char  alignment,
                         unsigned char  layout_direction,
                         XRectangle     *clip)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies the window where the string is drawn.
fontlist	Specifies the font list for drawing the string.
string	Specifies a compound string.
gc	Specifies the graphics context that is used to draw the string.
x	Specifies the x-coordinate of the rectangle that will contain the string.
y	Specifies the y-coordinate of the rectangle that will contain the string.
width	Specifies the width of the rectangle that will contain the string.
alignment	Specifies the alignment of the string in the rectangle. Pass one of the following values: XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, or XmALIGNMENT_END.
layout_direction	Specifies the layout direction of the string segments. Pass XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L, or XmSTRING_DIRECTION_DEFAULT.
clip	Specifies an clip rectangle that restricts the area where the string will be drawn.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringDrawImage`() draws the compound string specified by *string* by painting the foreground and background pixels for each character. If string is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDrawImage`() provides a means of rendering a compound string that is analogous to the Xlib string rendering routines. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget.

In Motif 1.2 or later, if a segment of a compound string is associated with a font list entry that is a font set, the font member of the *gc* is left in an undefined state by the underlying call to `XmbDrawImageString`(). If a segment of the compound string is not associated with a font set, the *gc* must contain a valid font member. The *gc* must be created using `XtAllocateGC`(); graphics contexts created with `XtGetGC`() are not valid.

See Also

`XmStringDraw`(1), `XmStringDrawUnderline`(1), `XmRendition`(2).

Name

XmStringDrawUnderline – draw a compound string with an underlined sub-string.

Synopsis

```
void XmStringDrawUnderline (  Display        *display,
                              Window         window,
                              XmFontList     fontlist,
                              XmString       string,
                              GC             gc,
                              Position       x,
                              Position       y,
                              Dimension      width,
                              unsigned char  alignment,
                              unsigned char  layout_direction,
                              XRectangle     *clip,
                              XmString       underline)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies the window where the string is drawn.
fontlist	Specifies the font list for drawing the string.
string	Specifies a compound string.
gc	Specifies the graphics context that is used to draw the string.
x	Specifies the x-coordinate of the rectangle that will contain the string.
y	Specifies the y-coordinate of the rectangle that will contain the string.
width	Specifies the width of the rectangle that will contain the string.
alignment	Specifies the alignment of the string in the rectangle. Pass one of the following values: XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, or XmALIGNMENT_END.
layout_direction	Specifies the layout direction of the string segments. Pass XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L, or XmSTRING_DIRECTION_DEFAULT.
clip	Specifies an clip rectangle that restricts the area where the string will be drawn.

underline Specifies the substring that is to be underlined.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringDrawUnderline`() is similar to `XmStringDraw`(), but it also draws an underline beneath the first matching substring *underline* that is contained within *string*. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDrawUnderline`() provides a means of rendering a compound string and underlining a substring within it. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget.

In Motif 1.2 and later, if a segment of a compound string is associated with a font list entry that is a font set, the font member of the *gc* is left in an undefined state by the underlying call to `XmbDrawString`(). If a segment of the compound string is not associated with a font set, the *gc* must contain a valid font member. The *gc* must be created using `XtAllocateGC`(); graphics contexts created with `XtGetGC`() are not valid.

See Also

`XmStringDraw`(1), `XmStringDrawImage`(1), `XmRendition`(2).

Name

XmStringEmpty – determine whether there are text segments in a compound string.

Synopsis

Boolean XmStringEmpty (XmString *string*)

Inputs

string Specifies a compound string.

Returns

True if there are no text segments in the string or False otherwise.

Description

`XmStringEmpty`() returns True if no text segments exist in the specified *string* and False otherwise. If the routine is passed NULL, it returns True.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringEmpty`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringLength`(1), `XmStringLineCount`(1).

Name

XmStringExtent – get the smallest rectangle that contains a compound string.

Synopsis

void XmStringExtent (XmFontList *fontlist*, XmString *string*, Dimension **width*, Dimension **height*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Outputs

width	Returns the width of the containing rectangle.
height	Returns the height of the containing rectangle.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringExtent`() calculates the size of the smallest rectangle that can enclose the specified compound *string* and returns the *width* and *height* of the rectangle in pixels. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font from the character set of the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringExtent`() provides information that is useful if you need to render a compound string. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringBaseline`(1), `XmStringHeight`(1), `XmStringWidth`(1), `XmRendition`(2).

Name

XmStringFree – free the memory used by a compound string.

Synopsis

void XmStringFree (XmString *string*)

Inputs

string Specifies the compound string.

Description

`XmStringFree`() frees the memory used by the specified compound string.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. All of the routines that return a compound string allocate memory for the string. An application is responsible for this storage; `XmStringFree`() provides a way to free the memory.

When `XtGetValues`() is called for a resource that contains an XmString, a copy of the compound string is returned. The allocated storage is again the responsibility of the application and can be freed using `XmStringFree`().

Example

The following code fragment shows the use of `XmStringFree`():

```
Widget      toplevel, rowcol, pb;
XmString    str;
char        *text;

rowcol = XtVaCreateWidget ("rowcol", xmRowColumnWidgetClass, toplevel,
NULL);

str = XmStringCreateLocalized ("Testing, testing...");
pb = XtVaCreateManagedWidget ("pb", xmPushButtonWidgetClass, rowcol,
                                    XmNlabelString, str, NULL);
XmStringFree (str);
...
XtVaGetValues (pb, XmNlabelString, &str, NULL);
text = (char *) XmStringUnparse (str, NULL,
                                      XmCHARSET_TEXT,
                                      XmCHARSET_TEXT,
                                      NULL, 0, XmOUTPUT_ALL)[1];
```

```
printf ("PushButton's label is %s\n", text);
XmStringFree (str);
XtFree (text);
```

See Also

`XmStringCreate`(1), `XmStringCreateLocalized`(1), `XmStringCreateLtoR`(1), `XmStringCreateSimple`(1), `XmStringDirectionCreate`(1), `XmStringSegmentCreate`(1), `XmStringSeparatorCreate`(1).

1.Erroneously given as XmStringGetLtoR() in 2nd edition. XmStringGetLtoR() is deprecated from Motif 2.0 onwards.

Name

XmStringFreeContext – free a string context.

Synopsis

void XmStringFreeContext (XmStringContext *context*)

Inputs

context Specifies the string context that is to be freed.

Description

`XmStringFreeContext`() deallocates the string context structure specified by context.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. XmStringFreeContext() is the last of the string context routines that an application should call when processing a compound string, as it frees the string context data structure. An application begins by calling XmStringInitContext() to create a string context and then makes repeated calls to either XmStringGetNextComponent() or XmStringGetNextSegment() to cycle through the compound string.

The most common use of these routines is in converting a compound string to a regular character string when the compound string uses multiple fontlist element tags or it has a right-to-left orientation.

Example

The following code fragment shows how to convert a compound string into a character string:

```
XmString            str;
XmStringContext     context;
char                *text, buf[128], *p;
XmStringCharSet     tag;
XmStringDirection   direction;
Boolean             separator;

XtVaGetValues (widget, XmNlabelString, &str, NULL);

if (!XmStringInitContext (&context, str)) {
    XmStringFree (str);
    XtWarning ("Can’t convert compound string.");
    return;
```

```
}

/* p keeps a running pointer thru buf as text is read */ p = buf;
while (XmStringGetNextSegment (context, &text, &tag, &direction, &separa-
tor)) {
    /* copy text into p and advance to the end of the string */
    p += (strlen (strcpy (p, text)));

    if (separator == True) {
        /* if there's a separator... */
        *p++ = '\n';
        *p = 0; /* add newline and null-terminate */
    }
    XtFree (text); /* we're done with the text; free it */
}

XmStringFreeContext (context);
XmStringFree (str);

printf ("Compound string:\n%s\n", buf);
```

See Also

`XmStringInitContext`(1), `XmStringGetNextSegment`(1), `XmStringGetNextComponent`(1), `XmStringPeekNextComponent`(1).

Name

XmStringGenerate – generate a compound string.

Synopsis

```
XmString XmStringGenerate ( XtPointer     text,
                            XmStringTag   tag,
                            XmTextType    type,
                            XmStringTag   rendition)
```

Inputs

text	Specifies the data forming the value of the compound string.
tag	Specifies the tag used in creating the compound string.
type	Specifies the type of text.
rendition	Specifies a rendition tag.

Returns

A new compound string.

Availability

Motif 2.0 and later.

Description

`XmStringGenerate`() is a convenience function which invokes `XmStringParseText`() using a default parse table in order to convert *text* into a compound string. The default parse table maps tab characters to XmSTRING_COMPONENT_TAB, and newline characters to XmSTRING_COMPONENT_SEPARATOR components of the compound string. If a *rendition* tag is specified, the resulting compound string is placed within matching components of type XmSTRING_RENDITION_BEGIN and XmSTRING_RENDITION_END which contain the *rendition*. The type of the input *text* is specified by *type*, and is one of XmCHARSET_TEXT, XmWIDECHAR_TEXT, or XmMULTIBYTE_TEXT. *type* also specifies the type of the *tag* which is used in creating the compound string. If *tag* is NULL and the input *text* is of type XmCHARSET_TEXT, then the compound string is created with the *tag* set to XmFONTLIST_DEFAULT_TAG. If *tag* is NULL and the input *text* is of type XmWIDECHAR_TEXT or XmMULTIBYTE_TEXT, then the *tag* used is constructed from the value of _MOTIF_DEFAULT_LOCALE.

Usage

The function returns allocated storage, and it is the responsibility of the programmer to reclaim the space by calling `XmStringFree`() at an appropriate point.

In Motif 2.0 and later, in common with other objects, the compound string is manipulated as a reference counted data structure. XmString functions prior to Motif 2.0 handle ASN.1 strings, and the data structures are only used internally.

Example

The following code converts data taken from a Text widget into a compound string:

```
XmString convert_text (Widget text)
{
    /* ignoring widechar text values */
    char        *value = XmTextGetString (text);
    XmString    xms   = (XmString) 0;

    if (value) {
        xms = XmStringGenerate ((XtPointer) value,
                                XmFONTLIST_DEFAULT_TAG,
                                XmCHARSET_TEXT, NULL);
        XtFree (value);
    }
    /* caller must free this */
    return xms;
}
```

See Also

`XmStringFree(1), XmStringPutRendition`(1), `XmRendition`(2).

Name

XmStringGetLtoR – get a text segment from a compound string.

Synopsis

Boolean XmStringGetLtoR (XmString *string*, XmStringCharSet *tag*, char ***text*)

Inputs

string	Specifies the compound string.
tag	Specifies the font list element tag.

Outputs

text	Returns the NULL-terminated character string.

Returns

True if there is a matching text segment or False otherwise.

Availability

In Motif 2.0 and later, the function is obsolete, and is replaced by `XmStringUnparse`().

Description

`XmStringGetLtoR`() looks for a text segment in *string* that matches the font list element tag specified by *tag*. *tag* can have the value XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. The routine returns True if a text segment is found. *text* returns a pointer to the NULL-terminated character string that contains the text from the segment. Storage for the returned character string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringGetLtoR`() allows you to retrieve a character string from a compound string, so that you can use the string with the standard C string manipulation functions.

In Motif 1.1, compound strings use character set identifiers rather than font list element tags. The character set identifier for a compound string can have the value XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

`XmStringGetLtoR`() gets the first text segment from the compound string that is associated with the specified tag. If the *string* contains multiple font list element tags, you must cycle through the compound string and retrieve each segment individually in order to retrieve the entire string. The routine only gets strings with a left-to-right orientation.

See Also

`XmStringCreate`(1), `XmStringCreateLtoR`(1), `XmStringGetNextSegment`(1), `XmStringUnparse`(1).

Name

XmStringGetNextComponent – retrieves information about the next compound string
component.

Synopsis

```
XmStringComponentType
XmStringGetNextComponent (  XmStringContext          context,
                            char                     **text,
                            XmStringCharSet          *tag,
                            XmStringDirection        *direction,
                            XmStringComponentType    *unknown_tag,
                            unsigned short
*unknown_length,
                            unsigned char
**unknown_value)
```

Inputs

context	Specifies the string context for the compound string.

Outputs

text	Returns the NULL-terminated string for a text component.
tag	Returns the font list element tag for a tag component.
direction	Returns the string direction for a direction component.
unknown_tag	Returns the tag of an unknown component.
unknown_length	Returns the length of an unknown component.
unknown_value	Returns the value of an unknown component.

Returns

The type of the compound string component. The type is one of the values described below.

Availability

In Motif 2.0 and later, `XmStringGetNextComponent`() is obsolete, and is replaced by `XmStringGetNextTriple`().

Description

`XmStringGetNextComponent`() reads the next component in the compound string specified by *context* and returns the type of component found. The return value indicates which, if any, of the output parameters are valid. Storage for the returned values is allocated by the routine and must be freed by the application using `XtFree`().

For the type XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG, the font list element tag is returned in *tag*. In Motif 2.0 and later, the type

XmSTRING_COMPONENT_CHARSET is obsolete and is retained for compatibility with Motif 1.2. The type indicates that the character set identifier is returned in *tag*. XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG replaces XmSTRING_COMPONENT_CHARSET.

For the string component types XmSTRING_COMPONENT_TEXT and XmSTRING_COMPONENT_LOCALE_TEXT, the text string is returned in *text*. For XmSTRING_COMPONENT_DIRECTION, the direction is returned in *direction*. Only one of *tag*, *text*, and *direction* can be valid at any one time.

The type XmSTRING_COMPONENT_SEPARATOR indicates that the next component is a separator, while XmSTRING_COMPONENT_END specifies the end of the compound string. For type XmSTRING_COMPONENT_UNKNOWN, the tag, length, and value of the unknown component are returned in the corresponding arguments.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is called first to create the string context. `XmStringGetNextComponent`() cycles through the components in the compound string. When an application is done processing the string, it should call `XmStringFreeContext`() with the same context to free the allocated data.

Structures

A XmStringComponentType can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN

In Motif 2.0 and later, the following additional types are defined:

XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

See Also

XmStringFreeContext(1), XmStringGetNextTriple(1), XmStringGetNextSegment(1), XmStringInitContext(1), XmStringPeekNextComponent(1).

Name

XmStringGetNextSegment – retrieves information about the next compound string segment.

Synopsis

```
Boolean XmStringGetNextSegment (  XmStringContext   context,
                                  char              **text,
                                  XmStringCharSet   *charset,
                                  XmStringDirection *direction,
                                  Boolean           *separator)
```

Inputs

context	Specifies the string context for the compound string.

Outputs

text	Returns the NULL-terminated string for the segment.
tag	Returns the font list element tag for the segment.
direction	Returns the string direction for the segment.
separator	Returns whether or not the next component is a separator.

Returns

True if a valid segment is located or False otherwise.

Availability

In Motif 2.0 and later, the function is obsolete, and is replaced by `XmStringGetNextTriple`().

Description

`XmStringGetNextSegment`() retrieves the text string, font list element tag, and direction for the next segment of the compound string specified by *context*. The routine returns True if a valid segment is retrieved; otherwise, it returns False. Storage for the returned text is allocated by the routine and must be freed by the application using `XtFree`().

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is called first to create the string *context*. `XmStringGetNextSegment`() cycles through the segments in the compound string. The Boolean *separator* can be used to determine whether or not the next component in the compound string is a separator. When an application is done processing the string, it should call `XmStringFreeContext`() with the same context to free the allocated data.

The most common use of these routines is in converting a compound string to a regular character string when the compound string uses multiple fontlist element tags or it has a right-to-left orientation.

See Also

`XmStringFreeContext`(1), `XmStringGetLtoR`(1), `XmStringGetNextComponent`(1), `XmStringGetNextTriple`(1), `XmStringInitContext`(1), `XmStringPeekNextComponent`(1), `XmStringPeekNextTriple`(1).

Name

XmStringGetNextTriple – retrieve information about the next component.

Synopsis

XmStringComponentType
XmStringGetNextTriple (XmStringContext *context*, unsigned int **length*, XtPointer **value*)

Inputs

context Specifies the string context for the compound string.

Outputs

length Returns the length of the value of the component.
value Returns the value of the component.

Returns

The type of the component.

Availability

Motif 2.0 and later.

Description

`XmStringGetNextTriple`() is a convenience function which returns the type, *length*, and *value* of the next component within the compound string associated with *context*. The context is an opaque structure used for walking along compound strings one component at a time, and is initialized through a call to `XmStringInitContext`().

Usage

If either of *value* or *length* are NULL pointers, the function immediately returns XmSTRING_COMPONENT_END without fetching the next string segment. Otherwise, *value* is initially set to point to NULL, and *length* is reset to zero, and the next segment is processed. The function allocates memory for the returned value, which should be reclaimed at an appropriate point by calling `XtFree`().

Structures

An XmStringComponentType can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN

In Motif 2.0 and later, the following additional types are defined:

XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

Example

The following code fragment shows how to convert a compound string into a character string:

```
XmString                str;
XmStringContext         context;
char                    *text, buf[128], *p;
XmStringComponentType   type;
unsigned int            len;

/* Fetch the Compound String from somewhere */
XtVaGetValues (widget, XmNlabelString, &str, NULL);

if (!XmStringInitContext (&context, str)) {
    XmStringFree (str);
    XtWarning ("Can't convert compound string.");
    return;
}

/* p keeps a running pointer through buf as text is read */
p = buf;

/* Ignoring locale or widechar text for simplicity */
while ((type = XmStringGetNextTriple (context, &len, &text)) !=
XmSTRING_COMPONENT_END)
{
    switch (type) {
        case XmSTRING_COMPONENT_TAB             :
                        *p++ = '\t';
                        break;
        case XmSTRING_COMPONENT_SEPARATOR       :
                        *p++ = '\n';
                        *p   = '\0';
                        break;
        case XmSTRING_COMPONENT_TEXT            :
                        (void) strcpy (p, text);
```

```
                                p += len;
                                break;
        }
        XtFree (text);
}

XmStringFreeContext (context);
XmStringFree (str);
printf ("Compound string:\n%s\n", buf);
```

See Also

`XmStringFreeContext`(1), `XmStringGetNextComponent`(1), `XmStringGetNextSegment`(1), `XmStringInitContext`(1), `XmStringPeekNextComponent`(1), `XmStringPeekNextTriple`(1).

Name

XmStringHasSubstring – determine whether a compound string contains a substring.

Synopsis

Boolean XmStringHasSubstring (XmString *string*, XmString *substring*)

Inputs

string	Specifies the compound string.
substring	Specifies the substring.

Returns

True if *string* contains *substring* or False otherwise.

Description

`XmStringHasSubstring`() determines whether the compound string *substring* is contained within any single segment of the compound string *string*. *substring* must have only a single segment. The routine returns True if the *string* contains the *substring* and False otherwise.

If two compound strings are created with `XmStringCreateLocalized`() in the same language environment and they satisfy the above condition, `XmStringHasSubstring`() returns True. If two strings are created with `XmStringCreate`() using the same character set and they satisfy the condition, the routine also returns True. When comparing a compound string created by `XmStringCreate`() with a compound string created by `XmStringCreateSimple`() the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringHasSubstring`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringEmpty`(1), `XmStringLength`(1), `XmStringLineCount`(1).

Name

XmStringHeight – get the line height of a compound string.

Synopsis

Dimension XmStringHeight (XmFontList *fontlist*, XmString *string*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Returns

The height of the compound string.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringHeight`() returns the height, in pixels, of the specified compound *string*. If *string* contains multiple lines, where a separator component delimits each line, then the total height of all of the lines is returned. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font from the character set of the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringHeight`() provides information that is useful if you need to render a compound string. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringBaseline`(1), `XmStringExtent`(1), `XmStringWidth`(1), `XmRendition`(2).

Name

XmStringInitContext – create a string context.

Synopsis

Boolean XmStringInitContext (XmStringContext *_context_, XmString _string_)

Inputs

string Specifies the compound string.

Outputs

context Returns the allocated string context structure.

Returns

True if the string context is allocated or False otherwise.

Description

`XmStringInitContext`() creates a string context for the specified compound _string_. This string context allows an application to access the contents of a compound string.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is the first of the three string context routines that an application should call when processing a compound string, as it creates the string context data structure. The _context_ is passed to `XmStringGetNextTriple`() to cycle through the compound string. When an application is done processing the string, it should call `XmStringFreeContext`() with the same _context_ to free the allocated data.

The most common use of these routines is in converting a compound string to a regular character string when the compound string uses multiple fontlist element tags or it has a right-to-left orientation.

Example

The following code fragment shows how to convert a compound string into a character string:

```
XmString                str;
XmStringContext         context;
char                    *text, buf[128], *p;
XmStringComponentType   type;
unsigned int            len;

/* Fetch the Compound String from somewhere */
```

```
XtVaGetValues (widget, XmNlabelString, &str, NULL);

if (!XmStringInitContext (&context, str)) {
    XmStringFree (str);
    XtWarning ("Can't convert compound string.");
    return;
}

/* p keeps a running pointer through buf as text is read */
p = buf;

/* Ignoring locale or widechar text for simplicity */
while ((type = XmStringGetNextTriple (context, &len, &text)) !=
XmSTRING_COMPONENT_END)
{
    switch (type) {
        case XmSTRING_COMPONENT_TAB             :
            *p++ = '\t';
            break;
        case XmSTRING_COMPONENT_SEPARATOR       :
            *p++ = '\n';
            *p   = '\0';
            break;
        case XmSTRING_COMPONENT_TEXT            :
            (void) strcpy (p, text);
            p += len;
            break;
    }

    XtFree (text);
}

XmStringFreeContext (context);
XmStringFree (str);
printf ("Compound string:\n%s\n", buf);
```

See Also

`XmStringFreeContext`(1), `XmStringGetNextComponent`(1), `XmStringGetNextTriple`(1), `XmStringGetNextSegment`(1), `XmStringPeekNextComponent`(1), `XmStringPeekNextTriple`(1).

Name

XmStringIsVoid – determine whether there are valid segments in a compound string.

Synopsis

Boolean XmStringIsVoid (XmString *string*)

Inputs

string Specifies a compound string.

Returns

True if there are no segments in the string or False otherwise.

Availability

`XmStringIsVoid`() is available from Motif 2.0 or later.

Description

`XmStringIsVoid`() checks to see whether there any text, tab, or separator segments within the specified *string*. If the routine is passed NULL, it returns True. If the *string* contains text, tab or separator components, it returns False. Otherwise, it returns True.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, and locale components.

See Also

`XmStringEmpty`(1), `XmStringLength`(1), `XmStringLineCount`(1).

Name

XmStringLength – get the length of a compound string.

Synopsis

int XmStringLength (XmString *string*)

Inputs

string Specifies the compound string.

Returns

The length of the compound string.

Availability

In Motif 2.0 and later, the function is obsolete, and is replaced by `XmStringByteStreamLength`().

Description

`XmStringLength`() returns the length, in bytes, of the specified compound *string*. The calculation includes the length of all tags, direction indicators, and separators. The routine returns 0 (zero) if the structure of *string* is invalid.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringLength`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings. However, this routine cannot be used to get the length of the text represented by the compound string; it is not the same as strlen().

See Also

`XmStringByteStreamLength`(1), `XmStringEmpty`(1), `XmStringLineCount`(1).

Name

XmStringLineCount – get the number of lines in a compound string.

Synopsis

int XmStringLineCount (XmString *string*)

Inputs

string Specifies the compound string.

Returns

The number of lines in the compound string.

Description

XmStringLineCount() returns the number of lines in the specified compound *string*. The line count is determined by adding 1 to the number of separators in the string.

Usage

In Motif 1.2 and later, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringLineCount`() provides information that is useful in laying out components that display compound strings.

Example

The following routine shows how to read the contents of a file into a buffer and then convert the buffer into a compound string. The routine also returns the number of lines in the compound string:

```
XmString ConvertFileToXmString (char *filename, int *lines)
{
    struct stat    statb;
    int            fd, len, lines;
    char           *text;
    XmString       str;

    *lines = 0;

    if ((fd = open (filename, O_RDONLY)) < 0) {
        XtWarning ("internal error -- can't open file");
        return (XmString) 0;
    }

    if ((fstat (fd, &statb) == -1) || !(text = XtMalloc ((len = statb.st_size) + 1))) {
        XtWarning("internal error -- can't show text");
```

