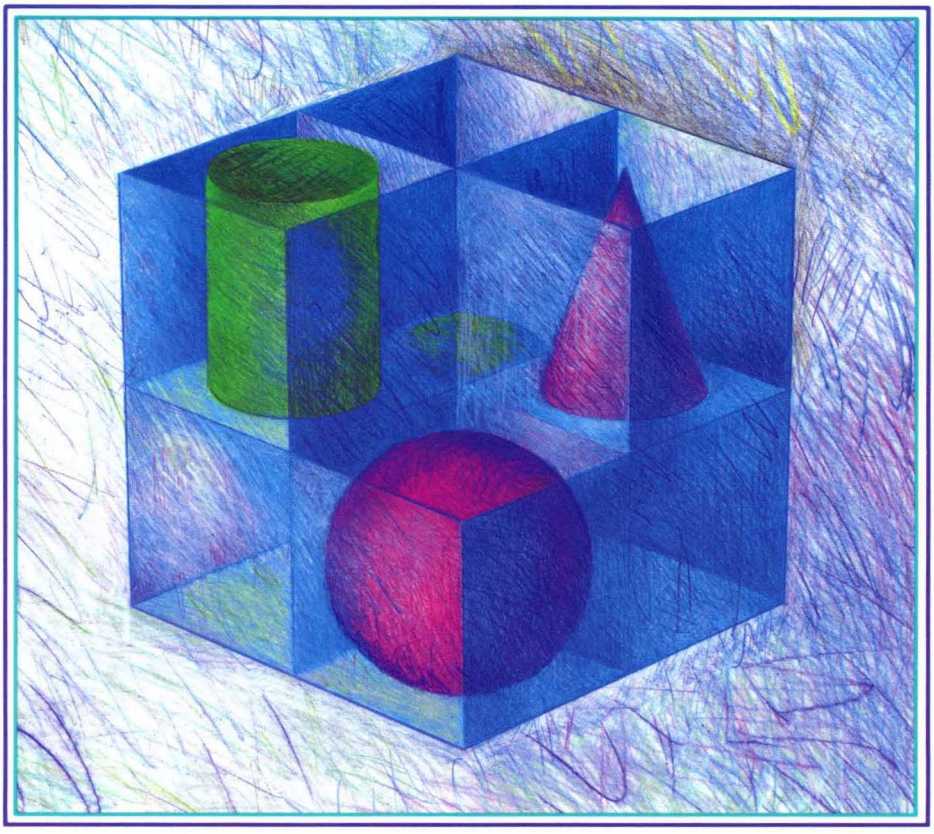
OSF/Motif™

Release 1.2
Revised and Updated

Programmer's Reference



OPEN SOFTWARE FOUNDATION

OSF/Motif™ Programmer's Reference

Revision 1.2

(For OSF/Motif Release 1.2)

Open Software Foundation

P T R Prentice Hall, Englewood Cliffs, New Jersey 07632

Cover design
and cover illustration: **BETH FAGAN**

This book was formatted with troff

Published by P T R Prentice-Hall, Inc.
A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-643115-1

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Contents

Preface

The *OSF/Motif Programmer's Reference* contains the reference pages for OSF/Motif ™ commands and functions, including toolkit, window manager, and user interface language commands and functions.

Audience

This document is written for programmers who want to write applications using Motif ™ interfaces.

This document assumes that the reader is familiar with the American National Standards Institute (ANSI) C programming language. It also assumes that the reader has a general understanding of the X Window System, the Xlib library, and the X Toolkit Intrinsics (Xt).

Applicability

This is Revision 1.2 of this document. It applies to Release 1.2 of the OSF/Motif software system.

Purpose

The purpose of this reference is to provide detailed descriptions of the Motif commands and functions.

Document Usage

This document is organized into one chapter and three appendixes:

- Chapter 1 contains all the reference pages for the Motif commands and functions.
- Appendix A contains a list of the constraint arguments and automatically created children for widgets available within UIL.
- Appendix B contains a list of the reasons and controls, or children, that UIL supports for each Motif Toolkit object.
- Appendix C contains a list of the UIL arguments and their data types.

Reference Page Format

The reference pages in this volume use the following format:

Purpose

This section gives a short description of the interface.

Synopsis

This section describes the appropriate syntax for using the interface.

Description

This section describes the behavior of the interface. On widget reference pages there are tables of resource values in the descriptions. These tables have the following headings:

Name Contains the name of the resource. Each new resource is described following the new resources table.

Class Contains the class of the resource.

Type Contains the type of the resource.

Default Contains the default value of the resource.

Access Contains the access permissions for the resource. A **C** in this column means the resource can be set at widget creation time. An **S** means the resource can be set anytime. A **G** means the resource's value can be retrieved.

Examples

This sections gives practical examples for using the interface.

Return Value

This section lists the values returned by function interfaces.

Errors

This section describes the error conditions associated with using this interface.

Related Information

This section provides cross-references to related interfaces and header files described within this document.

Related Documents

For additional information about OSF/Motif, refer to the following documents:

- The *Application Environment Specification — User Environment Volume* defines a stable set of routines for creating user interface applications.
- The *OSF/Motif Style Guide* explains the principles of user interface design for application developers.
- The *OSF/Motif User's Guide* explains how to interact with OSF/Motif applications.

For additional information about Xlib and Xt, refer to the following X Window System documents:

- *Xlib—C Language X Interface* is the specification for Xlib.
- *X Toolkit Intrinsics—C Language Interface* is the specification for Xt.

Typographic and Keying Conventions

This document uses the following typographic conventions:

Bold **Bold** words or characters represent system elements that an application or user must use literally, such as functions, data types, commands, flags, and pathnames. **Bold** words also indicate the first use of a term included in the glossary.

Italic *Italic* words or characters represent variable values and arguments that an application or user must supply.

`Constant width`
Examples and information that the system displays appear in this typeface.

< > Angle brackets enclose the name of a key on the keyboard.

ComponentName
Components of the user interface are represented by uppercase letters for each major word in the name of the component,

such as PushButton.

Keyboard Conventions

Because not all keyboards are the same, it is difficult to specify keys that are correct for every manufacturer's keyboard. To solve this problem, this reference describes keys using a **virtual key** mechanism. The term *virtual* implies that the keys as described do not necessarily correspond to a fixed set of actual keys. Instead, virtual keys are linked to actual keys by means of **virtual bindings**. A given virtual key may be bound to different physical keys for different keyboards.

See the *OSF/Motif Programmer's Guide* for information on the mechanism for binding virtual keys to actual keys. For details see the **VirtualBindings(3X)** reference page in this document.

Mouse Conventions

Mouse buttons are described in this reference using a **virtual button** mechanism to better describe behavior independent from the number of buttons on the mouse. This guide assumes a 3-button mouse. On a 3-button mouse, the leftmost mouse button is usually defined as **BSelect**, the middle mouse button is usually defined as **BTransfer**, and the rightmost mouse button is usually defined as **BMenu**. For details about how virtual mouse buttons are usually defined, see the **VirtualBindings(3X)** reference page in this document.

Problem Reporting

If you have any problems with the software or documentation, please contact your software vendor's customer service department.

Chapter 1

Reference Pages

This chapter contains the reference pages for the *OSF/Motif Programmer's Reference*.

mwm—The Motif Window Manager

Synopsis

mwm [*options*]

Description

mwm is an X Window System client that provides window management functionality and some session management functionality. It provides functions that facilitate control (by the user and the programmer) of elements of window state such as placement, size, icon/normal display, and input-focus ownership. It also provides session management functions such as stopping a client.

Options

-display *display*
This option specifies the display to use; see **X(1)**.

-xrm *resourcestring*
This option specifies a resource string to use.

-multiscreen
This option causes **mwm** to manage all screens on the display. The default is to manage only a single screen.

-name *name*
This option causes **mwm** to retrieve its resources using the specified name, as in *name*resource*.

-screens *name [name [...]]*
This option specifies the resource names to use for the screens managed by **mwm**. If **mwm** is managing a single screen, only the first name in the list is used. If **mwm** is managing multiple screens, the names are assigned to the screens in order, starting with screen 0. Screen 0 gets the first name, screen 1 the second name, and so on.

Appearance

The following sections describe the basic default behaviors of windows, icons, the icon box, input focus, and window stacking. The appearance and behavior of the window manager can be altered by changing the configuration of specific resources. Resources are defined under the heading "X DEFAULTS."

Screens

By default, **mwm** manages only the single screen specified by the **-display** option or the DISPLAY environment variable (by default, screen 0). If the **-multiscreen** option is specified or if the **multiScreen** resource is True, **mwm** tries to manage all the screens on the display.

When **mwm** is managing multiple screens, the **-screens** option can be used to give each screen a unique resource name. The names are separated by blanks, for example, **-screens mwm0 mwm1**. If there are more screens than names, resources for the remaining screens will be retrieved using the first name. By default, the screen number is used for the screen name.

Windows

Default **mwm** window frames have distinct components with associated functions:

Title Area

In addition to displaying the client's title, the title area is used to move the window. To move the window, place the pointer over the title area, pressing button 1 and dragging the window to a new location. By default, a wire frame is moved during the drag to indicate the new location. When the button is released, the window is moved to the new location.

Title Bar The title bar includes the title area, the minimize button, the maximize button, and the window menu button. In shaped windows, such as round windows, the title bar floats above the window.

Minimize Button

To turn the window into an icon, click button 1 on the minimize button (the frame box with a *small* square in it).

Maximize Button

To make the window fill the screen (or enlarge to the largest size allowed by the configuration files), click button 1 on the maximize button (the frame box with a *large* square in it).

Window Menu Button

The window menu button is the frame box with a horizontal bar in it. To pull down the window menu, press button 1. While pressing, drag the pointer on the menu to your selection, then release the button when your selection is highlighted. Pressing button 3 in the title bar or resize border handles also posts the window menu.

Alternately, you can click button 1 to pull down the menu and keep it posted; then position the pointer and select. You can also post the window menu by pressing **<Shift> <Esc>** or **<Alt> <Space>**. Double-clicking button 1 with the pointer on the window menu button closes the window. The following table lists the contents of the window menu.

Default Window Menu		
Selection	**Accelerator**	**Description**
Restore	**<Alt> <F5>**	
	Restores the window to its size before minimizing or maximizing	
Move	**<Alt> <F7>**	Allows the window to be moved with keys or mouse
Size	**<Alt> <F8>**	Allows the window to be resized
Minimize	**<Alt> <F9>**	Turns the window into an icon
Maximize	**<Alt> <F10>**	Makes the window fill the screen
Lower	**<Alt> <F3>**	Moves window to bottom of window stack
Close	**<Alt> <F4>**	Causes client to terminate

Resize Border Handles

To change the size of a window, move the pointer over a resize border handle (the cursor changes), press button 1, and drag the window to a new size. When the button is released, the window is resized. While dragging is being done, a rubber-band outline is displayed to indicate the new window size.

Matte An optional matte decoration can be added between the client area and the window frame. A matte is not actually part of the window frame. There is no functionality associated with a matte.

Icons

Icons are small graphic representations of windows. A window can be minimized (iconified) using the minimize button on the window frame. Icons provide a way to reduce clutter on the screen.

Pressing mouse button 1 when the pointer is over an icon causes the icon's window menu to pop up. Releasing the button (press and release without moving mouse equals a click) causes the menu to stay posted. The menu contains the selections described in the following table.

Icon Window Menu		
Selection	**Accelerator**	**Description**
Restore	**<Alt> <F5>**	Opens the associated window
Move	**<Alt> <F7>**	Allows the icon to be moved with keys
Size	**<Alt> <F8>**	Inactive (not an option for icons)
Minimize	**<Alt> <F9>**	Inactive (not an option for icons)
Maximize	**<Alt> <F10>**	Opens the associated window and makes it fill the screen
Lower	**<Alt> <F3>**	Moves icon to bottom of icon stack
Close	**<Alt> <F4>**	Removes client from **mwm** management

Note that pressing button 3 over an icon also causes the icon's window menu to pop up. To make a menu selection, drag the pointer over the menu and release button 3 when the desired item is highlighted.

Double-clicking button 1 on an icon invokes the **f.restore_and_raise** function and restores the icon's associated window to its previous state. For example, if a maximized window is iconified, then double-clicking button 1 restores it to its maximized state. (In general, double-clicking a mouse button is a quick way to perform a function.) Pressing **<Shift> <Esc>** or **<Menu>** (the pop-up menu key) causes the icon window menu of the currently selected icon to pop up.