```
        (void) close (fd);
        return (XmString) 0;
    }

    (void) read (fd, text, len);
    text[len] = '\0';
    str = XmStringGenerate ((XtPointer) text,
                            XmFONTLIST_DEFAULT_TAG,
                            XmCHARSET_TEXT, NULL);[1]
    XtFree (text);
    (void) close (fd);
    *lines = XmStringLineCount (str);
    return str;
}
```

See Also

`XmStringEmpty`(1), `XmStringLength`(1).

1.Erroneously given as XmStringCreateLtoR() in 2nd edition. XmStringCreateLtoR() is deprecated from Motif 2.0 onwards.

Name

XmStringNConcat – concatenate a specified portion of a compound string to another compound string.

Synopsis

XmString XmStringNConcat (XmString *string1*, XmString *string2*, int *num_bytes*)

Inputs

string1	Specifies a compound string.
string2	Specifies the compound string that is appended.
num_bytes	Specifies the number of bytes of string2 that are appended.

Returns

A new compound string.

Availability

In Motif 2.0 and later, the function is obsolete, and is only maintained for backwards compatibility.

Description

`XmStringNConcat`() returns the compound string formed by appending bytes from *string2* to the end of *string1*, leaving the original compound strings unchanged. *num_bytes* of string are appended, which includes tags, directional indicators, and separators. Storage for the result is allocated within this routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

If *num_bytes* is less than the length of *string2*, the resulting string could be invalid. In this case, `XmStringNConcat`() appends as many bytes as possible, up to a maximum of *num_bytes*, to ensure the creation of a valid string.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringNConcat`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringConcat`(1), `XmStringCopy`(1), `XmStringNCopy`(1).

Name

XmStringNCopy – copy a specified portion of a compound string.

Synopsis

XmString XmStringNCopy (XmString *string*, int *num_bytes*)

Inputs

string	Specifies a compound string.
num_bytes	Specifies the number of bytes of string that are copied.

Returns

A new compound string.

Availability

In Motif 2.0 and later, the function is obsolete, and is only maintained for backwards compatibility.

Description

`XmStringNCopy`() copies *num_bytes* bytes from the compound string *string* and returns the resulting copy, leaving the original string unchanged. The number of bytes copied includes tags, directional indicators, and separators. Storage for the result is allocated within this routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

If *num_bytes* is less than the length of string, the resulting string could be invalid. In this case, `XmStringNCopy`() copies as many bytes as possible, up to a maximum of *num_bytes* to ensure the creation of a valid string.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringNCopy`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringConcat`(1), `XmStringCopy`(1), `XmStringNConcat`(1).

Name

XmStringParseText – convert a string to a compound string.

Synopsis

```
XmString XmStringParseText (  XtPointer      text,
                              XtPointer      *text_end,
                              XmStringTag    tag,
                              XmTextType     type,
                              XmParseTable   parse_table,
                              Cardinal       parse_count,
                              XtPointer      client_data)
```

Inputs

text	Specifies a string to be converted.
text_end	Specifies a pointer into text where parsing is to finish.
tag	Specifies the tag to be used in creating the compound string.
type	Specifies the type of the text and the tag.
parse_table	Specifies a table used for matching characters in the input text.
parse_count	Specifies the number of items in the parse_table.
client_data	Specifies application data to pass to any parse procedures within the parse_table.

Outputs

text_end	Returns a location within the text where parsing finished.

Returns

The converted compound string.

Availability

Motif 2.0 and later.

Description

`XmStringParseText`() converts the string specified by *text* into a compound string. A *parse_table* can be specified which consists of a set of mappings to control the conversion process. The contents of the string to be converted can be in one of a number of formats: simple characters, multibyte, or wide characters. The *type* parameter specifies the type of the input *text*, and is also used to interpret the *tag* which is used in creating text components within the returned compound string. *text_end* is both an input and an output parameter: as an input parameter, it specifies a location within *text* where parsing is to terminate; as an output parameter, it points to a location within *text* where parsing actually finished. Supplying NULL for *text_end* is interpreted to mean that parsing should stop at the occurrence of a null byte.

Usage

If *type* is XmCHARSET_TEXT, the input *text* is assumed to consist of a simple array of characters, and the *tag* is interpreted as the name of a charset to use in constructing the returned compound string. If *tag* is NULL, a default charset using XmFONTLIST_DEFAULT_TAG is used.

If the *type* is XmMULTIBYTE_TEXT or XmWIDECHAR_TEXT, the input *text* is assumed to be in multibyte or widechar text format respectively, and the *tag* is interpreted as a locale specifier. The *tag* should either be specified as NULL or _MOTIF_DEFAULT_LOCALE: if NULL, a locale component with a value of _MOTIF_DEFAULT_LOCALE is created in any case.

A parse table can be specified for controlling the conversion process. A parse table consists of a set of XmParseMapping objects, which have match pattern, substitution pattern and parse procedure components. The head of the input stream is compared against elements within the parse table, and if there is a correspondence between the input and a parse mapping match pattern, the parse mapping object is used to construct the output compound string at that point in the conversion, either by directly inserting the substitution pattern, or by invoking the parse procedure of the mapping object. The parse mapping specifies how the input pointer is advanced, and the process is repeated, comparing the head of the input against the parse table. At the end of the conversion, the *text_end* parameter is set to point to the location within the input *text* where parsing actually terminated. Depending upon the way in which the parse table interacts with the input *text*, the returned *text_end* may not be the same location which the programmer specified if more or less of the input *text* is consumed.

An implicit automatic conversion takes place where there is no matching parse mapping object for the head of the input. In other words, it is not necessary to provide a parse table to convert everything: parse tables are only required where specific inputs need to be handled specially. `XmStringParseText`() uses an internal parse mapping which handles changes in string direction in the absence of a supplied mapping for the task. A parse mapping handles string direction changes if the XmNpattern resource of the object is equal to XmDIRECTION_CHANGE.

`XmStringParseText`() allocates memory for the returned compound string. It is the responsibility of the programmer to reclaim the space through a call to `XmStringFree`() at an appropriate point.

Example

The following specimen code converts an input string containing tab and newline characters into a compound string:

```
XmParseTable    parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
XmString        tmp;
XmString        output;
Arg             av[4];
Cardinal        ac;

/* map \t to a tab component */
tmp = XmStringComponentCreate (XmSTRING_COMPONENT_TAB, 0,
NULL);
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\t");         ac++;
parse_table[0] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);

/* map \n to a separator component */
tmp = XmStringSeparatorCreate();
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\n");         ac++;
parse_table[1] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);

/* convert the (unspecified) input string into a compound string *
 output = XmStringParseText (input, NULL, NULL, XmCHARSET_TEXT,
parse_table, 2, NULL);
XmParseTableFree (parse_table);
```

See Also

`XmStringFree(1), XmStringGenerate`(1), `XmStringUnparse`(1). `XmParseMapping`(2).

Name

XmStringPeekNextComponent – returns the type of the next compound string component.

Synopsis

XmStringComponentType XmStringPeekNextComponent (XmStringContext *context*)

Inputs

context Specifies the string context for the compound string.

Returns

The type of the compound string component. The type is one of the values described below.

Availability

In Motif 2.0 and later, the function is obsolete, and `XmStringPeekNextTriple`() is preferred.

Description

`XmStringPeekNextComponent`() checks the next component in the compound string specified by *context* and returns the type of the component found. The routine shows what would be returned by a call to `XmStringGetNextComponent`(), without actually updating *context*.

The returned type XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG indicates that the next component is a font list element tag. In Motif 1.2, the type XmSTRING_COMPONENT_-CHARSET is obsolete and is retained for compatibility with Motif 1.1. The type indicates that the next component is a character set identifier. XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG replaces XmSTRING_COMPONENT_CHARSET.

The types XmSTRING_COMPONENT_TEXT and XmSTRING_COMPONENT_LOCALE_TEXT specify that the next component is text. XmSTRING_COMPONENT_DIRECTION indicates that the next component is a string direction component.

The type XmSTRING_COMPONENT_SEPARATOR indicates that the next component is a separator, while XmSTRING_COMPONENT_END specifies the end of the compound string. The type XmSTRING_COMPONENT_UNKNOWN, indicates that the type of the next component is unknown.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is called first to create the string context. `XmStringPeekNextComponent`() peeks at the next component in the compound string without cycling through the component. When an application is done processing the string, it should call `XmStringFreeContext`() with the same context to free the allocated data.

Structures

A XmStringComponentType can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN

In Motif 2.0 and later, the following additional types are defined:

XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

See Also

`XmStringFreeContext`(1), `XmStringGetNextComponent`(1), `XmStringGetNextSegment`(1), `XmStringPeekNextTriple`(1), `XmStringInitContext`(1).

Name

XmStringPeekNextTriple – retrieve the type of the next component.

Synopsis

XmStringComponentType XmStringPeekNextTriple (XmStringContext *context*)

Inputs

context Specifies the string context for the compound string.

Returns

The type of the next component.

Availability

Motif 2.0 and later.

Description

`XmStringPeekNextTriple`() returns the type of the next component without updating the compound string *context*.

Usage

An XmStringContext is an opaque data type which is used for walking along a compound string one component at a time. It is initialized by a call to XmStringInitContext. Each successive call to `XmStringGetNextComponent`() adjusts the string context to point to the next component. `XmStringPeekNextTriple`() returns the type of the next component without adjusting the *context*, and thus it can be used to look ahead into the compound string.

Structures

The string component type can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN
XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

See Also

XmStringFreeContext(1), XmStringGetNextComponent(1), XmStringGetNextSegment(1), XmStringInitContext(1), XmStringPeekNextComponent(1).

Name

XmStringPutRendition – add rendition components to a compound string.

Synopsis

XmString XmStringPutRendition (XmString *string*, XmStringTag *rendition*)

Inputs

string	Specifies a compound string which requires rendition components.
rendition	Specifies a tag used to create the rendition components.

Returns

A newly allocated compound string with rendition components.

Availability

Motif 2.0 and later.

Description

`XmStringPutRendition`() is a convenience function which places XmSTRING_COMPONENT_RENDITION_BEGIN and XmSTRING_COMPONENT_RENDITION_END components containing *rendition* around a compound string. The *string* is not modified by the procedure, which takes a copy.

Usage

`XmStringPutRendition`() allocates space for the returned compound string, and it is the responsibility of the programmer to reclaim the space at an appropriate point by calling `XmStringFree`().

See Also

`XmStringFree`(1).

Name

XmStringSegmentCreate – create a compound string segment.

Synopsis

```
XmString XmStringSegmentCreate ( char              *text,
                                 XmStringCharSet   tag,
                                 XmStringDirection direction,
                                 Boolean           separator)
```

Inputs

text	Specifies the text component of the compound string segment.
tag	Specifies the font list element tag.
direction	Specifies the value of the direction component. Pass either XmSTRING_DIRECTION_L_TO_R or XmSTRING_DIRECTION_R_TO_L.
separator	Specifies whether or not a separator is added to the compound string.

Returns

A new compound string.

Availability

In Motif 2.0 and later, the function is deprecated. A combination of `XmStringComponentCreate`() and `XmStringConcat`() is preferred.

Description

`XmStringSegmentCreate`() creates a compound string segment that contains the specified *text*, *tag*, and *direction*. If *separator* is True, a separator is added to the segment, following the *text*. If *separator* is False, the compound string segment does not contain a separator. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringSegmentCreate`() allows you to create a single segment that can be concatenated with a compound string containing other segments.

See Also

`XmStringCreate`(1), `XmStringFree`(1).

Name

XmStringSeparatorCreate – create a compound string containing a separator component.

Synopsis

XmString XmStringSeparatorCreate (void)

Returns

A new compound string.

Description

`XmStringSeparatorCreate`() creates and returns a compound string containing a separator as its only component.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringSeparatorCreate`() allows you to create a separator component that can be concatenated with a compound string containing other components.

See Also

`XmStringCreate`(1), `XmStringFree`(1), `XmStringSegmentCreate`(1).

Name

XmStringTableParseStringArray – convert an array of strings into a compound string table.

Synopsis

```
XmStringTable XmStringTableParseStringArray (  XtPointer       *strings,
                                               Cardinal        count,
                                               XmStringTag     tag,
                                               XmTextType      type,
                                               XmParseTable
parse_table,
                                               Cardinal
parse_count,
                                               XtPointer
client_data)
```

Inputs

strings	Specifies an array of strings.
count	Specifies the number of items in strings.
tag	Specifies the tag used to create the resulting compound string table.
type	Specifies the type of each input string, and the tag.
parse_table	Specifies a parse table to control the conversion process.
parse_count	Specifies the number of parse mappings in parse_table.
client_data	Specifies application data to pass to parse procedures within the parse_table.

Returns

An array of compound strings.

Availability

Motif 2.0 and later.

Description

`XmStringTableParseStringArray`() converts an array of strings into an array of compound strings. `XmStringTableParseStringArray`() is no more than a convenience function which allocates space for an table of compound strings, and subsequently calls `XmStringParseText`() iteratively on each item within the *strings* array to convert the item into a compound string. Each converted item is placed within the allocated table at a corresponding location to its position in the *strings* array. A *parse_table* can be specified which consists of a set of mappings to control the conversion process. The contents of each of the strings to be converted can be in one of a number of formats: simple characters, multibyte, or wide characters. The *type* parameter specifies the type

of the input strings, and is also used to interpret the *tag* which is used in creating text components within the returned compound string array.

Usage

The function calls XmStringParseText() passing NULL as the *text_end* (second) parameter: each item within the array of strings is converted until the occurrence of a terminating null byte. XmStringTableParseStringArray() returns allocated storage: the elements within the returned table are compound strings allocated by the internal call to XmStringParseText(), and these should each be freed at an appropriate point through XmStringFree(). XmStringTableParseStringArray() also allocates space for the table itself, and this should subsequently be freed using XtFree().

Structures

The XmTextType *type* parameter can take one of the following values:

XmCHARSET_TEXT
XmMULTIBYTE_TEXT
XmWIDECHAR_TEXT

See Also

XmStringFree(1), XmStringGenerate(1), XmStringParseText(1), XmStringTableUnparse(1), XmParseMapping(2).

Name

XmStringTableProposeTablist – create a tab list for a compound string table.

Synopsis

```
XmTabList XmStringTableProposeTablist (  XmStringTable  strings,
                                         Cardinal       string_count,
                                         Widget         widget,
                                         float          padding,
                                         XmOffsetModel
offset_model)
```

Inputs

strings	Specifies an array of compound strings.
string_count	Specifies the number of items in *strings*.
widget	Specifies a widget from which rendition information is calculated.
padding	Specifies a separation between columns.
offset_model	Specifies whether tabs are created at absolute or relative offsets.

Returns

A new XmTabList.

Availability

Motif 2.0 and later.

Description

`XmStringTableProposeTablist`() creates an XmTabList value which can be used to specify how an array of tabbed compound *strings* is aligned into columns.

A compound string is tabbed if it contains an XmSTRING_COMPONENT_TAB component between textual components: each text component forms an individual column entry. The strings are rendered with respect to a tab list: each tab contains a floating point offset which specifies the starting location of a column. `XmStringTableProposeTablist`() creates a tab list appropriate for laying out the given strings in a multi-column format.

The XmNunitType resource of widget is used to calculate the units in which the tab calculation is performed. Extra spacing between each column is specified by the *padding* parameter, and this is also interpreted in terms of the unit type of *widget*. The *offset_model* determines whether the floating point positions calculated for each tab in the returned XmTabList are at absolute locations (XmABSOLUTE), or relative to the previous tab (XmRELATIVE).

Usage

The tab list created by XmStringTableProposeTablist() can be applied to the render table of the widget where the strings are to be displayed by modifying the XmNtabList resource of an existing rendition through the procedure XmRenditionUpdate(). Alternatively, a new rendition can be created using XmRenditionCreate(), and thereafter merged into the widget render table using XmRenderTableAddRenditions().

XmStringTableProposeTablist() returns allocated storage, and it is the responsibility of the programmer to reclaim the allocated space at a suitable point by calling XmTabListFree().

If no render table is associated with *widget*, XmStringTableProposeTablist() invokes internal routines to deduce a default render table: these routines are not multi-thread safe.

See Also

XmTabCreate(1), XmTabFree(1), XmTabListCopy(1), XmTabListFree(1), XmRenderTableAddRenditions(1), XmRenditionCreate(1), XmRenditionUpdate(1), XmRendition(2).

Name

XmStringTableToXmString – convert compound string table to compound string.

Synopsis

XmString
XmStringTableToXmString (XmStringTable *table*, Cardinal *count*, XmString *break_component*)

Inputs

table	Specifies an array of compound strings.
count	Specified the number of items in the table.
break_component	Specifies a compound string used to separate converted table items.

Returns

A compound string.

Availability

Motif 2.0 and later.

Description

`XmStringTableToXmString`() is a convenience function which converts a *table* of compound strings into a single compound string. A `break_component` can be inserted between each component converted from the *table* in order to separate each.

Usage

`XmStringTableToXmString`() simply walks along the array of items within the *table*, concatenating a copy of each item to the result, along with a copy of the *break_component*. The *break_component* can be NULL, although a component of type XmSTRING_COMPONENT_TAB or XmSTRING_COMPONENT_SEPARATOR is a suitable choice. The function returns allocated storage, and it is the responsibility of the programmer to reclaim the space by calling `XmStringFree`() at a suitable point.

Example

The following code illustrates a basic call to `XmStringTableToXm-String`():

```
extern XmString     table table ;
extern int          table_count ;
XmString            xms;
XmString            break_component;

/* create a break component */
```

```
break_component = XmStringComponentCreate
                                    (XmSTRING_COMPONENT_SEPARATOR, 0,
                                    NULL)[1];

/* convert an (unspecified) compound string table */
xms = XmStringTableToXmString (table, table_count, break_component);

/* use the compound string */
...

/* free the allocated space */
XmStringFree (xms);
```

See Also

`XmStringConcat`(1), `XmStringCopy`(1), `XmStringFree`(1), `XmStringToXmStringTable`(1).

1.Erroneously given as XmComponentCreate() in 2nd edition. XmStringSeparatorCreate() would do here equally well.

Name

XmStringTableUnparse – convert a compound string table to an array of strings.

Synopsis

```
XtPointer *XmStringTableUnparse (  XmStringTable   table,
                                   Cardinal        count,
                                   XmStringTag     tag,
                                   XmTextType      tag_type,
                                   XmTextType      output_type,
                                   XmParseTable    parse_table,
                                   Cardinal        parse_count,
                                   XmParseModel    parse_model)
```

Inputs

table	Specifies the compound string table to unparse.
count	Specifies the number of compound strings in table.
tag	Specifies which text segments to unparse.
tag_type	Specifies the type of tag.
output_type	Specifies the type of conversion required.
parse_table	Specifies a parse table to control the conversion.
parse_count	Specifies the number of parse mappings in parse_table.
parse_model	Specifies how non-text components are converted.

Returns

An allocated string array containing the unparsed contents of the compound strings.

Availability

Motif 2.0 and later.

Description

`XmStringTableUnparse`() is a convenience function which unparses an array of compound strings. The XmStringTable *table* is converted into a string array, whose contents is determined by *output_type*, which can be XmCHARSET_TEXT, XmWIDECHAR_TEXT, or XmMULTIBYTE_TEXT. Only those text components within the *table* which match *tag* are converted: NULL converts all text components. An XmParseTable can be supplied which acts as a filter: each parse mapping in the table contains a pattern which must match a text component of the compound string if that component is to be converted.

parse_model determines how non-text components of string are handled when matching against *parse_table*. If the value is XmOUTPUT_ALL, all components are taken into account in the conversion process. If the value is

XmOUTPUT_BETWEEN, any non-text components which are not between two text components are ignored. If the value is XmOUTPUT_BEGINNING, any non-text components which do not precede a text component are ignored. Similarly, the value XmOUTPUT_END specifies that any non-text components which do not follow a text component are ignored. XmOUTPUT_BOTH is a combination of XmOUTPUT_BEGINNING and XmOUTPUT_END. The number of items within the returned string array is identical to the supplied number of compound strings. Each converted string occupies the same index in the returned array as the index of the source compound string in table.

Usage

`XmStringTableUnparse`() is the logical inverse of the routine `XmStringTableParseStringArray`(). It simply invokes `XmStringUnparse`() on each of the elements in the supplied *table*. The function returns allocated storage, both for the returned array, and for each element within the array. It is the responsibility of the programmer to reclaim the storage at an appropriate point by calling `XtFree`() on each element in the array and upon the array itself.

Structures

An XmParseModel has the following possible values:

XmOUTPUT_ALL
XmOUTPUT_BEGINNING
XmOUTPUT_BETWEEN
XmOUTPUT_BOTH
XmOUTPUT_END

See Also

`XmStringFree`(1), `XmStringParseText`(1), `XmStringUnparse`(1), `XmParseMapping`(2).

Name

XmStringToXmStringTable – convert a compound string to a compound string table.

Synopsis

Cardinal XmStringToXmStringTable (XmString *string*, XmString *break_comp*, XmStringTable **table*)

Inputs

string	Specifies a compound string.
break_comp	Specifies a component which indicates where to split string into an individual table element.

Outputs

table	Returns the converted compound string table.

Returns

The number of elements in the converted compound string table.

Availability

Motif 2.0 and later.

Description

`XmStringToXmStringTable`() is a convenience function which converts an XmString *string* into an array of compound strings. *break_comp* is a component which is used to determine how to split the string into individual items: components in string are considered to form contiguous sequences delimited by *break_comp*, each sequence forming a separate entry in the converted string table. Any component from *string* which matches the delimiting *break_comp* is not copied into the converted table. If *break_comp* is NULL, the returned compound string table contains a single entry: a copy of the original string.

Usage

`XmStringToXmStringTable`() is the inverse function to `XmStringTableToXmString`(). A *break_comp* component of type XmSTRING_COMPONENT_TAB or XmSTRING_COMPONENT_SEPARATOR is the most useful choice. The function returns allocated storage, and it is the responsibility of the programmer to reclaim the space by calling `XmStringFree`() on each element in the table, and then `XtFree`() on the table itself.

Example

The following code illustrates a basic call to `XmStringToXmStringTable`():

```
extern XmString     xms;
```

```
Cardinal        count;
XmStringTable   table;
XmString        break_component;
int             i;

/* create a break component */
break_component = XmStringComponentCreate
                                        (XmSTRING_COMPONENT_SEPARATOR, 0,
                                        NULL)[1];

/* convert an (unspecified) compound string */
count = XmStringToXmStringTable (xms, break_component, &table);

/* use the table */
...

/* free the allocated space */
for (i = 0; i < count; i++) {
    XmStringFree (table[i]);
}

XtFree ((char *) table);
```

See Also

`XmStringFree`(1), `XmStringTableToXmString`(1).

1.Erroneously given as XmComponentCreate() in 2nd edition. XmStringSeparatorCreate() could be used equally well.

Name

XmStringUnparse – convert a compound string into a string.

Synopsis

```
XtPointer XmStringUnparse (  XmString       string,
                             XmStringTag    tag,
                             XmTextType     tag_type,
                             XmTextType     output_type,
                             XmParseTable   parse_table,
                             Cardinal       parse_count,
                             XmParseModel   parse_model)
```

Inputs

string	Specifies the compound string to unparse.
tag	Specifies which text segments of string to unparse.
tag_type	Specifies the type of tag.
output_type	Specifies the type of conversion required.
parse_table	Specifies a parse table to control the conversion.
parse_count	Specifies the number of parse mappings in parse_table.
parse_model	Specifies how non-text components are converted.

Returns

An allocated string containing the unparsed contents of a compound string.

Availability

Motif 2.0 and later.

Description

`XmStringUnparse`() is a convenience function which unparses a compound string. The XmString *string* is converted into a string, whose contents is determined by *output_type*, which can be XmCHARSET_TEXT, XmWIDECHAR_TEXT, or XmMULTIBYTE_TEXT. Only those text components within the string which match *tag* are converted: NULL converts all text components. An XmParseTable can be supplied which acts as a filter: each parse mapping in the table contains a pattern which must match a text component of the compound string if that component is to be converted. *parse_model* determines how non-text components of string are handled when matching against *parse_table*. If the value is XmOUTPUT_ALL, all non-text components are used in the conversion process. If the value is XmOUTPUT_BETWEEN, any non-text components which fall between text components are used. If the value is XmOUTPUT_BEGINNING, non-text components which precede a text component are utilized. Similarly, XmOUTPUT_END uses non-text components which follow a text component. XmOUTPUT_BOTH is a combination of XmOUTPUT_BEGINNING and XmOUTPUT_END.

Usage

XmStringUnparse() is the logical inverse of the routine XmStringParse-Text(). The function returns allocated storage, and it is the responsibility of the programmer to reclaim the storage by calling XtFree() at an appropriate point.

Structures

An XmParseModel has the following possible values:

XmOUTPUT_ALL
XmOUTPUT_BEGINNING
XmOUTPUT_BETWEEN
XmOUTPUT_BOTH
XmOUTPUT_END

Example

The following specimen code outlines a basic call to XmStringUnparse(), which converts separator and tab components into the output:

```
XmParseTable    parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
XmString        tmp;
XmString        output;
char            *string;
Arg             av[4];
Cardinal        ac;

/* map tab component to \t */
tmp = XmStringComponentCreate (XmSTRING_COMPONENT_TAB, 0,
NULL);
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\t");         ac++;
parse_table[0] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);

/* map separator component to \n */
tmp = XmStringSeparatorCreate();
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\n");         ac++;
parse_table[1] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);
```

```
/* convert the (unspecified) compound string */
string = (char *) XmStringUnparse (xms, NULL, XmCHARSET_TEXT,
                                   XmCHARSET_TEXT, parse_table, 2,
                                   XmOUTPUT_ALL);

/* use the converted string */
....

/* free the allocated space */
XtFree (string);

/* Free the parse table: this also frees the parse mappings */
XmParseTableFree (parse_table, 2);
```

See Also

`XmStringFree`(1), `XmStringParseText`(1), `XmStringTableUnparse`(1), `XmParseMapping`(2).

Name

XmStringWidth – get the width of the longest line of text in a compound string.

Synopsis

Dimension XmStringWidth (XmFontList *fontlist*, XmString *string*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Returns

The width of the compound string.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringWidth`() returns the width, in pixels, of the longest line of text in the specified compound *string*. Lines in the compound string are delimited by separator components. If *string* is created with `XmStringCreateSimple`(), then fontlist must begin with the font from the character set of the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringWidth`() provides information that is useful if you need to render a compound string. Motif widgets render compound strings automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringBaseline`(1), `XmStringExtent`(1), `XmStringHeight`(1).

Name

XmTabCreate – create a tab stop.

Synopsis

```
XmTab XmTabCreate (  float          value,
                     unsigned char  units,
                     XmOffsetModel  offset_model,
                     unsigned char  alignment,
                     char           *decimal)
```

Inputs

value Specifies the value to be used in calculating the location of a tab stop.

units Specifies the units in which value is expressed.

offset_model Specifies whether the tab is at an absolute position, or relative to the previous tab.

alignment Specifies how text should be aligned to this tab stop.

decimal Specifies the multibyte character in the current locale to be used as a decimal point.

Returns

An XmTab object.

Availability

Motif 2.0 and later.

Description

`XmTabCreate`() creates a tab stop at the position specified by *value*, which is expressed in terms of *units*. If value is less than 0 (zero), a warning is displayed, and *value* is reset to 0.0. The *offset_model* determines whether the tab position is an absolute location (XmABSOLUTE) calculated from the start of any rendering, or relative to the previous tab stop (XmRELATIVE). *alignment* specifies how text is aligned with respect to the tab location: at present only XmALIGNMENT_BEGINNING is supported. *decimal* is a multibyte character which describes the decimal point character in the current locale.

Usage

A tab stop can be created, and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget. The *decimal* parameter is currently unused.

XmTabCreate() allocates storage for the object which it returns. It is the responsibility of the programmer to free the memory at an appropriate point by a call to XmTabFree().

Structures

The XmOffsetModel type has the following possible values:

XmABSOLUTE	XmRELATIVE

Valid values for the units parameter are:

XmPIXELS	
Xm100TH_MILLIMETERS	XmMILLIMETERS
Xm1000TH_INCHES	XmINCHES
Xm100TH_FONT_UNITS	XmFONT_UNITS
Xm100TH_POINTS	XmPOINTS
XmCENTIMETERS	

Example

The following code creates a multi-column arrangement of compound strings; it creates a set of XmTab objects, which are then inserted into an XmRendition object. The XmRendition object is added to a render table:

```
extern Widget widget;
int            i;
Arg            argv[3];
XmTab          tabs[MAX_COLUMNS];
XmTabList      tab_list = (XmTabList) 0;
XmRendition    rendition;
XmRenderTable  render_table;

/* Create the XmTab objects */
for (i = 0 ; i < MAX_COLUMNS ; i++) {
   tabs[i] = XmTabCreate ((float) 1.5,
                          XmINCHES,
                          ((i == 0) ? XmABSOLUTE :
                          XmRELATIVE),
                          XmALIGNMENT_BEGINNING,
                          ".");
}

/* Add them to an XmTabList */
tab_list = XmTabListInsertTabs (NULL, tabs,
MAX_COLUMNS, 0);

/* Put the XmTabList into an XmRendition object */
```

```
i = 0;
XtSetArg (argv[i], XmNtabList, tab_list); i++;
XtSetArg (argv[i], XmNfontName, “fixed”); i++;
XtSetArg (argv[i], XmNfontType, XmFONT_IS_FONT);
i++;
rendition = XmRenditionCreate (widget,NULL,argv,
i);

/* Add the XmRendition object into an XmRenderTable
*/
render_table = XmRenderTableAddRenditions (NULL,
                                           &rendi-
                                           tion,
                                           1,
                                           XmMERGE
                                           _NEW);

/* Apply to the widget */
XtVaSetValues (widget, XmNrenderTable,
render_table, NULL);
...

/* Free Up - render table applied to widget takes a
reference
** counted copy of everything
*/
for (i = 0 ; i < MAX_COLUMNS ; i++) {
   XmTabFree (tabs[i]);
}
XmTabListFree (tab_list);
XmRenditionFree (rendition);
XmRenderTableFree (render_table);
```

See Also

XmTabFree(1), XmTabGetValues(1), XmTabListInsertTabs(1), XmTabSetValue(1), XmRenditionCreate(1), XmRenderTableAddRenditions(1), XmRendition(2).

Name

XmTabFree – free the memory used by an XmTab object.

Synopsis

void XmTabFree (XmTab *tab*)

Inputs

tab Specifies an XmTab object.

Availability

Motif 2.0 and later.

Description

`XmTabFree`() reclaims the memory associated with tab, previously allocated by a call to `XmTabCreate`().

Usage

A tab stop can be created using `XmTabCreate`(), and inserted into an XmTab-List through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

See Also

`XmTabCreate`(1), `XmRendition`(2).

Name

XmTabGetValues – fetch the value of an XmTab object.

Synopsis

```
float XmTabGetValues (  XmTab           tab,
                        unsigned char   *units,
                        XmOffsetModel   *offset_model,
                        unsigned char   *alignment,
                        char            **decimal)
```

Inputs

tab	Specifies an XmTab object.

Outputs

units	Returns the units in which the tab stop is calculated.
offset_model	Returns the offset model.
alignment	Returns the text alignment with respect to the tab stop.
decimal	Returns the multibyte character used to represent the decimal point.

Returns

The distance, expressed in units, which the tab is offset.

Availability

Motif 2.0 and later.

Description

`XmTabGetValues`() retrieves the data associated with a tab stop object. The function returns the position value of a tab stop, which is expressed in terms of units. The *offset_model* determines whether the *tab* position is an absolute location (XmABSOLUTE) calculated from the start of any rendering, or relative to the previous tab stop (XmRELATIVE). *alignment* specifies how text is aligned with respect to the tab location: at present only XmALIGNMENT_BEGINNING is supported. *decimal* is a multibyte character which describes the decimal point character in the current locale.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget. The decimal value is currently unused.

Structures

The XmOffsetModel type has the following possible values:

XmABSOLUTE	XmRELATIVE

Valid values for units are:

XmPIXELS	
Xm100TH_MILLIMETERS	XmMILLIMETERS
Xm1000TH_INCHES	XmINCHES
Xm100TH_FONT_UNITS	XmFONT_UNITS
Xm100TH_POINTS	XmPOINTS
XmCENTIMETERS	

See Also

`XmTabCreate`(1), `XmTabSetValue`(1), `XmRendition`(2).

Name

XmTabListCopy – copy an XmTabList object.

Synopsis

XmTabList XmTabListCopy (XmTabList *tab_list*, int *offset*, Cardinal *count*)[1]

Inputs

tab_list	Specifies the tab list to copy.
offset	Specifies an index into the tab list from which to start copying.
count	Specifies the number of tab stops to copy.

Returns

A new tab list containing tabs copied from *tab_list*.

Availability

Motif 2.0 and later.

Description

`XmTabListCopy`() is a convenience function which creates a new XmTabList by copying portions of an existing *tab_list*. If *tab_list* is NULL, the function simply returns NULL. Otherwise, a new tab list is allocated, and selected tabs are copied, depending on the *offset* and *count* parameters. *count* specifies the number of tabs to copy, and *offset* determines where in the original list to start. If *offset* is zero, tabs are copied from the beginning of tab_list. If *offset* is negative, tabs are copied in reverse order from the end of *tab_list*. If *count* is zero, all tabs from *offset* to the end of *tab_list* are copied.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

`XmTabListCopy`() allocates storage for the object which it returns. It is the responsibility of the programmer to free the memory at an appropriate point by a call to `XmTabListFree`().

See Also

`XmTabCreate`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

1.Erroneously given as XmTabListTabCopy() in 2nd edition.

Name

XmTabListFree – free the memory used by a tab list.

Synopsis

void XmTabListFree (XmTabList *tab_list*)

Inputs

tab_list Specified the tab list to free.

Availability

Motif 2.0 and later.

Description

`XmTabListFree`() reclaims the space used by an XmTabList object, *tab_list*.

Usage

In common with other objects in Motif 2.0 and later, the tab (XmTab) and tab list (XmTabList) are dynamically allocated data structures which must be freed when no longer required. For example, `XmStringTableProposeTablist`() dynamically creates a tab list for rendering a multi-column compound string table which must be subsequently deallocated.

It is important to call `XmTabListFree`() instead of `XtFree`() because `XmTabListFree`() also reclaims storage for each of the elements in the tab list.

See Also

`XmStringTableProposeTablist`(1), `XmTabCreate`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListGetTab – retrieve a tab from a tab list

Synopsis

XmTab XmTabListGetTab (XmTabList *tab_list*, Cardinal *position*)

Inputs

tab_list	Specifies a tab list.
position	Specifies the position of the required tab within the tab_list.

Returns

A copy of the required tab.

Availability

Motif 2.0 and later.

Description

`XmTabListGetTab`() returns a copy of a tab from the XmTabList specified by *tab_list*. *position* determines where in the list of tabs the required tab is copied from. The first tab within *tab_list* is at *position* zero, the second tab at *position* 1, and so on. If *position* is not less than the number of tabs within the list, `XmTab-ListGetTab`() returns NULL.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

`XmTabListGetTab`() returns a copy of the tab within the original tab list, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabFree`().

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListInsertTabs – insert tabs into a tab list.

Synopsis

XmTabList XmTabListInsertTabs (XmTabList *tab_list*, XmTab **tabs*, Cardinal *tab_count*, int *position*)

Inputs

tab_list	Specifies the tab list into which tabs are added.
tabs	Specifies an array of tabs to add.
tab_count	Specifies the number of tabs within the tabs array.
position	Specifies an index into tab_list at which to insert the tabs.

Returns

A newly allocated tab list.

Availability

Motif 2.0 and later.

Description

`XmTabListInsertTabs`() creates a new tab list by merging a set of *tabs* into a *tab_list* at a given *position*. If *tabs* is NULL, or *tab_count* is zero, the function returns the original *tab_list*. If *position* is zero, the new tabs are inserted at the head of the new tab list, if *position* is 1, they are inserted after the first tab, and so forth. If *position* is greater than the number of tabs in *tab_list*, the new tabs are appended. If *position* is negative, the new tabs are inserted in reverse order at the end of the original list, so that the first new tab becomes the last tab in the new list.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

`XmTabListInsertTabs`() returns allocated storage, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabListFree`().

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListRemoveTabs – copy a tab list, excluding specified tabs.

Synopsis

```
XmTabList XmTabListRemoveTabs ( XmTabList   old_list,
                                Cardinal    *position_list,
                                Cardinal    position_count)
```

Inputs

old_list	Specifies the tab list to copy.
position_list	Specifies an array of tab positions to exclude.
position_count	Specifies the length of position_list.

Returns

A copy of *old_list*, with the specified positions excluded.

Availability

Motif 2.0 and later.

Description

`XmTabListRemoveTabs`() removes tabs from a *old_list* by allocating a new tab list which contains all the tabs of the original list except for those at specific positions within a *position_list*. The first tab within a tab list is at position zero, the second tab at position 1, and so on. If *old_list* is NULL, or if *position_list* is NULL, or if *position_count* is zero, the function returns *old_list* unmodified. Otherwise *old_list* is freed, as are any excluded tab elements, before the newly allocated tab list is returned.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

When the returned tab list differs from the original old_list parameter, `XmTab-ListRemoveTabs`() returns allocated storage, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabList-Free`().

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListReplacePositions – copy a tab list, replacing tabs at specified positions.

Synopsis

```
XmTabList XmTabListReplacePositions ( XmTabList    old_list,
                                      Cardinal     *position_list,
                                      XmTab        *tabs,
                                      Cardinal     tab_count)
```

Inputs

old_list	Specifies the tab list to modify.
position_list	Specifies an array of positions at which to replace the tabs.
tabs	Specifies the tabs which replace those in old_list.
tab_count	Specifies the number of tabs and positions.

Returns

A new tab list with tabs replaced at specified positions.

Availability

Motif 2.0 and later.

Description

`XmTabListReplacePositions`() creates a newly allocated tab list which contains all the original tabs in *old_list*, except that at any position contained within *position_list*, the corresponding tab from tabs is used instead of the original. That is, at the first position specified within *position_list*, the original tab is replaced by the first tab within *tabs*. The first tab within a tab list is at position zero, the second tab at position 1, and so on. If *old_list* is NULL, or if *tabs* is NULL, or if *position_list* is NULL, or if *tab_count* is zero, the function returns *old_list* unmodified. Otherwise *old_list* is freed, as are any replaced tab elements, before the newly allocated tab list is returned.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

When the returned tab list differs from the original old_list parameter, `XmTabListReplacePositions`() returns allocated storage, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabListFree`()

See Also

XmTabCreate(1), XmTabFree(1), XmTabListFree(1), XmTabListInsertTabs(1), XmTabListRemoveTabs(1), XmRendition(2).

Name

XmTabListTabCount – count the number of tabs in a tab list.

Synopsis

Cardinal XmTabListTabCount (XmTabList *tab_list*)

Inputs

tab_list The tab list to count.

Returns

The number of tab stops in tab_list.

Availability

Motif 2.0 and later.

Description

`XmTabListTabCount`()[1] is a convenience function which counts the number of XmTab objects contained within *tab_list*. If *tab_list* is NULL, the function returns zero.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmTabListRemoveTabs`(1), `XmRendition`*(s2.

1.Erroneously given as XmTabListCount() in 2nd edition.

Name

XmTabSetValue – set the value of a tab stop.

Synopsis

void XmTabSetValue (XmTab *tab*, float *value*)

Inputs

tab	Specifies the tab to modify.
value	Specifies the new value for the tab stop.

Availability

Motif 2.0 and later.

Description

`XmTabSetValue`() sets the value associated with an XmTab object. The *value* is a floating point quantity which either represents an absolute distance from the start of the current rendition, or a value relative to the previous tab stop. The offset model specified when the tab is created determines whether value is relative or absolute. If *value* is less than 0 (zero), a warning message is displayed and the function returns without modifying the tab.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

See Also

`XmTabCreate`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabStackGetSelectedTab –return the widgetID of the currently selected tab.

Synopsis

#include <Xm/TabStack.h>

Widget XmTabStackGetSelectedTab (Widget *widget*)

Inputs

widget	Specifies widget ID of the TabStack.

Outputs

value_return	Returns the widget ID of the currently selected tab.

Description

`XmTabStackGetSelectedTab`() retrieves the widget ID of the currently selected tab.

Usage

`XmTabStackGetSelectedTab`() is a convenience routine that retrieves the widget ID of the currently selected tab.

See Also

`XmTabStack`(2).

Name

XmTabStackIndexToWidget – returns the widget ID of the tab in the TabStack.

Synopsis

#include <Xm/TabStack.h>

Widget XmTabStackIndexToWidget (Widget *widget*, int *tab*)

Inputs

widget	Specifies the widget ID of the TabStack.
tab	Specifies the tab index retrieve.

Outputs

value_return	Returns the widget ID of the tab in the TabStack.

Description

`XmTabStackIndexToWidget`() returns the widget ID of the tab in the TabStack.

Usage

`XmTabStackIndexToWidget`() is a convenience routine that returns the widget ID of the tab in the TabStack

See Also

`XmTabStack`(2).

Name

XmTabStackSelectTab – set the currently displayed child of the TabStack.

Synopsis

#include <Xm/TabStack.h>

void XmTabStackSelectTab (Widget *tab*, Boolean *notify*)

Inputs

tab	Specifies the child of the TabStack to be selected.
notify	Specifies whether XmNtabSelectedCallback will be called as usual or not.

Description

`XmTabStackSelectTab`() sets the currently displayed child of the TabStack.

Usage

`XmTabStackSelectTab`() is a convenience routine that sets the currently displayed child of the TabStack. When *notify* is set to True, XmNtabSelectedCallback will be called as usual in response to the change of state. Set False to suppress this callback.

See Also

`XmTabStack`(2).

Name

XmTabStackXYToWidget – convert a pixel coordinate in the TabStack's window to the widget of the tab.

Synopsis

#include <Xm/TabStack.h>

Widget XmTabStackXYToWidget (Widget *widget*, int *x*, int *y*)

Inputs

widget	Specifies widget ID of the TabStack.
x, y	Specifies the pixel coordinates to convert.

Description

`XmTabStackXYToWidget`() converts a pixel coordinate (in the TabStack's window) to the widget ID of the tab occupying that space.

Usage

`XmTabStackXYToWidget`() is a convenience routine that converts an x/y pixel coordinate (in the TabStack's window) to the widget ID of the tab occupying that space.

See Also

`XmTabStack`(2).

Name

XmTargetsAreCompatible – determine whether or not the target types of a drag source and a drop site match.

Synopsis

```
#include <Xm/DragDrop.h>

Boolean XmTargetsAreCompatible (  Display     *display,
                                  Atom        *export_targets,
                                  Cardinal    num_export_targets,
                                  Atom        *import_targets,
                                  Cardinal    num_import_targets)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
export_targets	Specifies the list of target atoms to which the drag source can convert the data.
num_export_targets	Specifies the number of items in export_targets.
import_targets	Specifies the list of target atoms that are accepted by the drop site.
num_import_targets	Specifies the number of items in import_targets.

Returns

True if there is a compatible target or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmTargetsAreCompatible`() determines whether or not the import targets of a drop site match any of the export targets of a drag source. The routine returns True if the two objects have at least one target in common; otherwise it returns False.

Usage

Motif 1.2 and later supports the drag and drop model of selection actions. In a widget that acts as a drag source, a user can make a selection and then drag the selection, using BTransfer, to other widgets that are registered as drop sites. These drop sites can be in the same application or another application. In order for a drag and drop operation to succeed, the drag source and the drop site must both be able to handle data in the same format. `XmTargetsAreCompatible`() provides a way for an application to check if a drag source and a drop site support compatible formats.

See Also

XmDragContext(1), XmDropSite(1).

Name

XmTextClearSelection, XmTextFieldClearSelection – clear the primary selection.

Synopsis

```
#include <Xm/Text.h>

void XmTextClearSelection (Widget widget, Time time)

#include <Xm/TextF.h>

void XmTextFieldClearSelection (Widget widget, Time time)
```

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Description

`XmTextClearSelection`() and `XmTextFieldClearSelection`() clear the primary selection in the specified *widget*. `XmTextClearSelection`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldClearSelection`() only works for a TextField widget. For each routine, *time* specifies the server time of the event that caused the request to clear the selection.

Usage

`XmTextClearSelection`() and `XmTextFieldClearSelection`() provide a convenient way to deselect the text selection in a Text or TextField widget. If no text is selected, the routines do nothing. Any text that is stored in the clipboard or selection properties remains; the routines affect the selected text in the widget only. If you are calling one of these routines from a callback routine, you probably want to use the time field from the event pointer in the callback structure as the value of the *time* parameter. You can also use the value CurrentTime, but there is no guarantee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, and **Clear**) shows the use of XmTextClearSelection()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                 num = (int) client_data;
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *) call_data;
```

```
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);
        break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbut-
        ton.time);break;
        case 3:   result = XmTextPaste (text_w);break
        case 4:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

XmTextCopy(1), XmTextCopyLink(1), XmTextCut(1), XmTextGetSelection(1), XmTextGetSelectionPosition(1), XmTextGetSelectionWcs(1), XmTextSetSelection(1), XmText(2), XmTextField(2).

Name

XmTextCopy, XmTextFieldCopy, XmDataFieldCopy – copy the primary selection to the clipboard.

Synopsis

#include <Xm/Text.h>

Boolean XmTextCopy (Widget *widget*, Time *time*)

#include <Xm/TextF.h>

Boolean XmTextFieldCopy (Widget *widget*, Time *time*)

#include <Xm/DataF.h>

Boolean XmDataFieldCopy (Widget *widget*, Time *time*)

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Returns

True on success or False otherwise.

Description

`XmTextCopy`(), `XmTextFieldCopy`(), and `XmDataFieldCopy`() copy the primary selection in the specified *widget* to the clipboard. `XmTextCopy`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldCopy`() only works for a TextField widget and `XmDataFieldCopy`() only works for a DataField widget. For each routine, *time* specifies the server time of the event that caused the request to copy the selection. Both routines return True if successful. If the primary selection is NULL, if it is not owned by the specified widget, or if the function cannot obtain ownership of the clipboard selection, the routines return False.

In Motif 2.0 and later, `XmTextCopy`() interfaces with the Uniform Transfer Model by indirectly invoking the XmNconvertCallback procedures of the widget.

Usage

`XmTextCopy`(), `XmTextFieldCopy`(), and `XmDataFieldCopy`() copy the text that is selected in a Text, TextField, or DataField widget and place it on the clipboard. If you are calling one of these routines from a callback routine, you probably want to use the *time* field from the event pointer in the callback structure as the value of the time parameter. You can also use the value CurrentTime, but there is no guarantee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of `XmTextCopy`()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                     num = (int) client_data;
    XmAnyCallbackStruct*cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);
        break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbut-
        ton.time);break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

`XmTextClearSelection`(1), `XmTextCopyLink`(1), `XmTextCut`(1), `XmTextGetSelection`(1), `XmTextGetSelectionWcs`(1), `XmTextPaste`(1), `XmTextPasteLink`(1), `XmTextRemove`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2), `XmDataField`(2).

Name

XmTextCopyLink, XmTextFieldCopyLink – copy the primary selection to the clipboard.

Synopsis

#include <Xm/Text.h>

Boolean XmTextCopyLink (Widget *widget*, Time *time*)

#include <Xm/TextF.h>

Boolean XmTextFieldCopyLink (Widget *widget*, Time *time*)

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Returns

True on success or False otherwise.

Availability

Motif 2.0 and later.

Description

`XmTextCopyLink`() and `XmTextFieldCopyLink`() copy a link to the primary selection in the specified *widget* to the clipboard. `XmTextCopyLink`() works when widget is a Text widget or a TextField widget, while `XmTextFieldCopyLink`() only works for a TextField widget. For each routine, *time* specifies the server time of the event that caused the request to copy the selection. Both routines return True if successful. If the primary selection is NULL, if it is not owned by the specified widget, or if the function cannot obtain ownership of the clipboard selection, the routines return False.

`XmTextCopyLink`() and `XmTextFieldCopyLink`() interface with the Uniform Transfer Model by indirectly invoking XmNconvertCallback procedures for the widget. The Text widget itself does not copy links: convert procedures which the programmer provides are responsible for copying the link to the clipboard.

Usage

`XmTextCopyLink`() and `XmTextFieldCopyLink`() copy links to the text that is selected in a Text or TextField widget and place it on the clipboard. If you are calling one of these routines from a callback routine, you probably want to use the time field from the event pointer in the callback structure as the value of the *time* parameter. You can also use the value CurrentTime, but there is no guar-

antee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of XmTextCopyLink()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
    {
    int                     num = (int) client_data;
    XmAnyCallbackStruct     *cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);
        break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbut-
        ton.time);break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

XmTextClearSelection(1), XmTextCopy(1), XmTextCut(1), XmTextGetSelection(1), XmTextGetSelectionWcs(1), XmTextPaste(1), XmTextPasteLink(1), XmTextRemove(1), XmTextSetSelection(1), XmText(2), XmTextField(2).