Icon Box

When icons begin to clutter the screen, they can be packed into an icon box. (To use an icon box, **mwm** must be started with the icon box configuration already set.) The icon box is an **mwm** window that holds client icons. It includes one or more scroll bars when there are more window icons than the icon box can show at the same time. Double-clicking button 1 on the icon box's icon opens the icon box and allows access to the contained icons.

Icons in the icon box can be manipulated with the mouse. The following table summarizes the behavior of this interface. Button actions apply whenever the pointer is on any part of the icon. Note that double-clicking an icon in the icon box invokes the **f.restore_and_raise** function.

Button Action	Description
Button 1 click	Selects the icon
Button 1 double-click	Normalizes (opens) the associated window Raises an already open window to the top of the stack
Button 1 drag	Moves the icon
Button 3 press	Causes the menu for that icon to pop up
Button 3 drag	Highlights items as the pointer moves across the menu

Icon Menu for the Icon Box		
Selection	**Accelerator**	**Description**
Restore	**<Alt> <F5>**	Opens the associated window (if not already open)
Move	**<Alt> <F7>**	Allows the icon to be moved with keys
Size	**<Alt> <F8>**	Inactive
Minimize	**<Alt> <F9>**	Inactive
Maximize	**<Alt> <F10>**	Opens the associated window (if not already open) and maximizes its size
Lower	**<Alt> <F3>**	Inactive
Close	**<Alt> <F4>**	Removes client from **mwm** management

To pull down the window menu for the icon box itself, press button 1 with the pointer over the menu button for the icon box. The window menu of the icon box differs from the window menu of a client window: The "Close" selection is replaced with the "PackIcons" **<Shift> <Alt> <F7>** selection. When selected, PackIcons packs the icons in the box to achieve neat rows with no empty slots.

You can also post the window menu by pressing **<Shift> <Esc>** or **<Alt> <Space>**. Pressing **<Menu>** (the pop-up menu key) causes the icon window menu of the currently selected icon to pop up.

Input Focus

mwm supports (by default) a keyboard input focus policy of explicit selection. This means when a window is selected to get keyboard input, it continues to get keyboard input until the window is withdrawn from window management, another window is explicitly selected to get keyboard input, or the window is iconified. Several resources control the input focus. The client window with the keyboard input focus has the active window appearance with a visually distinct window frame.

The following tables summarize the keyboard input focus selection behavior.

Button Action	Object	Function Description
Button 1 press	Window / window frame	Keyboard focus selection
Button 1 press	Icon	Keyboard focus selection

Key Action	Function Description
<Alt> <Tab>	Move input focus to next window in window stack (available only in explicit focus mode)
<Alt> <Shift> <Tab>	Move input focus to previous window in window stack (available only in explicit focus mode)

Window Stacking

There are two types of window stacks: global window stacks and an application's local family window stack.

The global stacking order of windows may be changed as a result of setting the keyboard input focus, iconifying a window, or performing a window manager window stacking function. When keyboard focus policy is explicit, the default value of the **focusAutoRaise** resource is True. This causes a window to be raised to the top of the stack when it receives input focus, for example, when button 1 is pressed on the title bar. The key actions defined in the previous table will thus raise the window receiving focus to the top of the stack.

In pointer mode, the default value of **focusAutoRaise** is False; that is, the window stacking order is not changed when a window receives keyboard input focus. The following key actions can be used to cycle through the global window stack.

Key Action	Function Description
<Alt> <Esc>	Place top window on bottom of stack
<Alt> <Shift> <Esc>	Place bottom window on top of stack

By default, a window's icon is placed on the bottom of the stack when the window is iconified; however, the default can be changed by the **lowerOnIconify** resource.

Transient windows (secondary windows such as dialog boxes) stay above their parent windows by default. However, an application's local family stacking order may be changed to allow a transient window to be placed below its parent top-level window.

The following parameters show the modification of the stacking order for the **f.lower** function:

f.lower Lowers the transient window within the family (staying above the parent) and lowers the family in the global window stack.

f.lower [within]
Lowers the transient window within the family (staying above the parent), but does not lower the family in the global window stack.

f.lower [freeFamily]
Lowers the window free from its family stack (below the parent), but does not lower the family in the global window stack.

The parameters **within** and **freeFamily** can also be used with **f.raise** and **f.raise_lower**.

X Defaults

mwm is configured from its resource database. This database is built from the following sources. They are listed in order of precedence, low to high:

- **/usr/lib/X11/app-defaults/Mwm**
- **$HOME/Mwm**
- RESOURCE_MANAGER root window property or **$HOME/.Xdefaults**
- XENVIRONMENT variable or **$HOME/.Xdefaults-***host*
- **mwm** command line options

The filenames **/usr/lib/X11/app-defaults/Mwm** and **$HOME/Mwm** represent customary locations for these files. The actual location of the system-wide class resource file may depend on the XFILESEARCHPATH environment variable and the current language environment. The actual location of the user-specific class resource file may depend on the XUSERFILESEARCHPATH and XAPPLRESDIR environment variables and the current language environment.

Entries in the resource database may refer to other resource files for specific types of resources. These include files that contain bitmaps, fonts, and **mwm** specific resources, such as menus and behavior specifications (for example, button and key bindings).

Mwm is the resource class name of **mwm**, and **mwm** is the default resource name used by **mwm** to look up resources. The **-screens** command line option specifies resource names, such as **mwm_b+w** and **mwm_color**. In the following discussion of resource specification, **Mwm** and **mwm** (and the aliased **mwm** resource names) can be used interchangeably, but **mwm** takes precedence over **Mwm.**

mwm uses the following types of resources:

Component Appearance Resources
: These resources specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, feedback windows (for example, the window reconfiguration feedback window), client window frames, and icons.

General Appearance and Behavior Resources
: These resources specify **mwm** appearance and behavior (for example, window management policies). They are not set separately for different **mwm** user interface components.

Client Specific Resources
: These **mwm** resources can be set for a particular client window or class of client window. They specify client-specific icon and client window frame appearance and behavior.