Name

XmTextCut, XmTextFieldCut, XmDataFieldCut – copy the primary selection to the clipboard and remove the selected text.

Synopsis

#include <Xm/Text.h>

Boolean XmTextCut (Widget *widget*, Time *time*)

#include <Xm/TextF.h>

Boolean XmTextFieldCut (Widget *widget*, Time *time*)

#include <Xm/DataF.h>

Boolean XmDataFieldCut (Widget *widget*, Time *time*)

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Returns

True on success or False otherwise.

Description

`XmTextCut`(), `XmTextFieldCut`(), and `XmDataFieldCut`() copy the primary selection in the specified *widget* to the clipboard and then delete the primary selection. `XmTextCut`() works when widget is a Text widget or a TextField widget, while `XmTextFieldCut`() only works for a TextField widget and `XmDataFieldCut`() only works for a DataField widget. For each routine, *time* specifies the server time of the event that caused the request to cut the selection. Both routines return True if successful. If the widget is not editable, if the primary selection is NULL or if it is not owned by the specified widget, or if the function cannot obtain ownership of the clipboard selection, the routines return False.

`XmTextCut`(), `XmTextFieldCut`(), and `XmDataFieldCut`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

In Motif 2.0 and later, `XmTextCut`() interfaces with the Uniform Transfer Model by indirectly invoking the XmNconvertCallback procedures of the widget, firstly to transfer the selection to the clipboard, and secondly to delete the selection.

Usage

XmTextCut(), XmTextFieldCut() , and XmDataField Cut() copy the text that is selected in a Text, TextField, or DataField widget, place it on the clipboard, and then delete the selected text. If you are calling one of these routines from a callback routine, you probably want to use the time field from the event pointer in the callback structure as the value of the *time* parameter. You can also use the value CurrentTime, but there is no guarantee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, and **PasteLink**, **Clear**) shows the use of XmTextCut()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                     num = (int) client_data;
    XmAnyCallbackStruct*cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);
        break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbut-
        ton.time);break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

XmTextClearSelection(1), XmTextCopy(1), XmTextCopyLink(1), XmTextGetSelection(1), XmTextGetSelectionWcs(1), XmTextPaste(1), XmTextPasteLink(1), XmTextRemove(1),

XmTextSetSelection(1), XmText(2), XmTextField(2), XmDataField(2).

Name

XmTextDisableRedisplay – prevent visual update of a Text widget.

Synopsis

#include <Xm/Text.h>

void XmTextDisableRedisplay (Widget *widget*)

Inputs

widget Specifies the Text widget.

Availability

Motif 1.2 and later.

Description

`XmTextDisableRedisplay`() temporarily inhibits visual update of the specified Text *widget*. Even if the visual attributes of the widget have been modified, the appearance remains unchanged until `XmTextEnableRedisplay`() is called.

Usage

`XmTextDisableRedisplay`() and `XmTextEnableRedisplay`() allow an application to make multiple changes to a Text widget without immediate visual updates. When multiple changes are made with redisplay enabled, visual flashing often occurs. These routines eliminate this problem.

See Also

`XmTextEnableRedisplay`(1), `XmUpdateDisplay`(1), `XmText`(2).

Name

XmTextEnableRedisplay – allow visual update of a Text widget.

Synopsis

#include <Xm/Text.h>

void XmTextEnableRedisplay (Widget *widget*)

Inputs

widget Specifies the Text widget.

Availability

Motif 1.2 and later.

Description

`XmTextEnableRedisplay`() allows the specified Text *widget* to update its visual appearance. This routine is used in conjunction with `XmTextDisable-Redisplay`(), which prevents visual update of the Text widget. When `XmTex-tEnableRedisplay`() is called, the widget modifies its visuals to reflect all of the changes since the last call to `XmTextDisableRedisplay`(). All future changes that affect the visual appearance are displayed immediately.

Usage

`XmTextDisableRedisplay`() and `XmTextEnableRedisplay`() allow an application to make multiple changes to a Text widget without immediate visual updates. When multiple changes are made with redisplay enabled, visual flashing often occurs. These routines eliminate this problem.

See Also

`XmTextDisableRedisplay`(1), `XmUpdateDisplay`(1), `XmText\*`(s2.

Name

XmTextFindString – find the beginning position of a text string.

Synopsis

```
#include <Xm/Xm.h>

Boolean XmTextFindString (  Widget          widget,
                            XmTextPosition  start,
                            char            *string,
                            XmTextDirection direction,
                            XmTextPosition  *position)
```

Inputs

widget	Specifies the Text widget.
start	Specifies the position from which the search begins.
string	Specifies the string for which to search.
direction	Specifies the direction of the search. Pass either XmTEXT_FORWARD or XmTEXT_BACKWARD.

Outputs

position	Returns the position where the search string starts.

Returns

True if the string is found or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmTextFindString`() finds the beginning position of the specified *string* in the Text widget. Depending on the value of *direction*, the routine searches forward or backward from the specified *start* position for the first occurrence of *string*. If `XmTextFindString`() finds a match, it returns True and *position* specifies the position of the first character of the string as the number of characters from the beginning of the text, where the first character position is 0 (zero). If a match is not found, the routine returns False and the value of *position* is undefined.

Usage

`XmTextFindString`() is a convenience routine that searches the text in a Text widget for a particular *string*. Without the routine, the search must be performed using the standard string manipulation routines.

Example

The following routine shows the use of XmTextFindString() to locate a string in a text editing window. The search string is specified by the user in a single-line Text widget:

```
Widget text_w, search_w;

void search_text (void)
{
    char            *search_pat = (char *) 0;
    XmTextPosition  pos, search_pos;
    Boolean         found = False;

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    /* find next occurrence from current position -- wrap if necessary */
    pos = XmTextGetCursorPosition (text_w);
    found = XmTextFindString (text_w, pos, search_pat,
    XmTEXT_FORWARD, &search_pos);

    if (!found)
        found = XmTextFindString (text_w, 0, search_pat,
        XmTEXT_FORWARD, &search_pos);

    if (found)
        XmTextSetInsertionPosition (text_w, search_pos);

    XtFree (search_pat);
}
```

See Also

XmTextFindStringWcs(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmText(2).

Name

XmTextFindStringWcs – find the beginning position of a wide-character string in a Text widget.

Synopsis

```
#include <Xm/Text.h>

Boolean XmTextFindStringWcs (   Widget          widget,
                                XmTextPosition  start,
                                wchar_t         *wcstring,
                                XmTextDirection direction,
                                XmTextPosition  *position)
```

Inputs

widget	Specifies the Text widget.
start	Specifies the position from which the search begins.
wcstring	Specifies the wide-character string for which to search.
direction	Specifies the direction of the search. Pass either XmTEXT_FORWARD or XmTEXT_BACKWARD.

Outputs

position	Returns the position where the search string starts.

Returns

True if the string is found or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmTextFindStringWcs`() finds the beginning position of the specified wide-character *wcstring* in the Text *widget*. Depending on the value of *direction*, the routine searches forward or backward from the specified *start* position for the first occurrence of *wcstring*. If `XmTextFindStringWcs`() finds a match, it returns True and *position* specifies the position of the first character of the string as the number of characters from the beginning of the text, where the first character position is 0 (zero). If a match is not found, the routine returns False and the value of *position* is undefined.

Usage

In Motif 1.2, the Text widget supports wide-character strings. `XmTextFind-StringWcs`() is a convenience routine that searches the text in a Text widget for a particular wide-character string. The routine converts the wide-character string into a multi-byte string and then performs the search. Without the routine, the search must be performed using the standard string manipulation routines.

See Also

XmTextFindString(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmText(2).

Name

XmTextGetBaseline, XmTextFieldGetBaseline – get the position of the baseline.

Synopsis

#include <Xm/Text.h>

int XmTextGetBaseline (Widget *widget*)

#include <Xm/TextF.h>

int XmTextFieldGetBaseline (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The baseline position.

Description

`XmTextGetBaseline`() returns the y coordinate of the baseline of the first line of text in the specified Text *widget*, while `XmTextFieldGetBaseline`() returns the y coordinate of the baseline for the text in the specified TextField *widget*. `XmTextGetBaseline`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetBaseline`() only works for a TextField widget. For each routine, the returned value is relative to the top of the *widget* and it accounts for the margin height, shadow thickness, highlight thickness, and font ascent of the first font in the font list.

Usage

`XmTextGetBaseline`() and `XmTextFieldGetBaseline`() provide information that is useful when you are laying out an application and trying to align different components.

See Also

`XmTextGetCenterline`(1), `XmWidgetGetBaselines`(1), `XmWidgetGetDisplayRect`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetCenterline – get the height of vertical text.

Synopsis

#include <Xm/Text.h>

int XmTextGetCenterline (Widget *widget*)

Inputs

widget Specifies the Text widget.

Returns

The center line x position.

Availability

Motif 2.1 and later.

Description

`XmTextGetCenterline`() calculates the x coordinate of the centerline in a Text widget containing vertical text. If the layout direction of the Text widget does not match XmTOP_TO_BOTTOM_RIGHT_TO_LEFT the function returns zero. Otherwise the procedure calculates the x position of the centerline relative to the left of the Text. The margin width, shadow thickness, and the width of the font are taken into consideration when performing the calculation.

Usage

`XmTextGetCenterline`() provides information that is useful when you are laying out an application and trying to align different components.

See Also

`XmTextGetBaseline`(1), `XmWidgetGetCenterlines`(1), `XmWidgetGetDisplayRect`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetCursorPosition, XmTextFieldGetCursorPosition – get the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetCursorPosition (Widget *widget*)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetCursorPosition (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The value of the XmNcursorPosition resource.

Description

`XmTextGetCursorPosition`() and `XmTextFieldGetCursorPosition`() return the value of the XmNcursorPosition resource for the specified *widget*. `XmTextGetCursorPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetCursorPosition`() only works for a TextField widget. For each routine, the value specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

Usage

`XmTextGetCursorPosition`() and `XmTextFieldGetCursorPosition`() are convenience routines that return the value of the XmNcursorPosition resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextGetInsertionPosition`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetEditable, XmTextFieldGetEditable – get the edit permission state.

Synopsis

#include <Xm/Text.h>

Boolean XmTextGetEditable (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldGetEditable (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The state of the XmNeditable resource.

Description

`XmTextGetEditable`() and `XmTextFieldGetEditable`() return the value of the XmNeditable resource for the specified Text or TextField *widget*. `XmTextGetEditable`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetEditable`() only works for a TextField widget.

Usage

By default, the XmNeditable resource is True, which means that a user can edit the text string. Setting the resource to False makes a text area read-only. `XmTextGetEditable`() and `XmTextFieldGetEditable`() are convenience routines that return the value of the XmNeditable resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextSetEditable`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetInsertionPosition, XmTextFieldGetInsertionPosition – get the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetInsertionPosition (Widget *widget*)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetInsertionPosition (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The value of the XmNcursorPosition resource.

Description

The functions, `XmTextGetInsertionPosition`() and `XmTextFieldGetInsertionPosition`(), return the value of the XmNcursorPosition resource for the specified *widget*. `XmTextGetInsertionPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetInsertionPosition`() only works for a TextField widget. For each routine, the value specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

Usage

The functions, `XmTextGetInsertionPosition`() and `XmTextFieldGetInsertionPosition`(), are convenience routines that return the value of the XmNcursorPosition resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetCursorPosition`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetLastPosition, XmTextFieldGetLastPosition – get the position of the last character of text.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetLastPosition (Widget *widget*)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetLastPosition (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The position of the last text character.

Description

`XmTextGetLastPosition`() and `XmTextFieldGetLastPosition`() return the position of the last character of text in the specified *widget*. `XmText-GetLastPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetLastPosition`() only works for a Text-Field widget. For each routine, the returned value specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero).

Usage

`XmTextGetLastPosition`() and `XmTextFieldGetLastPosition`() are convenience routines that return the number of characters of text in a Text or TextField widget.

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetTopCharacter`(1), `XmTextScroll`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetTopCharacter`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetMaxLength, XmTextFieldGetMaxLength – get the maximum possible length of a text string.

Synopsis

```
#include <Xm/Text.h>

int XmTextGetMaxLength (Widget widget)

#include <Xm/TextF.h>

int XmTextFieldGetMaxLength (Widget widget)
```

Inputs

widget Specifies the Text or TextField widget.

Returns

The value of the XmNmaxLength resource.

Description

`XmTextGetMaxLength`() and `XmTextFieldGetMaxLength`() return the value of the XmNmaxLength resource for the specified Text or TextField *widget*. `XmTextGetMaxLength`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetMaxLength`() only works for a TextField widget. For each routine, the returned value specifies the maximum allowable length of a text string that a user can enter from the keyboard.

Usage

`XmTextGetMaxLength`() and `XmTextFieldGetMaxLength`() are convenience routines that return the value of the XmNmaxLength resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextSetMaxLength`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSelection, XmTextFieldGetSelection, XmDataFieldGetSelection – get the value of the primary selection.

Synopsis

#include <Xm/Text.h>

char * XmTextGetSelection (Widget *widget*)

#include <Xm/TextF.h>

char * XmTextFieldGetSelection (Widget *widget*)

#include <Xm/DataF.h>

char *XmDataFieldGetSelection (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

A string containing the primary selection.

Description

`XmTextGetSelection`(), `XmTextFieldGetSelection`(), and `XmDataFieldGetSelection()` return a pointer to a character string containing the primary selection in the specified *widget*. `XmTextGetSelection`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSelection`() only works for a TextField widget and `XmDataFieldGetSelection()` only works for a TextField widget. For each routine, if no text is selected in the widget, the returned value is NULL. Storage for the returned string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

`XmTextGetSelection`(), `XmTextFieldGetSelection`(), and `XmDataFieldGetSelection()` provide a convenient way to get the current selection from a Text, TextField, or DataField widget.

See Also

`XmTextGetSelectionPosition`(1), `XmTextGetSelectionWcs`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2), `XmDataField(2)`.

Name

XmTextGetSelectionPosition, XmTextFieldGetSelectionPosition,XmDataFieldGetSelectionPosition – get the position of the primary selection.

Synopsis

```
#include <Xm/Text.h>

Boolean XmTextGetSelectionPosition (    Widget          widget,
                                        XmTextPosition  *left,
                                        XmTextPosition  *right)

#include <Xm/TextF.h>

Boolean XmTextFieldGetSelectionPosition (   Widget          widget,
                                            XmTextPosition  *left,
                                            XmTextPosition  *right)
#include <Xm/DataF.h>
char * XmDataFieldGetSelectionPosition      Widget          widget
                                            XmText Position *left,
                                            XmTextPostion   *right)
```

Inputs

widget Specifies the Text or TextField widget.

Outputs

left Returns the position of the left boundary of the primary selection.

right Returns the position of the right boundary of the primary selection.

Returns

True if widget owns the primary selection or False otherwise.

Description

The functions, `XmTextGetSelectionPosition`(), `XmTextFieldGetSelectionPosition`() , and `XmDataFieldGetSelectionPosition()` return the *left* and *right* boundaries of the primary selection for the specified *widget*. `XmTextGetSelectionPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSelectionPosition`() only works for a TextField widget and XmDataFieldGetSelectionPosition() only works for a DataField widget. Each boundary value specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). Each routine returns True if the specified Text, TextField, or DataField *widget* owns the primary selection; otherwise, the routine returns False and the values of *left* and *right* are undefined.

Usage

The functions, XmTextGetSelectionPosition(), XmTextFieldGetSelectionPosition() and XmDataFieldGetSelectionPostion, provide a convenient way to get the position of the current selection from a Text, TextField, or DataField widget.

See Also

XmTextGetSelection(1), XmTextGetSelectionWcs(1), XmTextSetSelection(1), XmText(2), XmTextField(2), XmDataField(2).

Name

XmTextGetSelectionWcs, XmTextFieldGetSelectionWcs – get the wide-character value of the primary selection.

Synopsis

#include <Xm/Text.h>

wchar_t * XmTextGetSelectionWcs (Widget *widget*)

#include <Xm/TextF.h>

wchar_t * XmTextFieldGetSelectionWcs (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

A wide-character string containing the primary selection.

Availability

Motif 1.2 and later.

Description

`XmTextGetSelectionWcs`() and `XmTextFieldGetSelectionWcs`() return a pointer to a wide-character string containing the primary selection in the specified *widget*. `XmTextGetSelectionWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSelectionWcs`() only works for a TextField widget. For each routine, if no text is selected in the widget, the returned value is NULL. Storage for the returned wide-character string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. `XmTextGetSelectionWcs`() and `XmTextFieldGetSelectionWcs`() provide a convenient way to get the current selection in wide-character format from a Text or TextField widget.

See Also

`XmTextGetSelection`(1), `XmTextGetSelectionPosition`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSource – get the text source.

Synopsis

#include <Xm/Text.h>

XmTextSource XmTextGetSource (Widget *widget*)

Inputs

widget Specifies the Text widget.

Returns

The source of the Text widget.

Description

`XmTextGetSource`() returns the source of the specified Text *widget*. Every Text widget has an XmTextSource data structure associated with it that functions as the text source and sink.

Usage

Multiple text widgets can share the same text source, which means that editing in one of the widgets is reflected in all of the others. `XmTextGetSource`() retrieves the source for a widget; this source can then be used to set the source of another Text widget using `XmTextSetSource`(). `XmTextGetSource`() is a convenience routine that returns the value of the XmNsource resource for the Text widget. Calling the routine is equivalent to calling `XtGetValues`() for the resource, although the routine accesses the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextSetSource`(1), `XmText`(2).

Name

XmTextGetString, XmTextFieldGetString – get the text string.

Synopsis

#include <Xm/Text.h>

char * XmTextGetString (Widget *widget*)

#include <Xm/TextF.h>

char * XmTextFieldGetString (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

A string containing the value of the Text or TextField widget.

Description

`XmTextGetString`() and `XmTextFieldGetString`() return a pointer to a character string containing the value of the specified *widget*. `XmTextGetString`() works when widget is a Text widget or a TextField widget, while `XmTextFieldGetString`() only works for a TextField widget. For each routine, if the string has a length of 0 (zero), the returned value is the empty string. Storage for the returned string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

`XmTextGetString`() and `XmTextFieldGetString`() are convenience routines that return the value of the XmNvalue resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you set the XmNvalueWcs resource, you can still use `XmTextGetString`() or `XmTextFieldGetString`() to retrieve the value of the widget, since the value is stored internally as a multi-byte string.

Example

The following routine shows the use of XmTextGetString() to retrieve the text from one Text widget and use the text to search for the string in another Text widget:

```
Widget text_w, search_w;

void search_text (void)
{
    char            *search_pat;
    XmTextPosition  pos, search_pos;
    Boolean         found = False;

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    /* find next occurrence from current position -- wrap if necessary */
    pos = XmTextGetCursorPosition (text_w);
    found = XmTextFindString (text_w, pos, search_pat,
    XmTEXT_FORWARD, &search_pos);

    if (!found)
        found = XmTextFindString (text_w, 0, search_pat,
        XmTEXT_FORWARD, &search_pos);

    if (found)
        XmTextSetInsertionPosition (text_w, search_pos);
        XtFree (search_pat);
}
```

See Also

XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextSetString(1), XmTextSetStringWcs(1), XmText(2), XmTextField(2).

Name

XmTextGetStringWcs, XmTextFieldGetStringWcs – get the wide-character text string.

Synopsis

```
#include <Xm/Text.h>

wchar_t * XmTextGetStringWcs (Widget widget)

#include <Xm/TextF.h>

wchar_t * XmTextFieldGetStringWcs (Widget widget)
```

Inputs

widget Specifies the Text or TextField widget.

Returns

A wide-character string containing the value of the Text or TextField widget.

Availability

Motif 1.2 and later.

Description

`XmTextGetStringWcs`() and `XmTextFieldGetStringWcs`() return a pointer to a wide-character string containing the value of the specified *widget*. `XmTextGetStringWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetStringWcs`() only works for a TextField widget. For each routine, if the string has a length of 0 (zero), the returned value is the empty string. Storage for the returned wide-character string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

`XmTextGetStringWcs`() and `XmTextFieldGetStringWcs`() are convenience routines that return the value of the XmNvalueWcs resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you use the XmNvalue resource to set the value of a widget, you can still use `XmTextGetStringWcs`() or `XmTextFieldGetStringWcs`() to retrieve the value of the widget, since the value can be converted to a wide-character string.

See Also

XmTextGetString(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextSetString(1), XmTextSetStringWcs(1), XmText(2), XmTextField(2).

Name

XmTextGetSubstring, XmTextFieldGetSubstring – get a copy of part of the text string.

Synopsis

```
#include <Xm/Text.h>

int XmTextGetSubstring ( Widget          widget,
                         XmTextPosition  start,
                         int             num_chars,
                         int             buffer_size,
                         char            *buffer)

#include <Xm/TextF.h>

int XmTextFieldGetSubstring (Widget          widget,
                             XmTextPosition  start,
                             int             num_chars,
                             int             buffer_size,
                             char            *buffer)
```

Inputs

widget	Specifies the Text or TextField widget.
start	Specifies the starting character position from which data is copied.
num_chars	Specifies the number of characters that are copied.
buffer_size	Specifies the size of buffer.
buffer	Specifies the character buffer where the copy is stored.

Returns

XmCOPY_SUCCEEDED on success, XmCOPY_TRUNCATED if fewer than *num_chars* are copied, or XmCOPY_FAILED on failure.

Availability

Motif 1.2 and later.

Description

`XmTextGetSubstring`() and `XmTextFieldGetSubstring`() get a copy of part of the internal text buffer for the specified *widget*. `XmTextGetSubstring`()[1] works when widget is a Text widget or a TextField widget, while `XmTextFieldGetSubstring`()[2] only works for a TextField widget. The routines copy *num_chars* characters starting at *start* position, which specifies the

1.Erroneously given as XmTextGetString() in 1st and 2nd editions.

2.Erroneously given as XmTextFieldGetString() in 1st and 2nd editions.

position as the number of characters from the beginning of the text, where the first character position is 0 (zero). The characters are copied into the provided *buffer* and are NULL-terminated.

`XmTextGetSubstring`() and `XmTextFieldGetSubstring`() return XmCOPY_SUCCEEDED on success. If the specified *num_chars* does not fit in the provided *buffer*, the routines return XmCOPY_TRUNCATED. In this case, *buffer* contains as many characters as would fit plus a NULL terminator. If either of the routines fails to make the copy, it returns XmCOPY_FAILED and the contents of *buffer* are undefined.

Usage

`XmTextGetSubstring`() and `XmTextFieldGetSubstring`() provide a convenient way to retrieve a portion of the text string in a Text or TextField widget. The routines return the specified part of the XmNvalue resource for the widget.

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you set the XmNvalueWcs resource, you can still use `XmTextGetSubstring`() or `XmTextFieldGetSubstring`() to retrieve part of the value of the widget, since the value is stored internally as a multi-byte string.

The necessary *buffer_size* for `XmTextGetSubstring`() and `XmTextFieldGetSubstring`() depends on the maximum number of bytes per character for the current locale. This information is stored in MB_CUR_MAX, a macro defined in <stdlib.h>. The *buffer* needs to be large enough to store the substring and a NULL terminator. You can use the following equation to calculate the necessary *buffer_size*:

buffer_size = (num_chars * *MB_CUR_MAX)* + *1*

See Also

`XmTextGetString`(1), `XmTextGetStringWcs`(1), `XmTextGetSubstringWcs`(1), `XmTextSetString`(1), `XmTextSetStringWcs`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSubstringWcs, XmTextFieldGetSubstringWcs – get a copy of part of the wide-character text string.

Synopsis

```
#include <Xm/Text.h>

int XmTextGetSubstringWcs ( Widget          widget,
                            XmTextPosition  start,
                            int             num_chars,
                            int             buffer_size,
                            wchar_t         *buffer)

#include <Xm/TextF.h>

int XmTextFieldGetSubstringWcs ( Widget          widget,
                                 XmTextPosition  start,
                                 int             num_chars,
                                 int             buffer_size,
                                 wchar_t         *buffer)
```

Inputs

widget	Specifies the Text or TextField widget.
start	Specifies the starting character position from which data is copied.
num_chars	Specifies the number of wide-characters that are copied.
buffer_size	Specifies the size of buffer.
buffer	Specifies the wide-character buffer where the copy is stored.

Returns

XmCOPY_SUCCEEDED on success, XmCOPY_TRUNCATED if fewer than num_chars are copied, or XmCOPY_FAILED on failure.

Availability

Motif 1.2 and later.