  Resource identifiers can be either a resource name (for example, foreground) or a resource class (for example, Foreground). If the value of a resource is a filename and if the filename is prefixed by ~/ (tilde, slash), then it is relative to the path contained in the HOME environment variable (generally the user's home directory).

Component Appearance Resources

The syntax for specifying component appearance resources that apply to window manager icons, menus, and client window frames is

Mwm*_resource_id_

For example, **Mwm*foreground** is used to specify the foreground color for **mwm** menus, icons, client window frames, and feedback dialogs.

The syntax for specifying component appearance resources that apply to a particular **mwm** component is

Mwm*[menu|icon|client|feedback]*_resource_id_

If **menu** is specified, the resource is applied only to **mwm** menus; if **icon** is specified, the resource is applied to icons; and if **client** is specified, the resource is applied to client window frames. For example, **Mwm*icon*foreground** is used to specify the foreground color for **mwm** icons, **Mwm*menu*foreground** specifies the foreground color for **mwm** menus, and **Mwm*client*foreground** is used to specify the foreground color for **mwm** client window frames.

The appearance of the title area of a client window frame (including window management buttons) can be separately configured. The syntax for configuring the title area of a client window frame is

Mwm*client*title*resource_id*

For example, **Mwm*client*title*foreground** specifies the foreground color for the title area. Defaults for title area resources are based on the values of the corresponding client window frame resources.

The appearance of menus can be configured based on the name of the menu. The syntax for specifying menu appearance by name is

Mwm*menu*menu_name*resource_id*

For example, **Mwm*menu*my_menu*foreground** specifies the foreground color for the menu named **my_menu**. The user can also specify resources for window manager menu components, that is, the gadgets that make up the menu. These may include for example, a menu title, title separator, one or more buttons, and separators. If a menu contains more than one instance of a class, such as multiple PushButtonGadgets, the name of the first instance is **PushButtonGadget1,** the second is **PushButtonGadget2,** and so on. The following list identifies the naming convention used for window manager menu components:

Menu Title LabelGadget	**TitleName**
Menu Title SeparatorGadget	**TitleSeparator**
CascadeButtonGadget	**CascadeButtonGadget***n*
PushButtonGadget	**PushButtonGadget***n*
SeparatorGadget	**SeparatorGadget***n*

Refer to the reference page for each class for a list of resources that can be specified.

The following component appearance resources that apply to all window manager parts can be specified.

Component Appearance Resources—All Window Manager Parts			
Name	**Class**	**Value Type**	**Default**
background	Background	color	varies[1]
backgroundPixmap	BackgroundPixmap	string[2]	varies[1]
bottomShadowColor	Foreground	color	varies[1]
bottomShadowPixmap	BottomShadowPixmap	string[2]	varies[1]
fontList	FontList	string[3]	"fixed"
foreground	Foreground	color	varies[1]
saveUnder	SaveUnder	T/F	F
topShadowColor	Background	color	varies[1]
topShadowPixmap	TopShadowPixmap	string[2]	varies[1]

[1]The default is chosen based on the visual type of the screen.
[2]Image name. See **XmInstallImage(3X)**.
[3]X11 X Logical Font Description.

background (class **Background**)
This resource specifies the background color. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

backgroundPixmap (class **BackgroundPixmap**)
This resource specifies the background pixmap of the **mwm** decoration when the window is inactive (does not have the keyboard focus). The default value is chosen based on the visual type of the screen.

bottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color. This color is used for the lower and right bevels of the window manager decoration. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

bottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow pixmap. This pixmap is used for the lower and right bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

fontList (class **FontList**)
This resource specifies the font used in the window manager decoration. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

foreground (class **Foreground**)
This resource specifies the foreground color. The default is chosen based on the visual type of the screen.

saveUnder (class **SaveUnder**)
This is used to indicate whether "save unders" are used for **mwm** components. For this to have any effect, save unders must be implemented by the X server. If save unders are implemented, the X server saves the contents of windows obscured by windows that have the save under attribute set. If the **saveUnder** resource is True, **mwm** will set the save under attribute on the window manager frame of any client that has it set. If **saveUnder** is False, save unders will not be used on any window manager frames. The default value is False.

topShadowColor (class **Background**)
This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

topShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow pixmap. This pixmap is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

The following component appearance resources that apply to frame and icons can be specified.

Frame and Icon Components			
Name	**Class**	**Value Type**	**Default**
activeBackground	Background	color	varies[1]
activeBackgroundPixmap	BackgroundPixmap	string[2]	varies[1]
activeBottomShadowColor	Foreground	color	varies[1]
activeBottomShadowPixmap	BottomShadowPixmap	string[2]	varies[1]
activeForeground	Foreground	color	varies[1]
activeTopShadowColor	Background	color	varies[1]
activeTopShadowPixmap	TopShadowPixmap	string[2]	varies[1]

[1]The default is chosen based on the visual type of the screen.
[2]See **XmInstallImage(3X)**.

activeBackground (class **Background**)
This resource specifies the background color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBackgroundPixmap (class **ActiveBackgroundPixmap**)
This resource specifies the background pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeForeground (class **Foreground**)
This resource specifies the foreground color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowColor (class **Background**)
This resource specifies the top shadow color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

General Appearance and Behavior Resources

The syntax for specifying general appearance and behavior resources is

Mwm*_resource_id_

For example, **Mwm*keyboardFocusPolicy** specifies the window manager policy for setting the keyboard focus to a particular client window.

The following general appearance and behavior resources can be specified.

General Appearance and Behavior Resources			
Name	**Class**	**Value Type**	**Default**
autoKeyFocus	AutoKeyFocus	T/F	T
autoRaiseDelay	AutoRaiseDelay	millisec	500
bitmapDirectory	BitmapDirectory	directory	/usr/include/\ X11/bitmaps
buttonBindings	ButtonBindings	string	**DefaultBut\ tonBindings**
cleanText	CleanText	T/F	T
clientAutoPlace	ClientAutoPlace	T/F	T
colormapFocusPolicy	ColormapFocusPolicy	string	keyboard
configFile	ConfigFile	file	.mwmrc
deiconifyKeyFocus	DeiconifyKeyFocus	T/F	T
doubleClickTime	DoubleClickTime	millisec.	multi-click time
enableWarp	enableWarp	T/F	T
enforceKeyFocus	EnforceKeyFocus	T/F	T
fadeNormalIcon	FadeNormalIcon	T/F	F
feedbackGeometry	FeedbackGeometry	string	center on screen
frameBorderWidth	FrameBorderWidth	pixels	varies
iconAutoPlace	IconAutoPlace	T/F	T
iconBoxGeometry	IconBoxGeometry	string	6x1+0-0
iconBoxName	IconBoxName	string	iconbox
iconBoxSBDisplayPolicy	IconBoxSBDisplayPolicy	string	all
iconBoxTitle	IconBoxTitle	XmString	Icons
iconClick	IconClick	T/F	T
iconDecoration	IconDecoration	string	varies
iconImageMaximum	IconImageMaximum	wxh	50x50
iconImageMinimum	IconImageMinimum	wxh	16x16
iconPlacement	IconPlacement	string	left bottom
iconPlacementMargin	IconPlacementMargin	pixels	varies
interactivePlacement	InteractivePlacement	T/F	F

General Appearance and Behavior Resources			
Name	**Class**	**Value Type**	**Default**
keyBindings	KeyBindings	string	"DefaultKey\ Bindings"
keyboardFocusPolicy	KeyboardFocusPolicy	string	explicit
limitResize	LimitResize	T/F	T
lowerOnIconify	LowerOnIconify	T/F	T
maximumMaximumSize	MaximumMaximumSize	wxh (pixels)	2X screen w&h
moveThreshold	MoveThreshold	pixels	4
moveOpaque	MoveOpaque	T/F	F
multiScreen	MultiScreen	T/F	F
passButtons	PassButtons	T/F	F
passSelectButton	PassSelectButton	T/F	T
positionIsFrame	PositionIsFrame	T/F	T
positionOnScreen	PositionOnScreen	T/F	T
quitTimeout	QuitTimeout	millisec.	1000
raiseKeyFocus	RaiseKeyFocus	T/F	F
resizeBorderWidth	ResizeBorderWidth	pixels	varies
resizeCursors	ResizeCursors	T/F	T
screens	Screens	string	varies
showFeedback	ShowFeedback	string	all
startupKeyFocus	StartupKeyFocus	T/F	T
transientDecoration	TransientDecoration	string	menu title
transientFunctions	TransientFunctions	string	-minimize -maximize
useIconBox	UseIconBox	T/F	F
wMenuButtonClick	WMenuButtonClick	T/F	T
wMenuButtonClick2	WMenuButtonClick2	T/F	T

autoKeyFocus (class **AutoKeyFocus**)

This resource is available only when the keyboard input focus policy is explicit. If **autoKeyFocus** is given a value of True, then when a window with the keyboard input focus is withdrawn from window management or is iconified, the focus is set to the previous window that had the focus. If the value given is False, there is no automatic setting

of the keyboard input focus. It is recommended that both **autoKeyFocus** and **startupKeyFocus** be True to work with tear off menus. The default value is True.

autoRaiseDelay (class **AutoRaiseDelay**)
This resource is available only when the **focusAutoRaise** resource is True and the keyboard focus policy is pointer. The **autoRaiseDelay** resource specifies the amount of time (in milliseconds) that **mwm** will wait before raising a window after it gets the keyboard focus. The default value of this resource is 500 (ms).

bitmapDirectory (class **BitmapDirectory**)
This resource identifies a directory to be searched for bitmaps referenced by **mwm** resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is **/usr/include/X11/bitmaps**. The directory **/usr/include/X11/bitmaps** represents the customary locations for this directory. The actual location of this directory may vary on some systems. If the bitmap is not found in the specified directory, XBMLANGPATH is searched.

buttonBindings (class **ButtonBindings**)
This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the **mwm** resource description file. These button bindings are *merged* with the built-in default bindings. The default value for this resource is **DefaultButtonBindings**.

cleanText (class **CleanText**)
This resource controls the display of window manager text in the client title and feedback windows. If the default value of True is used, the text is drawn with a clear (no stipple) background. This makes text easier to read on monochrome systems where a background pixmap is specified. Only the stippling in the area immediately around the text is cleared. If False, the text is drawn directly on top of the existing background.

clientAutoPlace (class **ClientAutoPlace**)
This resource determines the position of a window when the window has not been given a program- or user-specified position. With a value of True, windows are positioned with the top left corners of the frames offset horizontally and vertically. A value of False causes the currently configured position of the window to be used. In either case, **mwm** will attempt to place the windows totally on-screen. The default value is True.

colormapFocusPolicy (class **ColormapFocusPolicy**)
This resource indicates the colormap focus policy that is to be used. If the resource value is explicit, a colormap selection action is done on a client window to set the colormap focus to that window. If the value is pointer, the client window containing the pointer has the colormap focus. If the value is keyboard, the client window that has the keyboard input focus has the colormap focus. The default value for this resource is keyboard.

configFile (class **ConfigFile**)
The resource value is the pathname for an **mwm** resource description file.

If the pathname begins with ~/ (tilde, slash), **mwm** considers it to be relative to the user's home directory (as specified by the HOME environment variable). If the LANG environment variable is set, **mwm** looks for **$HOME/$LANG/***configFile*. If that file does not exist or if LANG is not set, **mwm** looks for **$HOME/***configFile*.

If the **configFile** pathname does not begin with ~/, **mwm** considers it to be relative to the current working directory.

If the **configFile** resource is not specified or if that file does not exist, **mwm** uses several default paths to find a configuration file. If the LANG environment variable is set, **mwm** looks for the configuration file first in **$HOME/$LANG/.mwmrc.** If that file does not exist or if LANG is not set, **mwm** looks for **$HOME/.mwmrc**. If that file does not exist and if LANG is set, **mwm** next looks for the file **system.mwmrc** in the $LANG subdirectory of an implementation-dependent directory. (The default for this directory, if not changed by the implementation, is **/usr/lib/X11**.) If that file does not exist or if LANG is not set, **mwm** looks for the file **system.mwmrc** in the same implementation-dependent directory.

deiconifyKeyFocus (class **DeiconifyKeyFocus**)
This resource applies only when the keyboard input focus policy is explicit. If a value of True is used, a window receives the keyboard input focus when it is normalized (deiconified). True is the default value.

doubleClickTime (class **DoubleClickTime**)
This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double-click. The default value of this resource is the display's multiclick time.