Description

`XmTextGetSubstringWcs`() and `XmTextFieldGetSubstringWcs`() get a copy of part of the internal wide-character text buffer for the specified *widget*. `XmTextGetSubstringWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSubstringWcs`() only works for a TextField widget. The routines copy num_chars wide-characters starting at *start* position, which specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). The wide-characters are copied into the provided *buffer* and are NULL-terminated.

XmTextGetSubstringWcs() and XmTextFieldGetSubstringWcs() return XmCOPY_SUCCEEDED on success. If the specified *num_chars* does not fit in the provided *buffer*, the routines return XmCOPY_TRUNCATED. In this case, *buffer* contains as many wide-characters as would fit plus a NULL terminator. If either of the routines fails to make the copy, it returns XmCOPY_FAILED and the contents of *buffer* are undefined.

Usage

XmTextGetSubstringWcs() and XmTextFieldGetSubstringWcs() provide a convenient way to retrieve a portion of the wide-character text string in a Text or TextField widget. The routines return the specified part of the XmNvalueWcs resource for the widget.

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you use the XmNvalue resource to set the value of a widget, you can still use XmTextGetSubstringWcs() or XmTextFieldGetSubstringWcs() to retrieve part of the value of the widget, since the value can be converted to a wide-character string.

The necessary *buffer_size* for XmTextGetSubstringWcs() and XmTextFieldGetSubstringWcs() is *num_chars* + 1.

See Also

XmTextGetString(1), XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextSetString(1), XmTextSetStringWcs(1), XmText(2), XmTextField(2).

Name

XmTextGetTopCharacter – get the position of the first character of text that is displayed.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetTopCharacter (Widget *widget*)

Inputs

widget Specifies the Text widget.

Returns

The position of the first visible character.

Description

`XmTextGetTopCharacter`() returns the value of the XmNtopCharacter resource for the specified Text *widget*. The returned value specifies the position of the first visible character of text as the number of characters from the beginning of the text, where the first character position in 0 (zero).

Usage

`XmTextGetTopCharacter`() is a convenience routine that returns the value of the XmNtopCharacter resource for a Text widget. Calling the routines is equivalent to calling `XtGetValues`() for the resource, although the routines accesses the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetLastPosition`(1), `XmTextScroll`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetTopCharacter`(1), `XmTextShowPosition`(1), `XmText`(2).

Name

XmTextInsert, XmTextFieldInsert – insert a string into the text string.

Synopsis

#include <Xm/Text.h>

void XmTextInsert (Widget *widget*, XmTextPosition *position*, char **value*)

#include <Xm/TextF.h>

void XmTextFieldInsert (Widget *widget*, XmTextPosition *position*, char **string*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position at which the string is inserted.
string	Specifies the string to be inserted.

Description

`XmTextInsert`() and `XmTextFieldInsert`() insert a text *string* in the specified Text or TextField *widget*. `XmTextInsert`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldInsert`() only works for a TextField widget. The specified *string* is inserted at *position*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To insert a string after the nth character, use a position value of n.

`XmTextInsert`() and `XmTextFieldInsert`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextInsert`() and `XmTextFieldInsert`() provide a convenient means of inserting text in a Text or TextField widget. The routines insert text by modifying the value of the XmNvalue resource of the widget.

Example

The following routine shows the use of `XmTextInsert`() to insert a message into a status Text widget:

```
Widget status;

void insert_text (char *message)
{
    XmTextPosition curpos = XmTextGetInsertionPosition (status);
```

```
    XmTextInsert (status, curpos, message);

    curpos = curpos + strlen (message);
    XmTextShowPosition (status, curpos);
    XmTextSetInsertionPosition (status, curpos);
}
```

See Also

XmTextInsertWcs(1), XmTextReplace(1), XmTextReplaceWcs(1), XmText(2), XmTextField(2).

Name

XmTextInsertWcs, XmTextFieldInsertWcs – insert a wide-character string into the text string.

Synopsis

#include <Xm/Text.h>

void XmTextInsertWcs (Widget *widget*, XmTextPosition *position*, wchar_t **wcstring*)[1]

#include <Xm/TextF.h>

void XmTextFieldInsertWcs (Widget *widget*, XmTextPosition *position*, wchar_t * *wcstring*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position at which the string is inserted.
wcstring	Specifies the wide-character string to be inserted.

Availability

Motif 1.2 and later.

Description

`XmTextInsertWcs`() and `XmTextFieldInsertWcs`() insert a wide-character text *wcstring* in the specified *widget*. `XmTextInsertWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldInsertWcs`() only works for a TextField widget. The specified *wcstring*[2] is inserted at *position*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To insert a string after the *n*th character, use a position value of n.

`XmTextInsertWcs`() and `XmTextFieldInsertWcs`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

1.Erroneously given as XmTextInsert() in 1st and 2nd editions.

2.Erroneously given as *string* in 1st and 2nd editions.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. XmTextInsertWcs() and XmTextFieldInsertWcs*()* provide a convenient means of inserting a wide-character string in a Text or TextField widget. The routines insert text by converting the wide-character string to a multi-byte string and then modifying the value of the XmNvalue resource of the widget.

See Also

XmTextInsert(1), XmTextReplace(1), XmTextReplaceWcs(1), XmText(2), XmTextField(2).

Name

XmTextPaste, XmTextFieldPaste – insert the clipboard selection.

Synopsis

#include <Xm/Text.h>

Boolean XmTextPaste (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldPaste (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

True on success or False otherwise.

Description

`XmTextPaste`() and `XmTextFieldPaste`() insert the clipboard selection at the current position of the insertion cursor in the specified *widget*. `XmText-Paste`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldPaste`() only works for a TextField widget. If the insertion cursor is within the current selection and the value of XmNpendingDelete is True, the current selection is replaced by the clipboard selection. Both routines return True if successful. If the *widget* is not editable or if the function cannot obtain ownership of the clipboard selection, the routines return False.

In Motif 2.0 and later, `XmTextPaste`() interfaces with the Uniform Transfer Model by indirectly invoking the XmNdestinationCallback procedures of the widget.

`XmTextPaste`() and `XmTextFieldPaste`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextPaste`() and `XmTextFieldPaste`() get the current selection from the clipboard and insert it at the location of the insertion cursor in the Text or TextField widget.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of `XmTextPaste`()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                     num = (int) client_data;
    XmAnyCallbackStruct     *cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);
        break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbut-
        ton.time);break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

`XmTextCopy`(1), `XmTextCopyLink`(1), `XmTextCut`(1), `XmTextPasteLink`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextPasteLink, XmTextFieldPasteLink – insert the clipboard selection.

Synopsis

#include <Xm/Text.h>

Boolean XmTextPasteLink (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldPasteLink (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

True on success or False otherwise.

Availability

Motif 2.0 and later.

Description

`XmTextPasteLink`() and `XmTextFieldPasteLink`() insert the clipboard selection at the current position of the insertion cursor in the specified *widget*. `XmTextPasteLink`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldPasteLink`() only works for a TextField widget. If the insertion cursor is within the current selection and the value of XmNpendingDelete is True, the current selection is replaced by the clipboard selection. Both routines return True if successful. If the *widget* is not editable or if the function cannot obtain ownership of the clipboard selection, the routines return False.

`XmTextPasteLink`() and `XmTextFieldPasteLink`() interface with the Uniform Transfer Model by indirectly invoking the XmNdestinationCallback procedures of the widget. The Text widget itself does not create links to the primary selection: destination callbacks provided by the programmer are responsible for performing any data transfer required.

`XmTextPasteLink`() and `XmTextFieldPasteLink`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

XmTextPasteLink() and XmTextFieldPasteLink() get the current selection from the clipboard and insert a link to it at the location of the insertion cursor in the Text or TextField widget.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of XmTextPasteLink()[1]**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                     num = (int) client_data;
    XmAnyCallbackStruct     *cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);
        break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbut-
        ton.time);break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

XmTextCopy(1), XmTextCopyLink(1), XmTextCut(1), XmTextPaste(1), XmText(2), XmTextField(2).

1.Erroneously given as XmTextPaste() in 1st and 2nd editions.

Name

XmTextPosToXY, XmTextFieldPosToXY – get the x, y position of a character position.

Synopsis

#include <Xm/Text.h>

Boolean XmTextPosToXY (Widget *widget*, XmTextPosition *position*, Position **x*, Position **y*)

#include <Xm/TextF.h>

Boolean XmTextFieldPosToXY (Widget *widget*, XmTextPosition *position*, Position **x*, Position **y*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the character position.

Outputs

x	Returns the x-coordinate of the character position.
y	Returns the y-coordinate of the character position.

Returns

True if the character position is displayed in the widget or False otherwise.

Description

`XmTextPosToXY`() and `XmTextFieldPosToXY`() return the *x* and *y* coordinates of the character at the specified *position* within the specified *widget*. `XmTextPosToXY`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldPosToXY`() only works for a TextField widget. Character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. The returned coordinate values are specified relative to the upper-left corner of widget. Both routines return True if the character at *position* is currently displayed in the widget. Otherwise, the routines return False and no values are returned in the x and y arguments.

Usage

`XmTextPosToXY`() and `XmTextFieldPosToXY`() provide a way to determine the actual position of a character in a Text or TextField widget. This information is useful if you need to perform additional event processing or draw special graphics in the widget.

See Also

`XmTextXYToPos`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextRemove, XmTextFieldRemove – delete the primary selection.

Synopsis

#include <Xm/Text.h>

Boolean XmTextRemove (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldRemove (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

True on success or False otherwise.

Description

`XmTextRemove`() and `XmTextFieldRemove`() delete the primary selected text from the specified *widget*. `XmTextRemove`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldRemove`() only works for a TextField widget. Both routines return True if successful. If the *widget* is not editable, if the primary selection is NULL, or if it is not owned by the specified widget, the routines return False.

`XmTextRemove`() and `XmTextFieldRemove`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextRemove`() and `XmTextFieldRemove`() are like `XmTextCut`() and `XmTextFieldCut`(), in that they remove selected text from a Text or TextField widget. However, the routines do not copy the selected text to the clipboard before removing it.

See Also

`XmTextClearSelection`(1), `XmTextCut`(1), `XmTextGetSelection`(1), `XmTextGetSelectionPosition`(1), `XmTextGetSelectionWcs`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextReplace, XmTextFieldReplace – replace part of the text string.

Synopsis

```
#include <Xm/Text.h>

void XmTextReplace (Widget          widget,
                    XmTextPosition  from_pos,
                    XmTextPosition  to_pos,
                    char            *value)

#include <Xm/TextF.h>

void XmTextFieldReplace ( Widget          widget,
                          XmTextPosition  from_pos,
                          XmTextPosition  to_pos,
                          char            *value)
```

Inputs

widget	Specifies the Text or TextField widget.
from	Specifies the starting position of the text that is to be replaced.
to	Specifies the ending position of the text that is to be replaced.
value	Specifies the replacement string.

Description

`XmTextReplace`() and `XmTextFieldReplace`() replace a portion of the text *string* in the specified *widget*. `XmTextReplace`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldReplace`() only works for a TextField widget. The specified *value* replaces the text starting at *from_pos* and continuing up to, but not including, *to_pos*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To replace the characters after the *n*th character up to the *m*th character, use a *from_pos* value of *n* and a *to_pos* value of *m*.

`XmTextReplace`() and `XmTextFieldReplace`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextReplace`() and `XmTextFieldReplace`() provide a convenient means of replacing text in a Text or TextField widget. The routines replace text by modifying the value of the XmNvalue resource of the widget.

Example

The following routine shows the use of XmTextReplace() to replace all of the occurrences of a string in a Text widget. The search and replacement strings are specified by the user in single-line Text widgets:

```
Widget text_w, search_w, replace_w;

void search_and_replace (void)
{
    char            *search_pat, *new_pat;
    XmTextPosition  curpos, searchpos;
    int             search_len, pattern_len;
    Boolean         found = False;

    search_len = XmTextGetLastPosition (search_w);

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    pattern_len = XmTextGetLastPosition (replace_w);

    if (!(new_pat = XmTextGetString (replace_w)) || !*new_pat) {
        XtFree (search_pat);
        XtFree (new_pat);
        return;
    }

    curpos = 0;
    found = XmTextFindString (text_w, curpos, search_pat,
    XmTEXT_FORWARD, &searchpos);

    while (found) {
        XmTextReplace (text_w, searchpos, searchpos + search_len, new_pat);
        curpos = searchpos + 1;
        found = XmTextFindString (text_w, curpos, search_pat,
        XmTEXT_FORWARD, &searchpos);
    }

    XtFree (search_pat);
    XtFree (new_pat);
}
```

See Also

XmTextInsert(1), XmTextInsertWcs(1), XmTextReplaceWcs(1), XmText(2), XmTextField(2).

Name

XmTextReplaceWcs, XmTextFieldReplaceWcs – replace part of the wide-character text string.

Synopsis

```
#include <Xm/Text.h>

void XmTextReplaceWcs (Widget          widget,
                       XmTextPosition  from_pos,
                       XmTextPosition  to_pos,
                       wchar_t         *wcstring)

#include <Xm/TextF.h>

void XmTextFieldReplaceWcs ( Widget          widget,
                             XmTextPosition  from_pos,
                             XmTextPosition  to_pos,
                             wchar_t         *wcstring)
```

Inputs

widget	Specifies the Text or TextField widget.
from_pos	Specifies the starting position of the text that is to be replaced.
to_pos	Specifies the ending position of the text that is to be replaced.
wcstring	Specifies the replacement wide-character string.

Availability

Motif 1.2 and later.

Description

`XmTextReplaceWcs`() and `XmTextFieldReplaceWcs`() replace a portion of the text string in the specified widget with the specified wide-character string *wcstring*. `XmTextReplaceWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldReplaceWcs`() only works for a TextField widget. The specified *wcstring* replaces the text starting at *from_pos* and continuing up to, but not including, *to_pos*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To replace the characters after the *n*th character up to the *m*th character, use a *from_pos* value of *n* and a *to_pos* value of *m*.

`XmTextReplaceWcs`() and `XmTextFieldReplaceWcs`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. XmTextReplaceWcs() and XmTextFieldReplaceWcs() provide a convenient means of replacing a string in a Text or TextField widget with a wide-character string. The routines convert the wide-character string to a multi-byte string and then replace the text by modifying the value of the XmNvalue resource of the widget.

See Also

XmTextInsert(1), XmTextInsertWcs(1), XmTextReplace(1), XmText(2), XmTextField(2).

Name

XmTextScroll – scroll the text.

Synopsis

#include <Xm/Text.h>

void XmTextScroll (Widget *widget*, int *lines*)

Inputs

widget	Specifies the Text widget.
lines	Specifies the number of lines.

Description

`XmTextScroll`() scrolls the text in the specified Text *widget* by the specified number of *lines*. The text is scrolled upward if *lines* is positive and downward if *lines* is negative. In the case of vertical text, a positive value scrolls the text forwards, and a negative value scrolls backwards.

Usage

`XmTextScroll`() provides a way to perform relative scrolling in a Text widget. The Text widget does not have to be the child of a ScrolledWindow for the scrolling to occur. The routine simply changes the currently viewable region of text.

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetLastPosition`(1), `XmTextGetTopCharacter`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetTopCharacter`(1), `XmText`(2).

Name

XmTextSetAddMode, XmTextFieldSetAddMode, XmDataFieldSetAddMode – set the add mode state.

Synopsis

#include <Xm/Text.h>

void XmTextSetAddMode (Widget *widget*, Boolean *state*)

#include <Xm/TextF.h>

void XmTextFieldSetAddMode (Widget *widget*, Boolean *state*)

#include <Xm/DataF.h>

void XmDataFieldSetAddMode (Widget *widget*, Boolean *state*)

Inputs

widget	Specifies the Text or TextField widget.
state	Specifies the state of add mode.

Description

`XmTextSetAddMode`(), `XmTextFieldSetAddMode`(), and `XmDataFieldSetAddMode()` set the state of add mode for the specified *widget*. `XmTextSetAddMode`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetAddMode`() only works for a TextField widget and `XmDataFieldSetAddMode()` only works for a DataField widget. If *state* is True add mode is turned on; if *state* is False, add mode is turned off. When a Text, TextField, or DataField widget is in add mode, the user can move the insertion cursor without altering the primary selection.

Usage

`XmTextSetAddMode`(), `XmTextFieldSetAddMode`(), and `XmDataFieldSetAddMode()` provide a way to change the state of add mode in a Text, TextField, or DataField widget. The distinction between normal mode and add mode is only important for making keyboard-based selections. In normal mode, the location cursor and the selection move together, while in add mode, the location cursor and the selection can be separate.

See Also

`XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmText`(2), `XmTextField`(2), `XmDataField(2)`.

Name

XmTextSetCursorPosition, XmTextFieldSetCursorPosition – set the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

void XmTextSetCursorPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/TextF.h>

void XmTextFieldSetCursorPosition (Widget *widget*, XmTextPosition *position*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position of the insertion cursor.

Description

`XmTextSetCursorPosition`() and `XmTextFieldSetCursorPosition`() set the value of the XmNcursorPosition resource to position for the specified *widget*. `XmTextSetCursorPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetCursorPosition`() only works for a TextField widget. This resource specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

`XmTextSetCursorPosition`() and `XmTextFieldSetCursorPosition`() also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget if the position of the insertion cursor changes.

Usage

`XmTextSetCursorPosition`() and `XmTextFieldSetCursorPosition`() are convenience routines that set the value of the XmNcursorPosition resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetEditable, XmTextFieldSetEditable, XmDataFieldSetEditable – set the edit permission state.

Synopsis

#include <Xm/Text.h>

void XmTextSetEditable (Widget *widget*, Boolean *editable*)

#include <Xm/TextF.h>

void XmTextFieldSetEditable (Widget *widget*, Boolean *editable*)

#include <Xm/DataF.h>

void XmDataFieldSetEditable (Widget *widget*, Boolean *editable*)

Inputs

widget	Specifies the Text or TextField widget.
editable	Specifies whether or not the text can be edited.

Description

`XmTextSetEditable`(), `XmTextFieldSetEditable`(), and `XmDataFieldSetEditable`() set the value of the XmNeditable resource to editable for the specified *widget*. `XmTextSetEditable`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetEditable`() only works for a TextField widget and `XmDataFieldSetEditable`() only works for a DataField widget.

Usage

By default, the XmNeditable resource is True, which means that a user can edit the text string. Setting the resource to False makes a text area read-only. `XmTextSetEditable`(), `XmTextFieldSetEditable`(), and `XmDataFieldSetEditable`() are convenience routines that set the value of the XmNeditable resource for a Text, TextField, or DataField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

See Also

`XmTextGetEditable`(1), `XmText`(2), `XmTextField`(2), `XmDataField`(2).

Name

XmTextSetHighlight, XmTextFieldSetHighlight, XmDataFieldSetHighlight – highlight text.

Synopsis

```
#include <Xm/Text.h>

void XmTextSetHighlight (  Widget          widget,
                           XmTextPosition  left,
                           XmTextPosition  right,
                           XmHighlightMode mode)

#include <Xm/TextF.h>

void XmTextFieldSetHighlight (  Widget          widget,
                                XmTextPosition  left,
                                XmTextPosition  right,
                                XmHighlightMode mode)
#include <Xm/DataF.h>

void XmDataFieldSetHighlight (  Widget          widget,
                                XmTextPosition  left,
                                XmTextPosition  right,
                                XmHighlightMode mode
```

Inputs

widget	Specifies the Text, TextField, or DataField widget.
left	Specifies the left boundary position of the text to be highlighted.
right	Specifies the right boundary position of the text to be highlighted.
mode	Specifies the highlighting mode. Pass one of the following values: XmHIGHLIGHT_NORMAL, XmHIGHLIGHT_SELECTED, or XmHIGHLIGHT_SECONDARY_SELECTED[1].

Description

`XmTextSetHighlight`(), `XmTextFieldSetHighlight`(), and `XmDataFieldSetHighlight()` highlight text in the specified *widget* without selecting the text. `XmTextSetHighlight`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetHighlight`() only works for a TextField widget and `XmDataFieldSetHighlight()` only works for a DataField widget. The *left* and *right* arguments specify the boundary

1. Motif 2.0 defines the additional value XmSEE_DETAIL for the enumerated type, but does not use it for the Text components. The Compound String Text, CSText, supports the notion, but this widget is abortive, and has been removed from the 2.1 distribution. XmSEE_DETAIL is therefore redundant.

positions of the text that is to be highlighted. Each boundary value specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). The *mode* parameter indicates the type of highlighting that is done. XmHIGHLIGHT_NORMAL removes any highlighting, XmHIGHLIGHT_SELECTED uses reverse video highlighting, and XmHIGHLIGHT_SECONDARY_SELECTED uses underline highlighting.

Usage

`XmTextSetHighlight`(), `XmTextFieldSetHighlight`(), and `XmDataFieldSetHighlight()` provide a way to highlight text in a Text, TextField, or DataField widget. These routines are useful if you need to emphasize certain text in a widget. These routine only highlight text; they do not select the specified text.

Example

The following routine shows the use of `XmTextSetHighlight`() to highlight all of the occurrences of a string in a Text widget. The search string is specified by the user in a single-line Text widget:

```
Widget text_w, search_w;

void search_text (void)
{
    char            *search_pat;
    XmTextPosition  curpos, searchpos;
    int             len;
    Boolean         found = False;

    len = XmTextGetLastPosition (search_w);

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    curpos = 0;
    found = XmTextFindString (text_w, curpos, search_pat,
    XmTEXT_FORWARD, &searchpos);

    while (found) {
        XmTextSetHighlight (text_w, searchpos, searchpos + len,
                            XmHIGHLIGHT_SECONDARY_SELECTE
                            D);
        curpos = searchpos + 1;
        found = XmTextFindString (text_w, curpos, search_pat,
        XmTEXT_FORWARD, &searchpos);
```

```
    }
    XtFree (search_pat);
}
```

See Also

XmTextSetSelection(1), XmText(2), XmTextField(2), XmDataField(2).

Name

XmTextSetInsertionPosition, XmTextFieldSetInsertionPosition, XmDataFieldSetInsertionPosition – set the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

void XmTextSetInsertionPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/TextF.h>

void XmTextFieldSetInsertionPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/DataF.h>

void XmDataFieldSetInsertionPosition (Widget *widget*, XmTextPosition *position*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position of the insertion cursor.

Description

The functions, `XmTextSetInsertionPosition`(), `XmTextFieldSetInsertionPosition`(), and `XmDataFieldSetInsertionPosition`() set the value of the XmNcursorPosition resource to *position* for the specified *widget*. `XmTextSetInsertionPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetInsertionPosition`() only works for a TextField widget and `XmDataFieldSetInsertionPosition`() only works for a DataField widget. This resource specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

`XmTextSetInsertionPosition`(), `XmTextFieldSetInsertionPosition`(), and `XmDataFieldSetInsertionPosition`() also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget if the position of the insertion cursor changes.

Usage

The functions, `XmTextSetInsertionPosition`(), `XmTextFieldSetInsertionPosition`(), and `XmDataFieldSetInsertionPosition`() are convenience routines that set the value of the XmNcursorPosition resource for a Text, TextField , or DataField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines

access the value through the widget instance structures rather than through XtSetValues().

Example

The following code shows the use of XmTextSetInsertionPosition() in a routine that searches for a string in a Text widget and moves the insertion cursor to the string if it is found:

```
Widget text_w, search_w;

void search_text (void)
{
    char            *search_pat;
    XmTextPosition  pos, searchpos;
    Boolean         found = False;

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    pos = XmTextGetCursorPosition (text_w);
    found = XmTextFindString (text_w, pos, search_pat,
    XmTEXT_FORWARD, &searchpos);

    if (!found) {
        found = XmTextFindString (text_w, 0, search_pat,
        XmTEXT_FORWARD, &searchpos);
    }

    if (found)
        XmTextSetInsertionPosition (text_w, searchpos);
    XtFree (search_pat);
}
```

See Also

XmTextGetCursorPosition(1), XmTextGetInsertionPosition(1), XmTextSetCursorPosition(1), XmTextShowPosition(1), XmText(2), XmTextField(2), XmDataField(2).

Name

XmTextSetMaxLength, XmTextFieldSetMaxLength – set the maximum possible length of a text string.

Synopsis

#include <Xm/Text.h>

void XmTextSetMaxLength (Widget *widget*, int *max_length*)

#include <Xm/TextF.h>

void XmTextFieldSetMaxLength (Widget *widget*, int *max_length*)

Inputs

widget	Specifies the Text or TextField widget.
max_length	Specifies the maximum allowable length of the text string.