enableWarp (class **EnableWarp**)
The default value of this resource, True, causes **mwm** to warp the pointer to the center of the selected window during keyboard-controlled resize and move operations. Setting the value to False causes **mwm** to leave the pointer at its original place on the screen, unless the user explicitly moves it with the cursor keys or pointing device.

enforceKeyFocus (class **EnforceKeyFocus**)
If this resource is given a value of True, the keyboard input focus is always explicitly set to selected windows even if there is an indication that they are "globally active" input windows. (An example of a globally active window is a scroll bar that can be operated without setting the focus to that client.) If the resource is False, the keyboard input focus is not explicitly set to globally active windows. The default value is True.

fadeNormalIcon (class **FadeNormalIcon**)
If this resource is given a value of True, an icon is grayed out whenever it has been normalized (its window has been opened). The default value is False.

feedbackGeometry (class **FeedbackGeometry**)
This resource sets the position of the move and resize feedback window. If this resource is not specified, the default is to place the feedback window at the center of the screen. The value of the resource is a standard window geometry string with the following syntax:

[=]{+-}*xoffset*{+-}*yoffset*]

frameBorderWidth (class **FrameBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border without resize handles. The border width includes the 3-D shadows. The default value is based on the size and resolution of the screen.

iconAutoPlace (class **IconAutoPlace**)
This resource indicates whether the window manager arranges icons in a particular area of the screen or places each icon where the window was when it was iconified. The value True indicates that icons are arranged in a particular area of the screen, determined by the iconPlacement resource. The value False indicates that an icon is placed at the location of the window when it is iconified. The default is True.

iconBoxGeometry (class **IconBoxGeometry**)
This resource indicates the initial position and size of the icon box. The value of the resource is a standard window geometry string with the following syntax:

[=][*width*x*height*][{+-}*xoffset*{+-}*yoffset*]

If the offsets are not provided, the iconPlacement policy is used to determine the initial placement. The units for width and height are columns and rows.

The actual screen size of the icon box window depends on the **iconImageMaximum** (size) and **iconDecoration** resources. The default value for size is (6 * **iconWidth** + padding) wide by (1 * **iconHeight** + padding) high. The default value of the location is +0 -0.

iconBoxName (class **IconBoxName**)
This resource specifies the name that is used to look up icon box resources. The default name is **iconbox**.

iconBoxSBDisplayPolicy (class **IconBoxSBDisplayPolicy**)
This resource specifies the scroll bar display policy of the window manager in the icon box. The resource has three possible values: **all**, **vertical**, and **horizontal**. The default value, **all**, causes both vertical and horizontal scroll bars always to appear. The value **vertical** causes a single vertical scroll bar to appear in the icon box and sets the orientation of the icon box to horizontal (regardless of the **iconBoxGeometry** specification). The value **horizontal** causes a single horizontal scroll bar to appear in the icon box and sets the orientation of the icon box to vertical (regardless of the **iconBoxGeometry** specification).

iconBoxTitle (class **IconBoxTitle**)
This resource specifies the name that is used in the title area of the icon box frame. The default value is **Icons**.

iconClick (class **IconClick**)
When this resource is given the value of True, the system menu is posted and left posted when an icon is clicked. The default value is True.

iconDecoration (class **IconDecoration**)
This resource specifies the general icon decoration. The resource value is label (only the label part is displayed) or image (only the image part is displayed) or label image (both the label and image parts are

displayed). A value of activelabel can also be specified to get a label (not truncated to the width of the icon) when the icon is selected. The default icon decoration for icon box icons is that each icon has a label part and an image part (label image). The default icon decoration for standalone icons is that each icon has an active label part, a label part, and an image part (activelabel label image).

iconImageMaximum (class **IconImageMaximum**)
This resource specifies the maximum size of the icon *image*. The resource value is *width***x***height* (for example, 64x64). The maximum supported size is 128x128. The default value of this resource is 50x50.

iconImageMinimum (class **IconImageMinimum**)
This resource specifies the minimum size of the icon *image*. The resource value is *width***x***height* (for example, 32x50). The minimum supported size is 16x16. The default value of this resource is 16x16.

iconPlacement (class **IconPlacement**)
This resource specifies the icon placement scheme to be used. The resource value has the following syntax:

primary_layout secondary_layout [tight]

The layout values are one of the following:

Value	Description
top	Lay the icons out top to bottom.
bottom	Lay the icons out bottom to top.
left	Lay the icons out left to right.
right	Lay the icons out right to left.

A horizontal (vertical) layout value should not be used for both the *primary_layout* and the *secondary_layout* (for example, do not use top for the *primary_layout* and bottom for the *secondary_layout*). The *primary_layout* indicates whether, when an icon placement is done, the icon is placed in a row or a column and the direction of placement. The *secondary_layout* indicates where to place new rows or columns. For example, top right indicates that icons should be placed top to bottom on the screen and that columns should be added from right to left on the screen. The default placement is left bottom (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen). A *tight* value places icons with zero spacing in between icons. This value is useful for aesthetic reasons, as well as X-terminals with small screens.

iconPlacementMargin (class **IconPlacementMargin**)
This resource sets the distance between the edge of the screen and the icons that are placed along the edge of the screen. The value should be greater than or equal to 0. A default value (see below) is used if the value specified is invalid. The default value for this resource is equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column).

interactivePlacement (class **InteractivePlacement**)
This resource controls the initial placement of new windows on the screen. If the value is True, the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper left corner of the window. If the value is False, windows are placed according to the initial window configuration attributes. The default value of this resource is False.

keyBindings (class **KeyBindings**)
This resource identifies the set of key bindings for window management functions. If specified, these key bindings *replace* the built-in default bindings. The named set of key bindings is specified in the **mwm** resource description file. The default value for this resource is **DefaultKeyBindings**.

keyboardFocusPolicy (class **KeyboardFocusPolicy**)
If set to pointer, the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that **mwm** adds). If set to explicit, the policy is to have the keyboard focus set to a client window when the user presses button 1 with the pointer on the client window or any part of the associated **mwm** decoration. The default value for this resource is explicit.

limitResize (class **LimitResize**)
If this resource is True, the user is not allowed to resize a window to greater than the maximum size. The default value for this resource is True.

lowerOnIconify (class **LowerOnIconify**)
If this resource is given the default value of True, a window's icon appears on the bottom of the window stack when the window is minimized (iconified). A value of False places the icon in the stacking order at the same place as its associated window. The default value of this resource is True.

maximumMaximumSize (class **MaximumMaximumSize**)
This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is *width*x*height* (for example, 1024x1024) where the width and height are in pixels. The default value of this resource is twice the screen width and height.

moveOpaque (class **MoveOpaque**)
This resource controls whether the actual window is moved or a rectangular outline of the window is moved. A default value of False displays a rectangular outline on moves.

moveThreshold (class **MoveThreshold**)
This resource is used to control the sensitivity of dragging operations that move windows and icons. The value of this resource is the number of pixels that the locator is moved with a button down before the move operation is initiated. This is used to prevent window/icon movement when you click or double-click and there is unintentional pointer movement with the button down. The default value of this resource is 4 (pixels).

multiScreen (class **MultiScreen**)
This resource, if True, causes **mwm** to manage all the screens on the display. If False, **mwm** manages only a single screen. The default value is False.

passButtons (class **PassButtons**)
This resource indicates whether or not button press events are passed to clients after they are used to do a window manager function in the client context. If the resource value is False, the button press is not passed to the client. If the value is True, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is False.

passSelectButton (class **PassSelectButton**)
This resource indicates whether or not to pass the select button press events to clients after they are used to do a window manager function in the client context. If the resource value is False, then the button press will not be passed to the client. If the value is True, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is True.

positionIsFrame (class **PositionIsFrame**)
This resource indicates how client window position information (from the WM_NORMAL_HINTS property and from configuration requests) is to be interpreted. If the resource value is True, the information is

interpreted as the position of the MWM client window frame. If the value is False, it is interpreted as being the position of the client area of the window. The default value of this resource is True.

positionOnScreen (class **PositionOnScreen**)
This resource is used to indicate that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen (if the resource value is True). If a window is larger than the size of the screen, at least the upper left corner of the window is on-screen. If the resource value is False, windows are placed in the requested position even if totally off-screen. The default value of this resource is True.

quitTimeout (class **QuitTimeout**)
This resource specifies the amount of time (in milliseconds) that **mwm** will wait for a client to update the WM_COMMAND property after **mwm** has sent the WM_SAVE_YOURSELF message. The default value of this resource is 1000 (ms). (Refer to the **f.kill** function description for additional information.)

raiseKeyFocus (class **RaiseKeyFocus**)
This resource is available only when the keyboard input focus policy is explicit. When set to True, this resource specifies that a window raised by means of the **f.normalize_and_raise** function also receives the input focus. The default value of this resource is False.

resizeBorderWidth (class **ResizeBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border with resize handles. The specified border width includes the 3-D shadows. The default value is based on the size and resolution of the screen.

resizeCursors (class **ResizeCursors**)
This resource is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If True, the cursors are shown, otherwise the window manager cursor is shown. The default value is True.

screens (class **Screens**)
This resource specifies the resource names to use for the screens managed by **mwm**. If **mwm** is managing a single screen, only the first name in the list is used. If **mwm** is managing multiple screens, the names are assigned to the screens in order, starting with screen 0. Screen 0 gets the first name, screen 1 the second name, and so on. The default screen names are 0, 1, and so on.

showFeedback (class **ShowFeedback**)

This resource controls whether or not feedback windows or confirmation dialogs are displayed. A feedback window shows a client window's initial placement and shows position and size during move and resize operations. Confirmation dialogs can be displayed for certain operations.

The value for this resource is a list of names of the feedback options to be enabled or disabled; the names must be separated by a space. If an option is preceded by a minus sign, that option is excluded from the list. The *sign* of the first item in the list determines the initial set of options. If the sign of the first option is minus, **mwm** assumes all options are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), **mwm** starts with no options and builds up a list from the resource.

The names of the feedback options are shown in the following table.

Name	Description
all	Show all feedback (Default value)
behavior	Confirm behavior switch
kill	Confirm on receipt of KILL signal
move	Show position during move
none	Show no feedback
placement	Show position and size during initial placement
quit	Confirm quitting **mwm**
resize	Show size during resize
restart	Confirm **mwm** restart

The following sample command line illustrates the syntax for showFeedback:

Mwm*showFeedback: placement resize behavior restart

This resource specification provides feedback for initial client placement and resize, and enables the dialog boxes to confirm the restart and set behavior functions. It disables feedback for the move function. The default value for this resource is all.

startupKeyFocus (class **StartupKeyFocus**)

This resource is available only when the keyboard input focus policy is explicit. When given the default value of True, a window gets the

keyboard input focus when the window is mapped (that is, initially managed by the window manager). It is recommended that both **autoKeyFocus** and **startupKeyFocus** be True to work with tear off menus. The default value is True.

transientDecoration (class **TransientDecoration**)
This controls the amount of decoration that **mwm** puts on transient windows. The decoration specification is exactly the same as for the **clientDecoration** (client specific) resource. Transient windows are identified by the WM_TRANSIENT_FOR property, which is added by the client to indicate a relatively temporary window. The default value for this resource is menu title (that is, transient windows have frame borders and a titlebar with a window menu button).

An application can also specify which decorations **mwm** should apply to its windows. If it does so, **mwm** applies only those decorations indicated by both the application and the **transientDecoration** resource. Otherwise, **mwm** applies the decorations indicated by the **transientDecoration** resource. For more information, see the description of **XmNmwmDecorations** on the **VendorShell(3X)** reference page.

transientFunctions (class **TransientFunctions**)
This resource is used to indicate which window management functions are applicable (or not applicable) to transient windows. The function specification is exactly the same as for the **clientFunctions** (client specific) resource. The default value for this resource is -minimize -maximize.

An application can also specify which functions **mwm** should apply to its windows. If it does so, **mwm** applies only those functions indicated by both the application and the **transientFunctions** resource. Otherwise, **mwm** applies the functions indicated by the **transientFunctions** resource. For more information, see the description of **XmNmwmFunctions** on the **VendorShell(3X)** reference page.

useIconBox (class **UseIconBox**)
If this resource is given a value of True, icons are placed in an icon box. When an icon box is not used, the icons are placed on the root window (default value).

wMenuButtonClick (class **WMenuButtonClick**)
This resource indicates whether a click of the mouse when the pointer is over the window menu button posts and leaves posted the window menu. If the value given this resource is True, the menu remains posted. True is the default value for this resource.

wMenuButtonClick2 (class **WMenuButtonClick2**)
When this resource is given the default value of True, a double-click action on the window menu button does an **f.kill** function.