Description

`XmTextSetMaxLength`() and `XmTextFieldSetMaxLength`() set the value of the XmNmaxLength resource to *max_length* for the specified *widget*. `XmTextSetMaxLength`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetMaxLength`() only works for a TextField widget. This resource specifies the maximum allowable length of a text string that a user can enter from the keyboard.

Usage

`XmTextSetMaxLength`() and `XmTextFieldSetMaxLength`() are convenience routines that set the XmNmaxLength resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`(). The resource limits the length of a text string that a user may type, but it does not limit the length of strings entered with the XmNvalue or XmNvalueWcs resources or the `XmTextSetString`(), `XmTextFieldSetString`(), `XmTextSetStringWcs`(), and `XmTextFieldSetStringWcs`() routines.

See Also

`XmTextGetMaxLength`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetSelection, XmTextFieldSetSelection, XmDataFieldSetSelection – set the value of the primary selection.

Synopsis

#include <Xm/Text.h>

void XmTextSetSelection (Widget *widget*, XmTextPosition *first*, XmTextPosition *last*, Time *time*)

#include <Xm/TextF.h>

void XmTextFieldSetSelection (Widget *widget*, XmTextPosition *first*, XmTextPosition *last*, Time *time*)

#include <Xm/DataF.h>

char * XmDataFieldSetSelection (Widget *widget*)

Inputs

widget	Specifies the Text or TextField widget.
first	Specifies the first character position to be selected.
last	Specifies the last character position to be selected.
time	Specifies the time of the event that caused the request.

Description

`XmTextSetSelection`(), `XmTextFieldSetSelection`(), and `XmDataFieldSetSelection()` set the primary selection in the specified *widget*. `XmTextSetSelection`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetSelection`() only works for a TextField widget and `XmDataFieldSetSelection()` only works for the TextField widget. The *first* and *last* arguments specify the beginning and ending positions of the text that is to be selected. Each of these values specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). For each routine, *time* specifies the server time of the event that caused the request to set the selection.

`XmTextSetSelection`(), `XmTextFieldSetSelection`(), and `XmDataFieldSetSelection()` change the insertion cursor for the *widget* to the last position of the selection. The routines also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget.

Usage

`XmTextSetSelection`(), `XmTextFieldSetSelection`(), and `XmDataFieldSetSelection()` provide a convenient way to set the current selection in a Text, TextField, or DataField widget.

See Also

XmTextClearSelection(1), XmTextCopy(1), XmTextCut(1), XmTextGetSelection(1), XmTextGetSelectionPosition(1), XmTextGetSelectionWcs(1), XmTextRemove(1), XmText(2), XmTextField(2), XmDataField(2).

Name

XmTextSetSource – set the text source.

Synopsis

```
#include <Xm/Text.h>

void XmTextSetSource (Widget        widget,
                      XmTextSource  source,
                      XmTextPosition top_character,
                      XmTextPosition cursor_position)
```

Inputs

widget	Specifies the Text widget.
source	Specifies the text source.
top_character	Specifies the character position to display at the top of the widget.
cursor_position	Specifies the position of the insertion cursor.

Description

`XmTextSetSource`() sets the source of the specified Text *widget*. The *top_character* and *cursor_position* values specify positions as the number of characters from the beginning of the text, where the first character position is 0 (zero). If *source* is NULL, the Text widget creates a default string source and displays a warning message.

Usage

Multiple text widgets can share the same text source, which means that editing in one of the widgets is reflected in all of the others. `XmTextGetSource`() retrieves the source for a widget; this source can then be used to set the source of another Text widget using `XmTextSetSource`(). `XmTextSetSource`() is a convenience routine that sets the value of the XmNsource resource for the Text widget. Calling the routine is equivalent to calling `XtSetValues`() for the resource, although the routine accesses the value through the widget instance structures rather than through `XtSetValues`().

When a new text source is set, the old text source is destroyed unless another Text widget is using the old source. If you want to replace a text source without destroying it, create an unmanaged Text widget and set its source to the text source you want to save.

See Also

`XmTextGetSource`(1), `XmText`(2).

Name

XmTextSetString, XmTextFieldSetString, XmDataFieldSetString – set the text string.

Synopsis

#include <Xm/Text.h>

void XmTextSetString (Widget *widget*, char **value*)

#include <Xm/TextF.h>

void XmTextFieldSetString (Widget *widget*, char **value*)

#include <Xm/DataF.h>

void XmTextField (Widget *widget*, char **value*)

Inputs

widget	Specifies the Text or TextField widget.
value	Specifies the string value.

Availability

XmDataFieldSetString()-Motif 2.2 and later

Description

`XmTextSetString`(), `XmTextFieldSetString`(), and `XmDataFieldSetString()` set the current text string in the specified *widget* to the specified *value*. `XmTextSetString`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetString`() only works for a TextField widget and `XmDataFieldSetString()` only works for a DataField widget. Both functions also set the position of the insertion cursor to the beginning of the new text string.

`XmTextSetString`(), `XmTextFieldSetString`(), and `XmDataFieldSetString()` invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures. The routines also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget.

Usage

`XmTextSetString`(), `XmTextFieldSetString`(), and XmDataFieldSetString are convenience routines that set the value of the XmNvalue resource for a Text, TextField, or DataField widget. Calling one of the routines is equivalent to

calling XtSetValues() for the resource, although the routines access the value through the widget instance structures rather than through XtSetValues().

Example

The following code shows the use of XmTextSetString() in a routine that displays the contents of file in a Text widget. The filename is specified by the user in a TextField widget:

```
Widget text_w, file_w;

void read_file (void)
{
    char        *filename, *text;
    struct stat statb;
    int         fd, len;

    if (!(filename = XmTextFieldGetString (file_w)) || !*filename) {
        XtFree (filename);
        return;
    }

    if (!(fd = open (filename, O_RDONLY))) {
        XtWarning ("internal error -- can't open file");
    }

    if (fstat (fd, &statb) == -1 || !(text = XtMalloc ((len = statb.st_size) + 1))) {
        XtWarning("internal error -- can't show text");
        (void) close (fd);
    }

    (void) read (fd, text, len);
    text[len] = '\0';
    XmTextSetString (text_w, text);
    XtFree (text);
    XtFree (filename);
    (void) close (fd);
}
```

See Also

XmTextGetString(1), XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextSetStringWcs(1), XmText(2), XmTextField(2).

Name

XmTextSetStringWcs, XmTextFieldSetStringWcs, XmDataFieldSetStringWcs – set the wide-character text string.

Synopsis

#include <Xm/Text.h>

void XmTextSetStringWcs (Widget *widget*, wchar_t **wcstring*)

#include <Xm/TextF.h>

void XmTextFieldSetStringWcs (Widget *widget*, wchar_t **wcstring*)

#include <Xm/DataF.h>

void XmDataFieldSetStringWcs (Widget *widget*, wchar_t **wcstring*)

Inputs

widget	Specifies the Text or TextField widget.
wcstring	Specifies the wide-character string value.

Availability

XmDataFieldSetStringWcs()- Motif 2.2 and later, XmTextSetStringWcs, XmTextFieldSetStringWcs- Motif 1.2 and later.

Description

`XmTextSetStringWcs`(), `XmTextFieldSetStringWcs`(), and `XmDataFieldSetStringWcs()` set the current wide-character text string in the specified *widget* to the specified *wcstring*[1]. `XmTextSetStringWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetStringWcs`() only works for a TextField widget and `XmDataFieldSetStringWcs()` only works for a DataField widget. Both functions also set the position of the insertion cursor to the beginning of the new text string.

`XmTextSetStringWcs`(), `XmTextFieldSetStringWcs`(), and `XmDataFieldSetStringWcs()` invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures. The routines also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget.

1.Erroneously given as *string* in 1st and 2nd editions.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. XmTextSetStringWcs(), XmTextFieldSetStringWcs(), and XmDataFieldSetStringWcs() are convenience routines that set the value of the XmNvalueWcs resource for a Text, TextField, or DataField widget. Calling one of the routines is equivalent to calling XtSetValues() for the resource, although the routines access the value through the widget instance structures rather than through XtSetValues().

See Also

XmTextGetString(1), XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextSetString(1), XmText(2), XmTextField(2), XmDataField(2).

Name

XmTextSetTopCharacter – set the position of the first character of text that is displayed.

Synopsis

#include <Xm/Text.h>

void XmTextSetTopCharacter (Widget *widget*, XmTextPosition *top_character*)

Inputs

widget Specifies the Text widget.
top_character Specifies the position that is to be displayed at the top of the widget.

Description

`XmTextSetTopCharacter`() sets the value of the XmNtopCharacter resource to *top_character* for the specified Text *widget*. If the XmNeditMode resource is set to XmMULTI_LINE_EDIT, the routine scrolls the text so that the line containing the character position specified by *top_character* appears at the top of the widget, but does not shift the text left or right. Otherwise, the character position specified by *top_character* is displayed as the first visible character in the widget. *top_character* specifies a character position as the number of characters from the beginning of the text, where the first character position in 0 (zero).

Usage

`XmTextSetTopCharacter`() is a convenience routine that sets the value of the XmNtopCharacter resource for a Text widget. Calling the routines is equivalent to calling `XtSetValues`() for the resource, although the routines accesses the value through the widget instance structures rather than through `XtSetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetLastPosition`(1), `XmTextGetTopCharacter`(1), `XmTextScroll`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextShowPosition`(1), `XmText`(2).

Name

XmTextShowPosition, XmTextFieldShowPosition, XmDataFieldShowPosition – display the text at a specified position.

Synopsis

#include <Xm/Text.h>

void XmTextShowPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/TextF.h>

void XmTextFieldShowPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/DataF.h>

void XmDataFieldShowPosition (Widget *widget*, XmTextPosition *position*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the character position that is to be displayed.

Description

`XmTextShowPosition`(), `XmTextFieldShowPosition`(), and `XmDataFieldShowPosition()` cause the text character at *position* to be displayed in the specified *widget*. `XmTextShowPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldShowPosition`() only works for a TextField widget and `XmDataFieldShowPosition()` only works for a DataField widget. The *position* argument specifies the position as the number of characters from the beginning of the text, where the first character position in 0 (zero).

Usage

`XmTextShowPosition`() and `XmTextFieldShowPosition`() provide a way to force a Text or TextField widget to display a certain portion of its text. This routine is useful if you modify the value of widget and want the modification to be immediately visible without the user having to scroll the text. If the value of the XmNautoShowCursorPosition resource is True, you should set the insertion cursor to position as well. You can set the insertion cursor by setting the XmNcursorPosition[1] resource or by using `XmTextSetInsertionPosition`(), `XmTextFieldSetInsertionPosition`(), or `XmDataFieldSetInsertionPosition()`.

1.Erroneously given as XmcursorPosition in 1st and 2nd editions.

Example

The following code shows the use of XmTextShowPosition() in a routine that inserts a message into a status Text widget:

```
Widget status;

void insert_text (char *message)
{
    XmTextPosition curpos = XmTextGetInsertionPosition (status);

    XmTextInsert (status, curpos, message);
    curpos = curpos + strlen (message);
    XmTextShowPosition (status, curpos);
    XmTextSetInsertionPosition (status, curpos);
}
```

See Also

XmTextGetCursorPosition(1), XmTextGetInsertionPosition(1), XmTextSetCursorPosition(1), XmTextSetInsertionPosition(1), XmText(2), XmTextField(2), XmDataField(2).

Name

XmTextXYToPos, XmTextFieldXYToPos, XmDataFieldXYToPos – get the character position for an x, y position.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextXYToPos (Widget *widget*, Position *x*, Position *y*)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldXYToPos (Widget *widget*, Position *x*, Position *y*)

#include <Xm/DataF.h>

XmTextPosition XmDataFieldXYToPos (Widget *widget*, Position *x*, Position *y*)

Inputs

widget	Specifies the Text or TextField widget.
x	Specifies the x-coordinate relative to the upper-left corner of the widget.
y	Specifies the y-coordinate relative to the upper-left corner of the widget.

Returns

The character position that is closest to the x, y position.

Description

`XmTextXYToPos`(), `XmTextFieldXYToPos`(), and `XmDataFieldXYToPos()` return the position of the character closest to the specified *x* and *y* coordinates within the specified *widget*. `XmTextXYToPos`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldXYToPos`() only works for a TextField widget and `XmDataFieldXYToPos()` only works for a DataField. The *x* and *y* coordinates are relative to the upper-left corner of the widget. Character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text.

Usage

`XmTextXYToPos`(), `XmTextFieldXYToPos`(), and `XmTextXYToPos()` provide a way to determine the character at a particular coordinate in a Text, TextField, or DataField widget. This information is useful if you need to perform additional event processing or draw special graphics in the widget.

See Also

`XmTextPosToXY`(1), `XmText`(2), `XmTextField`(2), `XmDataField(2)`.

Name

XmToggleButtonGetState, XmToggleButtonGadgetGetState – get the state of a ToggleButton.

Synopsis

#include <Xm/ToggleB.h>

Boolean XmToggleButtonGetState (Widget *widget*)

#include <Xm/ToggleBG.h>

Boolean XmToggleButtonGadgetGetState (Widget *widget*)

Inputs

widget Specifies the ToggleButton or ToggleButtonGadget.

Returns

The state of the button.

Description

`XmToggleButtonGetState`() and `XmToggleButtonGadgetGetState`() return the state of the specified *widget*. `XmToggleButtonGetState`() works when *widget* is a ToggleButton or a ToggleButtonGadget, while `XmToggleButtonGadgetGetState`() only works for a ToggleButtonGadget. In Motif 1.2 and earlier, each of the routines returns True if the button is selected or False if the button is unselected.

In Motif 2.0 and later, a Toggle can be in any of three states: XmSET, XmINDETERMINATE, and XmUNSET, where XmUNSET is equivalent to False and XmSET is equivalent to True. The third indeterminate state is enabled if the Motif 2.x XmNtoggleMode resource of the widget is set to the value XmTOGGLE_INDETERMINATE. If the toggle mode is XmTOGGLE_BOOLEAN, the widget has only two dynamic states, which is consistent with Motif 1.2 behavior.

Usage

`XmToggleButtonGetState`() and `XmToggleButtonGadgetGetState`() are convenience routines that return the value of the XmNset resource for a ToggleButton or ToggleButtonGadget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

Because XmToggleButtonGetState() returns the toggle *set* element of its widget instance structure directly, and because XmINDETERMINATE is neither True nor False, programs relying on the strictly Boolean nature of XmToggleButtonGetState() are at risk of error if the toggle is configured for three states. Setting tri-state toggles using a convenience function should be performed using XmToggleButtonSetValue().

See Also

XmToggleButtonSetState(1), XmToggleButtonSetValue(1), XmToggleButton(2), XmToggleButtonGadget(2).

Name

XmToggleButtonSetState, XmToggleButtonGadgetSetState – set the state of a ToggleButton.

Synopsis

#include <Xm/ToggleB.h>

void XmToggleButtonSetState (Widget *widget*, Boolean *state*, Boolean *notify*)

#include <Xm/ToggleBG.h>

void XmToggleButtonGadgetSetState (Widget *widget*, Boolean *state*, Boolean *notify*)

Inputs

widget	Specifies the ToggleButton or ToggleButtonGadget.
state	Specifies the state of the button.
notify	Specifies whether or not the XmNvalueChangedCallback is called.

Description

`XmToggleButtonSetState`() and `XmToggleButtonGadgetSetState`() set the state of the specified *widget*. `XmToggleButtonSetState`() works when *widget* is a ToggleButton or a ToggleButtonGadget, while `XmToggleButtonGadgetSetState`() only works for a ToggleButtonGadget. In Motif 1.2 and earlier, if *state* is True, the button is selected, and when *state* is False, the button is deselected.

In Motif 2.0 and later, a Toggle can be in any of three states: XmSET, XmINDETERMINATE, and XmUNSET, where XmUNSET is equivalent to False and XmSET is equivalent to True. The third indeterminate state is enabled if the Motif 2.x XmNtoggleMode resource of the widget is set to the value XmTOGGLE_INDETERMINATE. If the toggle mode is XmTOGGLE_BOOLEAN, the widget has only two dynamic states, which is consistent with Motif 1.2 behavior.

If *notify* is True, the routines invoke the callbacks specified by the XmNvalueChangedCallback resource. If the specified *widget* is the child of a RowColumn with XmNradioBehavior set to True, the currently selected child of the RowColumn is deselected.

Usage

XmToggleButtonSetState() and XmToggleButtonGadgetSetState() are convenience routines that set the value of the XmNset resource for a ToggleButton or ToggleButtonGadget. Calling one of the routines is equivalent to calling XtSetValues() for the resource, although the routines access the value through the widget instance structures rather than through XtSetValues().

In Motif 2.0 and later, passing the value XmINDETERMINATE is mapped to XmSET. It is therefore not possible to set the XmINDETERMINATE state using XmToggleButtonSetState(). To set a Toggle into an indeterminate state through the convenience functions, call XmToggleButtonSetValue() or XmToggleButtonGadgetSetValue().

See Also

XmToggleButtonGetState(1), XmToggleButtonSetValue(1), XmToggleButton(2), XmToggleButtonGadget(2).

Name

XmToggleButtonSetValue, XmToggleButtonGadgetSetValue – set the value of a ToggleButton.

Synopsis

```
#include <Xm/ToggleB.h>

Boolean XmToggleButtonSetValue (Widget widget, XmToggleButtonState
state, Boolean notify)

#include <Xm/ToggleBG.h>

Boolean XmToggleButtonGadgetSetValue ( Widget                widget,
                                       XmToggleButtonState   state,
                                       Boolean               notify)
```

Inputs

widget	Specifies the ToggleButton or ToggleButtonGadget.
state	Specifies the state of the button.
notify	Specifies whether or not the XmNvalueChangedCallback is called.

Availability

Motif 2.0 and later.

Description

`XmToggleButtonSetValue`() and `XmToggleButtonGadgetSetValue`() are similar to `XmToggleButtonSetState`() and `XmToggleButtonGadgetSetState`(), except that it is possible to set the ToggleButton into an XmINDETERMINATE state, provided that the Toggle is in the correct mode. If the widget has the XmNtoggleMode resource of XmTOGGLE_INDETERMINATE, the routine sets the XmNset resource of the widget to the required state, calls any XmNvalueChangedCallback procedures if *notify* is True, and then returns True. Otherwise, the function returns False.

Usage

`XmToggleButtonSetValue`() and `XmToggleButtonGadgetSetValue`() are convenience routines that set the value of the XmNset resource for a ToggleButton or ToggleButtonGadget which can display an indeterminate state. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

Structures

The XmToggleButtonState type has the following possible values:

XmSET XmUNSET XmINDETERMINATE

See Also

`XmToggleButtonGetState`(1), `XmToggleButtonSetState`(1), `XmToggleButton`(2), `XmToggleButtonGadget`(2).

Name

XmTrackingEvent – allow for modal selection of a component.

Synopsis

#include <Xm/Xm.h>

Widget XmTrackingEvent (Widget *widget*, Cursor *cursor*, Boolean *confine_to*, XEvent **event_return*)

Inputs

widget	Specifies the widget in which the modal interaction occurs.
cursor	Specifies the cursor that is to be used as the pointer.
confine_to	Specifies whether or not the pointer is confined to widget.

Outputs

event_return	Returns the ButtonRelease or KeyRelease event.

Returns

The widget or gadget that contains the pointer or NULL if no widget or gadget contains the pointer.

Availability

Motif 1.2 and later.

Description

`XmTrackingEvent`() grabs the pointer and waits for the user to release BSelect or press and release a key, discarding all of the intervening events. The routine returns the ID of the widget or gadget containing the pointer when BSelect or the key is released and *event_return* contains the release event. If no widget or gadget contains the pointer when the release occurs, the function returns NULL. The modal interaction occurs within the specified *widget*, which is typically a top-level shell. During the interaction, *cursor* is used as the pointer shape. If *confine_to* is True, the pointer is confined to *widget* during the interaction; otherwise the pointer is not confined.

Usage

`XmTrackingEvent`() provides a way to allow a user to select a component. This modal interaction is meant to support a context-sensitive help system, where the user clicks on a widget to obtain more information about it. `XmTrackingEvent`() returns the selected widget, so that a help callback can be invoked to provide the appropriate information.

Example

The following code shows the use of XmTrackingEvent() in a routine that initiates context-sensitive help:

```
Widget toplevel, help_button;
...
XtAddCallback (help_button, XmNactivateCallback, query_for_help, toplevel);
...
void query_for_help (Widget widget, XtPointer client_data, XtPointer call_data)
{
    Cursor                cursor;
    Widget                top, help_widget;
    XmAnyCallbackStruct   cb;
    XtCallbackStatus      hascb;
    XEvent                *event;

    top = (Widget) client_data;
    cursor = XCreateFontCursor (XtDisplay (top), XC_question_arrow);
    help_widget = XmTrackingEvent (top, cursor, True, &event);

    while (help_widget != NULL) {
        hascb = XtHasCallbacks (help_widget, XmNhelpCallback);

        if (hascb == XtCallbackHasSome) {
            cb.reason = XmCR_HELP;
            cb.event = event;
            XtCallCallbacks (help_widget, XmNhelpCallback, (XtPointer)
            &cb);
            help_widget = NULL;
        }
        else
            help_widget = XtParent (help_widget);
    }
}
```

See Also

XmTrackingLocate(1).

Name

XmTrackingLocate – allow for modal selection of a component.

Synopsis

Widget XmTrackingLocate (Widget *widget*, Cursor *cursor*, Boolean *confine_to*)

Inputs

widget	Specifies the widget in which the modal interaction occurs.
cursor	Specifies the cursor that is to be used as the pointer.
confine_to	Specifies whether or not the pointer is confined to widget.

Returns

The widget or gadget that contains the pointer or NULL if no widget or gadget contains the pointer.

Availability

In Motif 1.2, `XmTrackingLocate`() is obsolete. It has been superseded by `XmTrackingEvent`().

Description

`XmTrackingLocate`() grabs the pointer and waits for the user to release BSelect or press and release a key, discarding all of the intervening events. The routine returns the ID of the widget or gadget containing the pointer when BSelect or the key is released. If no widget or gadget contains the pointer when the release occurs, the function returns NULL. The modal interaction occurs within the specified *widget*, which is typically a top-level shell. During the interaction, *cursor* is used as the pointer shape. If *confine_to* is True, the pointer is confined to widget during the interaction; otherwise the pointer is not confined. `XmTrackingLocate`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

`XmTrackingLocate`() provides a way to allow a user to select a component. This modal interaction is meant to support a context-sensitive help system, where the user clicks on a widget to obtain more information about it. `XmTrackingLocate`() returns the selected widget, so that a help callback can be invoked to provide the appropriate information.

See Also

`XmTrackingEvent`(1).

Name

XmTransfer – introduction to the uniform transfer model.

Synopsis

Public Header:
<Xm/Transfer.h>

Functions/Macros:
`XmTransferDone`*()*, `XmTransferSendRequest`(), `XmTransferSetParameters`*()*, `XmTransferStartRequest`(), `XmTransferValue`*()*.

Availability

Motif 2.0 and later.

Description

Motif widgets support several methods of data transfer. Data can be transferred from a widget to the Primary or Secondary selection, the Clipboard, or, through the drag and drop mechanisms, to another widget. Up until Motif 2.0, each of these data transfer operations require a different treatment by the programmer. In Motif 2.0 and later, the Uniform Transfer Model (UTM) makes it possible to perform data transfer using any of the transfer methods using a single programming interface. UTM is designed to allow applications to use common code for all the supported data transfer requirements, and is intentionally written to ease the way in which new transfer targets can be written. Data transfer code written prior to Motif 2.0 will continue to work in newer versions of the toolkit, although all the widgets have been rewritten to internally use the UTM where appropriate.

The UTM is implemented through two new callback resources: *XmNconvertCallback*, and *XmNdestinationCallback*, which are available in the Primitive widget class (and in any derived classes), as well as in the Container, Scale, and DrawingArea widget classes. The programmer provides XmNconvertCallback and XmNdestinationCallback procedures which communicate with one another in order to negotiate the target format in which the data is required. In addition, the programmer provides a transfer procedure which performs the insertion of data in the right format into the destination widget.

An XmNconvertCallback procedure is associated with the source of the data. It is responsible for converting the data, typically the selected items of the source widget, into the format requested by the destination. It may also provide a list of the supported transfer targets requested by a XmNdestinationCallback procedure. The convert procedure transfers data to the destination widget by placing values within the XmConvertCallbackStruct structure passed as a parameter to the callback.

The XmNdestinationCallback procedure is responsible for negotiating the format in which data is required at the destination widget. It may request the set of supported formats in which the source can export the data, although the simplest procedure requests data in a specific target format. In specifying the request to the source of the data, the callback specifies a further transfer procedure which performs the actual insertion of data at the destination. The destination callback communicates with the source by issuing requests using the `XmTransferValue`() routine. If the destination callback requests the full list of supported source targets, the transfer procedure itself decides the best format for the destination, and internally issues a further `XmTransferValue`() call, requesting the data in a specific target format. The programmer will have to provide the logic which determines the best format within the transfer procedure.

Usage

It is not necessary for the programmer to provide XmNconvertCallback and XmNdestinationCallback procedures for all the targets which Motif supports. As part of the UTM, Motif widgets which support the XmNconvertCallback and XmNdestinationCallback resources also have internal routines which enable automatic data transfer of a set of built-in target types. The internal routines are implemented using the XmQTtransfer trait, which has convertProc and destinationProc methods. Where no XmNconvertCallback is supplied by the programmer, the convertProc is invoked to perform the data conversion. Similarly, in the absence of a XmNdestinationCallback, UTM calls the destinationProc. A programmer only needs to implement XmNconvertCallback or XmNdestinationCallback procedures where a new target type is being provided over and above those handled by the widget class default procedures.

Structures

A pointer to the following structure is passed to callbacks on the XmNconvertCallback list:

```
typedef struct {
    int         reason;         /* the reason that the callback is
    invoked     */
    XEvent      *event;         /* points to event that triggered call-
    back        */
    Atom        selection;      /* selection for which conversion is
    requested   */
    Atom        target;         /* the conversion target */
    XtPointer   source_data;    /* selection source information */
    XtPointer   location_data;  /* information about the data to be
    transferred */
    int         flags;          /* input status of the conversion*/
```

```
    XtPointer       parm;           /* parameter data for the target*/
    int             parm_format;    /* format of parameter data*/
    unsigned long   parm_length;    /* number of elements in parameter
    data            */
    Atom            parm_type;      /* the type of the parameter data*/
    int             status;         /* output status of the conversion*/
    XtPointer       value;          /* returned conversion data*/
    Atom            type;           /* type of conversion data returned*/
    int             format;         /* format of the conversion data*/
    unsigned long   length;         /* number of elements in the conver-
    sion data       */
} XmConvertCallbackStruct;
```

selection represents the selection for which conversion is requested. CLIPBOARD, PRIMARY, SECONDARY, or _MOTIF_DROP are the possible values (that is, one of the types of data transfer which the UTM rationalizes).

target represents the required format for the data to transfer, expressed as an Atom.

source_data provides data related to the source of the selection. If *selection* is _MOTIF_DROP, then *source_data* points to a DragContext object, otherwise it is NULL.

location_data specifies where the data to be converted is to be found. If *location_data* is NULL, conversion data is interpreted as the widget's current selection. Otherwise, the interpretation of *location_data* is widget class specific.

flags specifies the current status of the conversion. Possible values of the enumerated type:

```
    XmCONVERTING_NONE       /* unused                              */
    XmCONVERTING_PARTIAL    /* some, but not all, of target data is
    converted               */
    XmCONVERTING_SAME       /* conversion target is source of the
    transfer data           */
    XmCONVERTING_TRANSACT   /* unused                              */
```

parm contains extra data associated with *target*. If *target* is the Atom represented by _MOTIF_CLIPBOARD_TARGETS or _MOTIF_DEFERRED_CLIPBOARD_TARGETS, then *parm* is one of XmCOPY, XmMOVE, or XmLINK. *parm* is an array of data items, the number of such items is specified by *parm_length*, and the type of each item by *parm_format*.

parm_format specifies whether the data within *parm* is represented by a list of char, short, or long quantities. If *parm_format* is 0 (zero), *parm* is NULL. A *parm_format* value of 8 indicates *parm* is logically a list of char, 16 represents a list of short quantities, and 32 is for a list of long values. *parm_format* indicates logical type, and not physical implementation: a *parm_format* of 32 indicates a list of long quantities even if a particular machine has 64-bit longs.

parm_length specifies the number of data items at the *parm* address.

parm_type specifies the type of the *parm* data, expressed as an Atom. The default is XA_INTEGER.

status specifies the status of the conversion. The initial (default) value is XmCONVERT_DEFAULT, which, if unchanged by the callback, invokes any conversion procedures associated with the widget class when the callback finishes. These are the convertProc methods of any XmQTtransfer trait which the widget holds. Any converted data produced by a widget class routine overwrites any data from the callback. If the callback sets *status* to XmCONVERT_MERGE, widget class conversion procedures are invoked, merging any data so produced with conversion data from the callback. A returned *status* of XmCONVERT_REFUSE indicates that the callback terminates the conversion process without completion, and no widget class procedures are to be invoked. XmCONVERT_DONE similarly results in no widget class procedure invocation, except that the callback indicates that conversion has been successfully completed.

value is where the callback places the result of the conversion process. The default is NULL. It is assumed that any *value* specified is dynamically allocated by the programmer, although the programmer must not subsequently free the value: this is performed by the UTM. It is the responsibility of the programmer to ensure that the *type*, *format*, and *length* elements are appropriately set to match any data placed in *value*.

type specifies the logical type of any data returned within *value*, expressed as an Atom.

format specifies whether the data within *value* is represented by a list of char, short, or long quantities. If *format* is 0 (zero), *value* is NULL. A *format* value of 8 indicates *value* is logically a list of char, 16 represents a list of short quantities, and 32 is for a list of long values. *format* indicates logical type, and not physical implementation: a *format* of 32 indicates a list of long values even if they are actually 64 bits.

length specifies the number of data items at the *value* address.

Callbacks on the XmNdestinationCallback list are passed a pointer to the following structure:

```
typedef struct {
    int         reason;           /* reason that the callback is invoked       */
    XEvent      *event;           /* points to event that triggered callback   */
    Atom        selection;        /* the requested selection type, as an Atom  */
    XtEnum      operation;        /* the type of transfer requested            */
    int         flags;            /* whether destination and source are the same
    */
    XtPointer   transfer_id;      /* unique identifier for the request         */
    XtPointer   destination_data; /* information about the destination         */
    XtPointer   location_data;    /* information about the data                */
    Time        time;             /* time when transfer operation started      */
} XmDestinationCallbackStruct;
```

selection specifies, as an Atom, the type of selection for which data transfer is required. CLIPBOARD, PRIMARY, SECONDARY, or _MOTIF_DROP are the possible logical values.

operation indicates the type of data transfer operation requested. The possible values are:

```
XmCOPY      /* copy transfer */
XmMOVE      /* move transfer */
XmLINK      /* link transfer */
XmOTHER     /* information contained within destination_data element */
```

If *operation* is XmOTHER, *destination_data* contains a pointer to an XmDropProcCallbackStruct, which contains an operation element. See XmDropSite for more information concerning the XmDropProcCallbackStruct.

flags indicates whether the source of the transfer data is also the destination. Possible values are:

```
XmCONVERTING_NONE /* destination is not the source of the data */
XmCONVERTING_SAME /* destination is the source of the data     */
```

transfer_id specifies a unique identifier for the current transfer request.

destination_data specifies information about the destination of the transfer operation. If the *selection* is _MOTIF_DROP, then the callback has been invoked by an XmDropProc of the drop site, and *destination_data* contains a pointer to an XmDropProcCallbackStruct. If the *selection* is SECONDARY, *destination_data* is an Atom representing a target type into which the selection owner suggests the transfer should be converted. Otherwise, *destination_data* is NULL.

location_data determines where the data is to be transferred. The interpretation varies between the widget classes. In the Container widget, the value of *location_data* is a pointer to an XPoint structure, containing the x and y coordinates of the transfer location. If *location_data* is NULL, data is to be transferred to the current cursor location for the widget.

time is the server time when the transfer operation was initiated.

Transfer procedures are passed a pointer to the following structure: the interpretation of the elements is the same as those in the callbacks described above.

```
typedef struct {
    int             reason;         /* reason that the callback is invoked
    */
    XEvent          *event;         /* points to event that triggered callback
    */
    Atom            selection;      /* the requested selection type, as an Atom
    */
    Atom            target;         /* the conversion target
    */
    Atom            type;           /* type of conversion data returned
    */
    XtPointer       transfer_id;    /* unique identifier for the request
    */
    int             flags;          /* whether destination and source are same
    */
    int             remaining;      /* number transfers remaining for transfer_id
    */
    XtPointer       value;          /* returned conversion data
    */
    unsigned long   length;         /* number of elements in conversion data
    */
    int             format;         /* format of the conversion data
    */
} XmSelectionCallbackStruct;
```

Example

The following is a specimen XmNdestinationCallback which simply requests from the source the list of export targets, and which specifies a transfer procedure for handling the data import.

```
void destination_handler (Widget w, XtPointer client_data, XtPointer call_data)
{
```

```
    XmDestinationCallbackStruct *dptr = (XmDestinationCallbackStruct *)
    call_data;
    Atom TARGETS = XmInternAtom (XtDisplay (w), "TARGETS", False);

    /* transfer procedure will issue a subsequent request */
    /* for data in a specific target format. it receives */
    /* the list of supported targets from the source.    */
    XmTransferValue (dptr->transfer_id, TARGETS, (XtCallbackProc)
    transfer_procedure,
                        NULL, XtLastTimestampProcessed (XtDisplay (w))[1]);
}
```

The following specimen XmNconvertCallback procedure exports the selected text within a Text widget: although the convertProc method associated with the Text's XmQTtransfer trait performs this task in a modified form, the code does outline the basic structure required.

```
void convert_callback (Widget w, XtPointer client_data, XtPointer call_data)
{
    XmConvertCallbackStruct    *cptr = (XmConvertCallbackStruct *)
    call_data;
    Atom                       TARGETS, CB_TARGETS,
    SELECTED_TEXT;
    Atom                       targets[1];
    Display                    *display = XtDisplay (w);

    TARGETS = XmInternAtom (display, "TARGETS", False);
    CB_TARGETS    = XmInternAtom (display,
    "_MOTIF_CLIPBOARD_TARGETS", False);
    SELECTED_TEXT = XmInternAtom (display, "SELECTED_TEXT",
    False);

    /* if the destination has requested the list of supported targets */
    /* this is returned in the callback data                          */
    if ((cptr->target == TARGETS) || (cptr->target == CB_TARGETS)) {
        targets[0]   = SELECTED_TEXT;
        cptr->type   = XA_ATOM;
        cptr->value = (XtPointer) targets;
        cptr->length = 1;
        cptr->format = 32;
        /* merge the data with the list of targets supported by */
        /* the convertProc method of the XmQTtransfer trait */
```

1.Erroneously given as XtLastTimestampProcessed() in 2nd edition.

```
        cptr->status = XmCONVERT_MERGE;
    }
    else {
        if (cptr->target == SELECTED_TEXT) {
            char *selection = XmTextGetSelection (w);
            /* destination has requested the new target */
            cptr->value = selection;
            /* exported target is the requested target */
            cptr->type   = cptr->target;
            cptr->format = 8;
            cptr->length = (selection ? strlen (selection) : 0);
            /* conversion complete */
            cptr->status = XmCONVERT_DONE;
        }
        else {
            /* target is one this procedure is not handling        */
            /* result is either XmCONVERT_MERGE or
            XmCONVERT_DEFAULT */
            /* depending on whether we throw away results from any   */
            /* other convert callback we have registered.   */
            /* the default is XmCONVERT_DEFAULT */
            return XmCONVERT_MERGE;[1]
        }
    }
    return XmCONVERT_MERGE;[2]
}
```

The following is a specimen transfer procedure, which is registered by a XmNdestinationCallback using XmTransferValue():

```
void transfer_procedure (Widget w, XtPointer client_data, XtPointer call_data)
{
    XmSelectionCallbackStruct    *sptr = (XmSelectionCallbackStruct *)
    call_data;
    Atom                         TARGETS, CB_TARGETS,
    SELECTED_TEXT;
    Display                      *display = XtDisplay (w);
```

1.The 2nd edition gave XmCONVERT_DEFAULT as the return value here. Since certain focus operations built into the toolkit use the Uniform Transfer Model as mechanism, you need to inherit these, so XmCONVERT_MERGE is the better value. Apologies.

2.As above.

```
Atom                    *targets, choice;
int                     i;

choice = (Atom) 0;
TARGETS = XmInternAtom (display, "TARGETS", False);
CB_TARGETS = XmInternAtom (display,
"_MOTIF_CLIPBOARD_TARGETS", False);    SELECTED_TEXT =
XmInternAtom (display, "SELECTED_TEXT", False);

if (((sptr->target == TARGETS) || (sptr->target == CB_TARGETS)) &&
    (sptr->type == XA_ATOM)) {
    /* destination callback requested list of targets from the source */
    /* the source convertCallback returns the list. We now choose... */
    targets = (Atom *) sptr->value;

    for (i = 0; i < sptr->length; i++) {
        if (targets[i] == SELECTED_TEXT) {
        /* the source exports selected text. lets pick this one... */
        choice = targets[i];
        }
    }

    /* There's no selection we like... */
    if (choice == (Atom) 0) {
        XmTransferDone (sptr->transfer_id,
        XmTRANSFER_DONE_FAIL);
        return;
    }

    /* now go back to source and ask for the data in format of choice */
    /* might as well use ourself again as transfer procedure...      */
    XmTransferValue (sptr->transfer_id, choice, /* Preferred
    SELECTED_TEXT target */
                        transfer_procedure, NULL, XtLastTimestampProc-
                        essed (display)[1]);
}
else if (sptr->target == SELECTED_TEXT) {
    /* insert the selected text at our own insertion point */
    XmTextPosition pos = XmTextGetInsertionPosition (w);
    XmTextInsert (w, pos, (char *) sptr->value);
    /* all done */
```

1.Erroneously given as XtLastTimestampProcessed() in 2nd edition.

```
            XmTransferDone (sptr->transfer_id,
            XmTRANSFER_DONE_SUCCEED);
        }
    }
```

See Also

XmTransferDone(1), XmTransferSendRequest(1), XmTransferSetParameters(1), XmTransferStartRequest(1), XmTransferValue(1), XmContainer(2), XmDrawingArea(2), XmPrimitive(2), XmScale(2).

Name

XmTransferDone – complete a data transfer operation.

Synopsis

#include <Xm/Transfer.h>

void XmTransferDone (XtPointer *transfer_id*, XmTransferStatus *status*)

Inputs

transfer_id	Specifies a unique identifier for the transfer operation.
status	Specifies the completion status of the transfer.

Availability

Motif 2.0 and later.

Description

Under the Uniform Transfer Model, `XmTransferDone`() completes a data transfer operation. The procedure is called from destination callbacks or transfer procedures in order to signal the end of data transfer back to the source of the data.

transfer_id uniquely identifies a transfer operation, and the value is supplied either from the *transfer_id* element of a XmDestinationCallbackStruct passed to the destination callback, or from the *transfer_id* element of a XmSelectionCallbackStruct passed to a transfer procedure. *status* is set to indicate the status of the current transfer operation, which is notified back to the selection owner.

status is one of XmTRANSFER_DONE_FAIL, XmTRANSFER_DONE_SUCCEED, XmTRANSFER_DONE_CONTINUE, or XmTRANSFER_DONE_DEFAULT. The *status* XmTRANSFER_DONE_DEFAULT ignores all remaining queued transfer operations which may have been initiated within the destination callbacks and invokes the widget class default transfer procedures. That is, any unprocessed multiple batched requests created between `XmTransferStartRequest`() and `XmTransferSendRequest`()[1] calls are skipped. If *status* is XmTRANSFER_DONE_FAIL, the XmNtransferStatus of the current DropTransfer object is set to XmTRANSFER_FAILURE. XmTRANSFER_DONE_SUCCEED and XmTRANSFER_DONE_CONTINUE are similar, except that with XmTRANSFER_DONE_CONTINUE the owner of the selection is not notified if the target is _MOTIF_SNAPSHOT.

1.Erroneously given as XmTransferEndRequest() in 2nd edition.

Usage

The Uniform Transfer Model (UTM) enhances the Motif 1.2 data transfer mechanisms by providing a standard interface through which Drag and Drop, Primary and Secondary selection, and Clipboard data transfer is achieved both to and from a widget. The implementation of the UTM is through XmNconvertCallback and XmNdestinationCallback resource procedures. A convert callback is associated with the source of the data, and it is responsible for exporting data in the format required by the destination widget.

The destination callback is responsible for requesting data from the source in the format which it requires, and it calls the function `XmTransferValue`() to do this. The destination callback typically does not import the data directly, but specifies a transfer procedure to perform the insertion of data at the destination widget. The transfer procedure is specified by passing a routine as a parameter to the `XmTransferValue`() call. When the transfer is finished, either because it is completed or because it is aborted due to an error, the transfer procedure calls `XmTransferDone`() to return the status to the source.

Structures

The XmTransferStatus type has the following possible values:

XmTRANSFER_DONE_CONTINUE
XmTRANSFER_DONE_DEFAULT
XmTRANSFER_DONE_FAIL
XmTRANSFER_DONE_SUCCEED

Example

The following specimen transfer procedure calls `XmTransferDone`() to indicate the status of the data drop:

```
void transfer_procedure (Widget w, XtPointer client_data, XtPointer call_data)
{
    XmSelectionCallbackStruct    *sptr = (XmSelectionCallbackStruct *)
    call_data;
    Atom                         TARGETS, CB_TARGETS,
    IMPORT_FORMAT;
    Display                      *display = XtDisplay (w);
    Atom                         *targets, choice;
    int                          i;

    TARGETS = XmInternAtom (display, "TARGETS", False);
    CB_TARGETS = XmInternAtom (display,
    "_MOTIF_CLIPBOARD_TARGETS", False);
```

```
IMPORT_FORMAT = XmInternAtom (display, "IMPORT_FORMAT",
False);

if (((sptr->target == TARGETS) || (sptr->target == CB_TARGETS)) &&
    (sptr->type == XA_ATOM)) {
    /* destination callback requested list of targets from the source */
    /* the source convertCallback returns the list. We now choose... */
    targets = (Atom *) sptr->value;
    choice = (Atom) 0;

    for (i = 0; i < sptr->length; i++) {
        if (targets[i] == IMPORT_FORMAT) {
            /* the source exports our required target... */
            choice = targets[i];
        }
    }

    if (choice == (Atom) 0) {
        /* source does not export what we require        */
        /* assume destinationProc in the XmQTtransferTrait */
        /* does not either...                            */
        XmTransferDone (sptr->transfer_id,
        XmTRANSFER_DONE_FAIL);
        return;
    }

    /* now go back to source and ask for the data in format of choice */
    /* might as well use ourself again as transfer procedure...      */
    XmTransferValue (sptr->transfer_id, choice, /* IMPORT_FORMAT */
                     transfer_procedure, NULL, XtLastTimestampProc-
                     essed (display)[1]);
}
else if (sptr->target == IMPORT_FORMAT) {
    /* perform whatever is required to import sptr->value */
    ...
    /* all done */
    XmTransferDone (sptr->transfer_id,
    XmTRANSFER_DONE_SUCCEED);
}
else {
    /* wrong export target */
```

1.Erroneously given as XtLastTimestampProcessed() in 2nd edition.

```
            XmTransferDone (sptr->transfer_id, XmTRANSFER_DONE_FAIL);
        }
    }
```

See Also

XmTransferSendRequest(1), XmTransferSetParameters(1), XmTransferStartRequest(1), XmTransferValue(1), XmTransfer(1), XmDropTransfer(1).

Name

XmTransferSendRequest – send a multiple transfer request.

Synopsis

#include <Xm/Transfer.h>

void XmTransferSendRequest (XtPointer *transfer_id*, Time *time*)

Inputs

transfer_id	Specifies a unique identifier for the transfer operation.
time	Specifies the time of the transfer.

Availability

Motif 2.0 and later.

Description

In the Uniform Transfer Model, `XmTransferSendRequest`() marks the end of a series of transfer requests started by `XmTransferStartRequest`(). *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallbackStruct passed to a destination callback or transfer procedure respectively. *time* specifies the time of the XEvent which initiated the data transfer. `XtLastTimestampProcessed`() is the simplest method of specifying the time value.

Usage

The Uniform Transfer Model (UTM) enhances the Motif 1.2 data transfer mechanisms by providing a standard interface through which Drag and Drop, Primary and Secondary selection, and Clipboard data transfer is achieved both to and from a widget. The implementation of the UTM is through XmNconvertCallback and XmNdestinationCallback resource procedures. The destination callback is responsible for requesting data from the source in the format which it requires, and it calls the function `XmTransferValue`() to do this. A set of data transfer requests can be queued by wrapping the series of `XmTransferValue`() calls within `XmTransferStartRequest`() and `XmTransferSendRequest`() calls.

See Also

`XmTransferDone`(1), `XmTransferSetParameters`(1), `XmTransferStartRequest`(1), `XmTransferValue`(1), `XmTransfer`(1).

Name

XmTransferSetParameters – set parameters for next transfer

Synopsis

```
#include <Xm/Transfer.h>

void XmTransferSetParameters ( XtPointer      transfer_id,
                               XtPointer      parm,
                               int            parm_format,
                               unsigned long  parm_length,
                               Atom           parm_type)
```

Inputs

transfer_id	Specifies a unique identifier for the transfer operation.
parm	Specifies parameters to be passed to conversion routines.
parm_format	Specifies the format of data in the parm argument.
parm_length	Specifies the number of elements within the parm data.
parm_type	Specifies the type of parm.

Availability

Motif 2.0 and later.

Description

In the Uniform Transfer Model, `XmTransferSetParameters`() defines parameter data for a subsequent `XmTransferValue`() call. *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallback-Struct passed to a destination callback or transfer procedure respectively.

parm specifies parameter data to be passed to the conversion function, and the XmNconvertCallback procedures, of the source widget which owns the selection. *parm_format* specifies whether the data within parm consists of 8, 16, or 32 bit quantities. *parm_length* specifies the number of elements, of size determined by *parm_format*, which are at the address *parm*. *parm_type* specifies the logical type of *parm*, and is application specific. Neither Motif, the X toolkit, nor the X library interpret *parm_type* in any manner.

Usage

The Uniform Transfer Model enhances the Motif 1.2 data transfer mechanisms by providing a standard interface by which the source and destination of the data transfer can communicate. The programmer provides XmNdestinationCallback procedures which issue the request to transfer data from the source of the transfer by calling XmTransferValue(). Any parameterized data for the XmTransferValue() procedure is established through a prior XmTransferSetParameters() call. XmTransferSetParameters() is a convenience function which maps simply onto a X Toolkit Intrinsics XtSetSelectionParameters() call.

See Also

XmTransferDone(1), XmTransferSendRequest(1), XmTransferStartRequest(1), XmTransferValue(1), XmTransfer(1).

Name

XmTransferStartRequest – initiate a multiple data transfer request

Synopsis

#include <Xm/Transfer.h>

void XmTransferStartRequest (XtPointer *transfer_id*)

Inputs

transfer_id Specifies a unique identifier for the current data transfer operation.

Availability

Motif 2.0 and later.

Description

`XmTransferStartRequest`() initiates the start of a series of transfer requests. *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallbackStruct passed to a destination callback or transfer procedure respectively.

Usage

In Motif 2.0 and later, the Uniform Transfer Model enhances the Motif 1.2 data transfer mechanisms by providing a standard interface by which the source and destination of the data transfer can communicate. A set of data transfer requests can be queued by wrapping the series of requests within `XmTransferStartRequest`() and `XmTransferSendRequest`() calls. The procedure `XmTransferValue`() provides the data transfer requests in the queue.

See Also

`XmTransferDone`(1), `XmTransferSendRequest`(1), `XmTransferSetParameters`(1), `XmTransferValue`(1), `XmTransfer`(1).

Name

XmTransferValue – transfer data to a destination

Synopsis

```
#include <Xm/Transfer.h>

void XmTransferValue ( XtPointer      transfer_id,
                       Atom           target,
                       XtCallbackProc callback,
                       XtPointer      client_data,
                       Time           time)
```

Inputs

transfer_id	Specifies a unique identifier for the current data transfer operation.
target	Specifies the target to which the selection is to be converted.
callback	Specifies a transfer procedure to be called when the selection has been converted by the source.
client_data	Specifies application data to be passed to callback.
time	Specifies the time of the transfer.

Availability

Motif 2.0 and later.

Description

The Uniform Transfer Model (UTM) enhances the Motif 1.2 data transfer mechanisms by providing a standard interface through which Drag and Drop, Primary and Secondary selection, and Clipboard data transfer is achieved both to and from a widget. The implementation of the UTM is through XmNconvertCallback and XmNdestinationCallback resource procedures. A convert callback is associated with the source of the data, and it is responsible for exporting data in the format required by the destination. The destination callback is responsible for requesting data from the source in the format which it requires, and it calls the function `XmTransferValue`() to do this. The destination callback itself does not typically insert the transferred data into the destination widget: it specifies a transfer procedure, which performs the import, as a parameter to the `XmTransferValue`() call.

`XmTransferValue`() arranges to transfer data from the source of transfer data to the destination. *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallbackStruct passed to a destination or transfer callback respectively.

target specifies the selection which is to be converted and transferred. If *target* is _MOTIF_DROP, the function invokes XmDropTransferStart() with internal transfer procedures to perform the data transfer. Otherwise, the data is extracted from the selection using XtGetSelectionValue().

callback is a transfer procedure, which is an application procedure that is called when the data is converted and available from the source. *client_data* is any data which the programmer wants to be passed to the *callback*. The *callback* is invoked with three parameters: the destination widget, the application *client_data*, and a pointer to a XmSelectionCallbackStruct.

time specifies the time of the XEvent which initiated the data transfer. XtLastTimestampProcessed() is the simplest method of specifying the time value.

Usage

XmTransferValue() is called from destination callbacks or transfer procedures to effect the actual transfer of data from the source, whether that be the clipboard or a widget. The programmer-defined callback replaces the XmNtransferProc added to a DropTransfer object, which the Uniform Transfer Model internally encapsulates and hides.

See Also

XmTransferDone(1), XmTransferSendRequest(1), XmTransferSetParameters(1), XmTransferStartRequest(1), XmTransfer(1), XmDropTransfer(2).

Name

XmTranslateKey – convert a keycode to a keysym using the default translator.

Synopsis

```
#include <Xm/Xm.h>

void XmTranslateKey ( Display     *display,
                      KeyCode     keycode,
                      Modifiers   modifiers,
                      Modifiers   *modifiers_return,
                      KeySym      *keysym_return)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
keycode Specifies the keycode that is translated.
modifiers Specifies the modifier keys that are applied to the keycode.

Outputs

modifiers_return Returns the modifiers used by the key translator to generate the keysym.
keysym_return Returns the resulting keysym.

Availability

Motif 1.2 and later.

Description

`XmTranslateKey`() is the default XtKeyProc translation procedure used by Motif applications. The routine takes a keycode and modifiers and returns the corresponding osf keysym.

Usage

The Motif toolkit uses a mechanism called *virtual bindings* to map one set of keysyms to another set. This mapping permits widgets and applications to use one set of keysyms in translation tables; applications and users can then customize the keysyms used in the translations based on the particular keyboard that is being used. Keysyms that can be used in this way are called *osf keysyms*. Motif maintains a mapping between the osf keysyms and the actual keysyms that represent keys on a particular keyboard. See the introduction to Section 2, *Motif and Xt Widget and Classes*, for more information about the mapping of osf keysyms to actual keysyms.

XmTranslateKey() is used by the X Toolkit during event processing to translate the keycode of an event to the appropriate osf keysym if there is a mapping for the keysym. The event is then dispatched to the appropriate action routine if there is a translation for the osf keysym.

If you need to provide a new translator with expanded functionality, you can call XmTranslateKey() to get the default translation. Use XtSetKeyTranslator() to register a new key translator. To reinstall the default behavior, you can call XtSetKeyTranslator() with XmTranslateKey() as the proc argument.

See Also

xmbind(4).

Name

XmUninstallImage – remove an image from the image cache.

Synopsis

Boolean XmUninstallImage (XImage *_image_)

Inputs

image Specifies the image structure to be removed.

Returns

True on success or False if image is NULL or it cannot be found.

Description

`XmUninstallImage`() removes the specified _image_ from the image cache. The routine returns True if it is successful. It returns False if image is NULL or if _image_ is not found in the image cache.

Usage

`XmUninstallImage`() removes an image from the image cache. Once an image is uninstalled, it cannot be referenced again and a new image can be installed with the same name. If you have created any pixmaps that use the image, they are not affected by the image being uninstalled, since they are based on image data, not the image itself. After an image has been uninstalled, you can safely free the image.

See Also

`XmDestroyPixmap`(1), `XmGetPixmap`(1), `XmInstallImage`(1).

Name

XmUpdateDisplay – update the display.

Synopsis

void XmUpdateDisplay (Widget *widget*)

Inputs

widget Specifies any widget.

Description

`XmUpdateDisplay`() causes all pending exposure events to be processed immediately, instead of having them remain in the queue until all of the callbacks have been invoked.

Usage

`XmUpdateDisplay`() provides applications with a way to force an visual update of the display. Because callbacks are invoked before normal exposure processing occurs, when a menu or a dialog box is unposted, the display is not updated until all of the callbacks have been called. This routine is useful whenever a time-consuming action might delay the redrawing of the windows on the display.

See Also

`XmDisplay`(2).

Name

XmVaCreateSimpleCheckBox – create a CheckBox compound object.

Synopsis

```
Widget XmVaCreateSimpleCheckBox ( Widget          parent,
                                  String          name,
                                  XtCallbackProc  callback,
                                  ...,
                                  NULL)
```

Inputs

parent Specifies the widget ID of the parent of the new widget.
name Specifies the string name of the new widget for resource lookup.
callback Specifies the callback procedure that is called when the value of a button changes.
..., NULL A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleCheckBox`() is a RowColumn convenience routine that creates a CheckBox with ToggleButtonGadgets as its children. This routine is similar to `XmCreateSimpleCheckBox`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the CheckBox. The *callback* argument specifies the callback routine that is added to the XmNvalueChangedCallback of each ToggleButtonGadget child of the CheckBox. When the *callback* is invoked, the button number of the button whose value has changed is passed to the *callback* in the *client_data* parameter.

The name of each ToggleButtonGadget child is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the CheckBox. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaCHECKBUTTON, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaCHECKBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a ToggleButtonGadget child of the CheckBox and its associated resources. (As of Motif 1.2, all but the label argument are ignored.)

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleCheckBox`() as in the following example:

```
Widget      toplevel, check_box;
XmString    normal, bold, italic;

normal = XmStringCreateLocalized ("normal");
bold = XmStringCreateLocalized ("bold");
italic = XmStringCreateLocalized ("italic");
check_box = XmVaCreateSimpleCheckBox (toplevel, "check_box", toggled,
                                      XmVaCHECKBUTTON, normal,
                                      NULL, NULL, NULL,
                                      XmVaCHECKBUTTON, bold,
                                      NULL, NULL, NULL,
                                      XmVaCHECKBUTTON, italic,
                                      NULL, NULL, NULL,
                                      NULL);
XmStringFree (normal);
XmStringFree (bold);
XmStringFree (italic);
```

See Also

`XmCheckBox`(2), `XmRowColumn`(2), `XmToggleButtonGadget`(2).

Name

XmVaCreateSimpleMenuBar – create a MenuBar compound object.

Synopsis

Widget XmVaCreateSimpleMenuBar (Widget *parent*, char **name*,..., NULL)

Inputs

parent	Specifies the widget ID of the parent of the new widget.
name	Specifies the string name of the new widget for resource lookup.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleMenuBar`() is a RowColumn convenience routine that creates a MenuBar with CascadeButtonGadgets as its children. This routine is similar to `XmCreateSimpleMenuBar`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the MenuBar.

The name of each CascadeButtonGadget is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the MenuBar. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaCASCADEBUTTON, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget child of the MenuBar and its associated resources.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by

one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleMenuBar`() as in the following example:

```
Widget      top, mainw, menubar, fmenu, emenu;
XmString    file, edit, new, quit, cut, clear, copy, paste;

file = XmStringCreateLocalized ("File");
edit = XmStringCreateLocalized ("Edit");
menubar = XmVaCreateSimpleMenuBar (mainw, "menubar",
                                   XmVaCASCADEBUTTON,
                                   file,'F',
                                   XmVaCASCADEBUTTON,
                                   edit,'E',
                                   NULL);
XmStringFree (file);
XmStringFree (edit);

new = XmStringCreateLocalized ("New");
quit = XmStringCreateLocalized ("Quit");
fmenu = XmVaCreateSimplePulldownMenu (menubar, "file_menu", 0, file_cb,
                                      XmVaPUSHBUTTON,
                                      new,'N', NULL, NULL,
                                      XmVaSEPARATOR,
                                      XmVaPUSHBUTTON,
                                      quit,'Q', NULL, NULL,
                                      NULL);
XmStringFree (new);
XmStringFree (quit);

cut = XmStringCreateLocalized ("Cut");
copy = XmStringCreateLocalized ("Copy");
clear = XmStringCreateLocalized ("Clear");
paste = XmStringCreateLocalized ("Paste");
emenu = XmVaCreateSimplePulldownMenu (menubar, "edit_menu", 0,
cut_paste,
                                      XmVaPUSHBUTTON,
                                      cut,'C', NULL, NULL,
                                      XmVaPUSHBUTTON,
                                      copy,'o', NULL, NULL,
                                      XmVaPUSHBUTTON,
                                      paste,'P', NULL, NULL,
```

```
                                        XmVaSEPARATOR,
                                        XmVaPUSHBUTTON,
                                        clear,'l', NULL, NULL,
                                        NULL);
    XmStringFree (cut);
    XmStringFree (clear);
    XmStringFree (copy);
    XmStringFree (paste);
```

See Also

`XmCascadeButtonGadget`(2), `XmMenuBar`(2), `XmRowColumn`(2).

Name

XmVaCreateSimpleOptionMenu – create an OptionMenu compound object.

Synopsis

```
Widget XmVaCreateSimpleOptionMenu (  Widget          parent,
                                     String          name,
                                     XmString        option_label,
                                     KeySym
option_mnemonic,
                                     int             button_set,
                                     XtCallbackProc  callback,
                                     ...,
                                     NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the new widget.
name	Specifies the string name of the new widget for resource lookup.
option_label	Specifies the label used for the OptionMenu.
option_mnemonic	Specifies the mnemonic character associated with the OptionMenu.
button_set	Specifies the initial setting of the OptionMenu.
callback	Specifies the callback procedure that is called when a button is activated.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleOptionMenu`() is a RowColumn convenience routine that creates an OptionMenu along with its submenu of CascadeButtonGadget and/or PushButtonGadget children. This routine is similar to `XmCreateSimpleOptionMenu`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the OptionMenu.

The *option_label*, *option_mnemonic*, and *button_set* arguments are used to set the XmNlabelString, XmNmnemonic, and XmNmenuHistory resources of the RowColumn respectively. The *button_set* parameter specifies the *n*th button child of the OptionMenu, where the first button is button 0 (zero); the XmNmenuHistory resource is set to the actual widget. The *callback* argument specifies the

callback routine that is added to the XmNactivateCallback of each CascadeButtonGadget and PushButtonGadget child in the submenu of the OptionMenu. When the *callback* is invoked, the button number of the button whose value has changed is passed to the *callback* in the *client_data* parameter.

The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the submenu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 (zero) to 1 less than the number of separators in the submenu. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaPUSHBUTTON, XmVaCASCADEBUTTON, XmVaSEPARATOR, XmVaDOUBLE_SEPARATOR, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaPUSHBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a PushButtonGadget in the pulldown submenu of the OptionMenu and its associated resources. If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget in the pulldown submenu of the OptionMenu and its associated resources. If the first argument in a group is XmVaSEPARATOR or XmVaDOUBLE_SEPARATOR, it is not followed by any arguments. These groups specify SeparatorGadgets in the pulldown submenu of the OptionMenu.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleOptionMenu`() as in the following example:

```
Widget      rc, option_menu;
XmString    draw_shape, line, square, circle;
```

```
draw_shape = XmStringCreateLocalized ("Draw Mode:");
line = XmStringCreateLocalized ("Line");
square = XmStringCreateLocalized ("Square");
circle = XmStringCreateLocalized ("Circle");
option_menu = XmVaCreateSimpleOptionMenu (rc, "option_menu",
draw_shape, 'D', 0, option_cb,
                                          XmVaPUSHBUTTON, line,
                                          'L', NULL, NULL,
                                          XmVaPUSHBUTTON,
                                          square, 'S', NULL, NULL,
                                          XmVaPUSHBUTTON, circle,
                                          'C', NULL, NULL,
                                          NULL);
XmStringFree (line);
XmStringFree (square);
XmStringFree (circle);
XmStringFree (draw_shape);
```

See Also

`XmOptionButtonGadget`(1), `XmOptionLabelGadget`(1), `XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmOptionMenu`(2), `XmPushButtonGadget`(2), `XmRowColumn`(2), `XmSeparatorGadget`(2).

Name

XmVaCreateSimplePopupMenu – create a PopupMenu compound object as the child of a MenuShell.

Synopsis

```
Widget XmVaCreateSimplePopupMenu ( Widget          parent,
                                   String          name,
                                   XtCallbackProc  callback,
                                   ...,
                                   NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the MenuShell.
name	Specifies the string name of the new widget for resource lookup.
callback	Specifies the callback procedure that is called when a button is activated or its value changes.
...,NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimplePopupMenu`() is a RowColumn convenience routine that creates a PopupMenu along with its button children. The routine creates the PopupMenu as a child of a MenuShell. This routine is similar to `XmCreateSimplePopupMenu`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the PopupMenu. The *callback* argument specifies the callback routine that is added to the XmNactivateCallback of each CascadeButtonGadget and PushButtonGadget child and the XmNvalueChangedCallback of each ToggleButtonGadget child in the PopupMenu. When the callback is invoked, the button number of the button whose value has changed is passed to the callback in the *client_data* parameter.

The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the menu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 (zero) to 1 less than the number of separators in the menu. The name of each title is label_*n*, where *n* is the number of the title, ranging from 0 (zero) to 1 less than the number of titles in the menu. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaPUSHBUTTON, XmVaCASCADEBUTTON, XmVaRADIOBUTTON, XmVaCHECKBUTTON, XmVaTITLE, XmVaSEPARATOR, XmVaDOUBLE_SEPARATOR, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaPUSHBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a PushButtonGadget child of the PopupMenu and its associated resources. If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget child of the PopupMenu and its associated resources. If the first argument in a group is XmVaRADIOBUTTON or XmVaCHECKBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. These groups specify ToggleButtonGadget children of the PopupMenu and their associated resources.

If the first argument is XmVaTITLE, it is followed by a title argument. This group specifies a LabelGadget title in the PopupMenu and its associated resource. If the first argument in a group is XmVaSEPARATOR or XmVaDOUBLE_SEPARATOR, it is not followed by any arguments. These groups specify SeparatorGadgets in the PopupMenu.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimplePopupMenu`() as in the following example:

```
Widget        drawing_a, popup_menu;
XmString      line, square, circle, quit, quit_acc;

line = XmStringCreateLocalized ("Line");
square = XmStringCreateLocalized ("Square");
circle = XmStringCreateLocalized ("Circle");
```

```
quit = XmStringCreateLocalized ("Quit");
quit_acc = XmStringCreateLocalized ("Ctrl-C");
popup_menu = XmVaCreateSimplePopupMenu (drawing_a, "popup",
popup_cb,
                                        XmVaPUSHBUTTON, line, NULL, NULL,
                                        NULL,
                                        XmVaPUSHBUTTON, square, NULL,
                                        NULL, NULL,
                                        XmVaPUSHBUTTON, circle, NULL,
                                        NULL, NULL,
                                        XmVaSEPARATOR,
                                        XmVaPUSHBUTTON, quit, NULL,
                                        "Ctrl<Key>c", quit_acc,
                                        NULL);
XmStringFree (line);
XmStringFree (square);
XmStringFree (circle);
XmStringFree (quit);
XmStringFree (quit_acc);
```

See Also

XmCascadeButtonGadget(2), XmLabelGadget(2), XmMenuShell(2), XmPopupMenu(2), XmPushButtonGadget(2), XmRowColumn(2), XmSeparatorGadget(2), XmToggleButtonGadget(2).

Name

XmVaCreateSimplePulldownMenu – create a PulldownMenu compound object as the child of a MenuShell.

Synopsis

```
Widget XmVaCreateSimplePulldownMenu (  Widget            parent,
                                       String            name,
                                       int
post_from_button,
                                       XtCallbackProc    callback,
                                       ...,
                                       NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the MenuShell.
name	Specifies the string name of the new widget for resource lookup.
post_from_button	Specifies the CascadeButton or CascadeButtonGadget in the parent widget to which the menu is attached.
callback	Specifies the callback procedure that is called when a button is activated or its value changes.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimplePulldownMenu`() is a RowColumn convenience routine that creates a PulldownMenu along with its button children. The routine creates the PulldownMenu as a child of a MenuShell. This routine is similar to `XmCreateSimplePulldownMenu`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the PulldownMenu.

The *post_from_button* parameter specifies the CascadeButton or CascadeButtonGadget to which the PulldownMenu is attached as a submenu. The argument specifies the nth CascadeButton or CascadeButtonGadget, where the first button is button 0 (zero). The *callback* argument specifies the callback routine that is added to the XmNactivateCallback of each CascadeButtonGadget and PushButtonGadget child and the XmNvalueChangedCallback of each ToggleButtonGadget child in the PulldownMenu. When the *callback* is invoked, the button

number of the button whose value has changed is passed to the callback in the *client_data* parameter.

The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the menu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 (zero) to 1 less than the number of separators in the menu. The name of each title is label_*n*, where *n* is the number of the title, ranging from 0 (zero) to 1 less than the number of titles in the menu. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaPUSHBUTTON, XmVaCASCADEBUTTON, XmVaRADIOBUTTON, XmVaCHECKBUTTON, XmVaTITLE, XmVaSEPARATOR, XmVaDOUBLE_SEPARATOR, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaPUSHBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a PushButtonGadget child of the PulldownMenu and its associated resources. If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget child of the PulldownMenu and its associated resources. If the first argument in a group is XmVaRADIOBUTTON or XmVaCHECKBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. These groups specify ToggleButtonGadget children of the PulldownMenu and their associated resources.

If the first argument is XmVaTITLE, it is followed by a title argument. This group specifies a LabelGadget title in the PulldownMenu and its associated resource. If the first argument in a group is XmVaSEPARATOR or XmVaDOUBLE_SEPARATOR, it is not followed by any arguments. These groups specify SeparatorGadgets in the PulldownMenu.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by

one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimplePulldownMenu`() as in the following example:

```
Widget      top, mainw, menubar, fmenu, emenu;
XmString    file, edit, new, quit, cut, clear, copy, paste;

file = XmStringCreateLocalized ("File");
edit = XmStringCreateLocalized ("Edit");
menubar = XmVaCreateSimpleMenuBar (mainw, "menubar",
                                   XmVaCASCADEBUTTON, file, 'F',
                                   XmVaCASCADEBUTTON, edit,
                                   'E',
                                   NULL);
XmStringFree (file);
XmStringFree (edit);

new = XmStringCreateLocalized ("New");
quit = XmStringCreateLocalized ("Quit");
fmenu = XmVaCreateSimplePulldownMenu (menubar, "file_menu", 0, file_cb,
                                      XmVaPUSHBUTTON, new,
                                      'N', NULL, NULL,
                                      XmVaSEPARATOR,
                                      XmVaPUSHBUTTON, quit,
                                      'Q', NULL, NULL,
                                      NULL);
XmStringFree (new);
XmStringFree (quit);

cut = XmStringCreateLocalized ("Cut");
copy = XmStringCreateLocalized ("Copy");
clear = XmStringCreateLocalized ("Clear");
paste = XmStringCreateLocalized ("Paste");
emenu = XmVaCreateSimplePulldownMenu (menubar, "edit_menu", 1,
cut_paste,
                                      XmVaPUSHBUTTON, cut, 'C',
                                      NULL, NULL,
                                      XmVaPUSHBUTTON, copy,
                                      'o', NULL, NULL,
                                      XmVaPUSHBUTTON, paste,
                                      'P', NULL, NULL,
```

```
                                        XmVaSEPARATOR,
                                        XmVaPUSHBUTTON, clear, 'l',
                                        NULL, NULL,
                                        NULL);
    XmStringFree (cut);
    XmStringFree (clear);
    XmStringFree (copy);
    XmStringFree (paste);
```

See Also

XmCascadeButtonGadget(2), XmLabelGadget(2), XmMenuShell(2), XmPulldownMenu(2), XmPushButtonGadget(2), XmRowColumn(2), XmSeparatorGadget(2), XmToggleButtonGadget(2).

Name

XmVaCreateSimpleRadioBox – create a RadioBox compound object.

Synopsis

```
Widget XmVaCreateSimpleRadioBox (  Widget          parent,
                                   String          name,
                                   int             button_set,
                                   XtCallbackProc  callback,
                                   ...,
                                   NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the new widget.
name	Specifies the string name of the new widget for resource lookup.
button_set	Specifies the initial setting of the RadioBox.
callback	Specifies the callback procedure that is called when the value of a button changes.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleRadioBox`() is a RowColumn convenience routine that creates a RadioBox with ToggleButtonGadgets as its children. This routine is similar to `XmCreateSimpleRadioBox`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the CheckBox. The *button_set* argument is used to set the XmNmenuHistory resource of the RowColumn. The parameter specifies the *n*th button child of the RadioBox, where the first button is button 0 (zero); the XmNmenuHistory resource is set to the actual widget. The *callback* argument specifies the callback routine that is added to the XmNvalueChangedCallback of each ToggleButtonGadget child of the RadioBox. When the *callback* is invoked, the button number of the button whose value has changed is passed to the *callback* in the *client_data* parameter.

The name of each ToggleButtonGadget child is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the RadioBox. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaRADIOBUTTON, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaRADIOBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a ToggleButtonGadget child of the RadioBox and its associated resources. (As of Motif 1.2, all but the label argument are ignored.)

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleRadioBox`() as in the following example:

```
Widget      toplevel, radio_box;
XmString    one, two, three;

one = XmStringCreateLocalized ("WFNX");
two = XmStringCreateLocalized ("WMJX");
three = XmStringCreateLocalized ("WXKS");
radio_box = XmVaCreateSimpleRadioBox (toplevel, "radio_box", 0, toggled,
                                      XmVaRADIOBUTTON, one, NULL,
                                      NULL, NULL,
                                      XmVaRADIOBUTTON, two, NULL,
                                      NULL, NULL,
                                      XmVaRADIOBUTTON, three,
                                      NULL, NULL, NULL,
                                      NULL);
XmStringFree (one);
XmStringFree (two);
XmStringFree (three);
```

See Also

`XmRadioBox`(2), `XmRowColumn`(2), `XmToggleButtonGadget`(2).

Name

XmWidgetGetBaselines – get the positions of the baselines in a widget.

Synopsis

Boolean XmWidgetGetBaselines (Widget *widget*, Dimension ***baselines*, int **line_count*)

Inputs

widget Specifies the widget for which to get baseline values.

Outputs

baselines Returns an array containing the value of each baseline of text in the widget.

line_count Returns the number of lines of text in the widget.

Returns

True if the widget contains at least one baseline or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmWidgetGetBaselines`() returns an array that contains the baseline values for the specified *widget*. For each line of text in the widget, the baseline value is the vertical offset in pixels from the origin of the bounding box of the widget to the text baseline. The routine returns the baseline values in *baselines* and the number of lines of text in the widget in *line_count*. `XmWidgetGetBaselines`() returns True if the *widget* contains at least one line of text and therefore has a baseline. If the *widget* does not contain any text, the routine returns False and the values of *baselines* and *line_count* are undefined. The routine allocates storage for the returned values. The application is responsible for freeing this storage using `XtFree`().

Usage

XmWidgetGetBaselines() provide information that is useful when you are laying out an application and trying to align different components.

See Also

`XmWidgetGetDisplayRect`(1).

Name

XmWidgetGetDisplayRect – get the display rectangle for a widget.

Synopsis

Boolean XmWidgetGetDisplayRect (Widget *widget*, XRectangle **displayrect*)

Inputs

widget Specifies the widget for which to get the display rectangle.

Outputs

displayrect Returns an XRectangle that specifies the display rectangle of the widget.

Returns

True if the widget has a display rectangle or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmWidgetGetDisplayRect`() gets the display rectangle for the specified *widget*. The routine returns the width, the height, and the x and y-coordinates of the upper left corner of the display rectangle in the *displayrect* XRectangle. All of the values are specified as pixels. The display rectangle for a widget is the smallest rectangle that encloses the string or the pixmap in the widget. `XmWidgetGetDisplayRect`() returns True if the widget has a display rectangle; other it returns False and the value of *displayrect* is undefined.

Usage

`XmWidgetGetDisplayRect`() provide information that is useful when you are laying out an application and trying to align different components.

See Also

`XmWidgetGetBaselines`(1).