Client Specific Resources

The syntax for specifying client specific resources is

Mwm*_client_name_or_class_*resource_id*

For example, **Mwm*mterm*windowMenu** is used to specify the window menu to be used with mterm clients. The syntax for specifying client specific resources for all classes of clients is

Mwm*_resource_id*

Specific client specifications take precedence over the specifications for all clients. For example, **Mwm*windowMenu** is used to specify the window menu to be used for all classes of clients that do not have a window menu specified.

The syntax for specifying resource values for windows that have an unknown name and class (that is, windows that do not have a WM_CLASS property associated with them) is

Mwm*defaults*_resource_id*

For example, **Mwm*defaults*iconImage** is used to specify the icon image to be used for windows that have an unknown name and class.

The following client specific resources can be specified.

Client Specific Resources			
Name	**Class**	**Value Type**	**Default**
clientDecoration	ClientDecoration	string	all
clientFunctions	ClientFunctions	string	all
focusAutoRaise	FocusAutoRaise	T/F	varies
iconImage	IconImage	pathname	(image)
iconImageBackground	Background	color	icon background
iconImageBottomShadowColor	Foreground	color	icon bottom shadow
iconImageBottomShadowPixmap	BottomShadow-Pixmap	color	icon bottom shadow pixmap
iconImageForeground	Foreground	color	varies
iconImageTopShadowColor	Background	color	icon top shadow color
iconImageTopShadowPixmap	TopShadow-Pixmap	color	icon top shadow pixmap
matteBackground	Background	color	background
matteBottomShadowColor	Foreground	color	bottom shadow color
matteBottomShadowPixmap	BottomShadow-Pixmap	color	bottom shadow pixmap
matteForeground	Foreground	color	foreground
matteTopShadowColor	Background	color	top shadow color
matteTopShadowPixmap	TopShadow-Pixmap	color	top shadow pixmap
matteWidth	MatteWidth	pixels	0

Client Specific Resources			
Name	**Class**	**Value Type**	**Default**
maximumClientSize	MaximumClientSize	wxh vertical horizontal	fill the screen
useClientIcon	UseClientIcon	T/F	F
usePPosition	UsePPosition	string	nonzero
windowMenu	WindowMenu	string	"Default-Window-Menu"

clientDecoration (class **ClientDecoration**)

This resource controls the amount of window frame decoration. The resource is specified as a list of decorations to specify their inclusion in the frame. If a decoration is preceded by a minus sign, that decoration is excluded from the frame. The *sign* of the first item in the list determines the initial amount of decoration. If the sign of the first decoration is minus, **mwm** assumes all decorations are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), then **mwm** starts with no decoration and builds up a list from the resource.

An application can also specify which decorations **mwm** should apply to its windows. If it does so, **mwm** applies only those decorations indicated by both the application and the **clientDecoration** resource. Otherwise, **mwm** applies the decorations indicated by the **clientDecoration** resource. For more information, see the description of **XmNmwmDecorations** on the **VendorShell(3X)** reference page.

Name	Description
all	Include all decorations (default value)
border	Window border
maximize	Maximize button (includes title bar)
minimize	Minimize button (includes title bar)
none	No decorations
resizeh	Border resize handles (includes border)
menu	Window menu button (includes title bar)
title	Title bar (includes border)

Examples:

Mwm*XClock.clientDecoration: -resizeh -maximize

This removes the resize handles and maximize button from XClock windows.

Mwm*XClock.clientDecoration: menu minimize border

This does the same thing as above. Note that either **menu** or **minimize** implies **title**.

clientFunctions (class **ClientFunctions**)

This resource is used to indicate which **mwm** functions are applicable (or not applicable) to the client window. The value for the resource is a list of functions. If the first function in the list has a minus sign in front of it, then **mwm** starts with all functions and subtracts from that set. If the first function in the list has a plus sign in front of it, then **mwm** starts with no functions and builds up a list. Each function in the list must be preceded by the appropriate plus or minus sign and separated from the next function by a space.

An application can also specify which functions **mwm** should apply to its windows. If it does so, **mwm** applies only those functions indicated by both the application and the **clientFunctions** resource. Otherwise, **mwm** applies the functions indicated by the **clientFunctions** resource. For more information, see the description of **XmNmwmFunctions** on the **VendorShell(3X)** reference page.

The following table lists the functions available for this resource.

Name	Description
all	Include all functions (default value)
none	No functions
resize	f.resize
move	f.move
minimize	f.minimize
maximize	f.maximize
close	f.kill

focusAutoRaise (class **FocusAutoRaise**)
When the value of this resource is True, clients are raised when they get the keyboard input focus. If the value is False, the stacking of windows on the display is not changed when a window gets the keyboard input focus. The default value is True when the **keyboardFocusPolicy** is explicit and False when the **keyboardFocusPolicy** is pointer.

iconImage (class **IconImage**)
This resource can be used to specify an icon image for a client (for example, **Mwm*myclock*iconImage**). The resource value is a pathname for a bitmap file. The value of the (client specific) **useClientIcon** resource is used to determine whether or not user supplied icon images are used instead of client supplied icon images. The default value is to display a built-in window manager icon image.

iconImageBackground (class **Background**)
This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (that is, specified by **Mwm*background** or **Mwm*icon*background**).

iconImageBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow color (that is, specified by **Mwm*icon*bottomShadowColor**).

iconImageBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow pixmap (that is, specified by **Mwm*icon*bottomShadowPixmap**).

iconImageForeground (class **Foreground**)
This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this resource varies depending on the icon background.

iconImageTopShadowColor (class **Background**)
This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color (that is, specified by **Mwm*icon*topShadowColor**).

iconImageTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow pixmap (that is, specified by **Mwm*icon*topShadowPixmap**).

matteBackground (class **Background**)
This resource specifies the background color of the matte, when **matteWidth** is positive. The default value of this resource is the client background color (that is, specified by **Mwm*background** or **Mwm*client*background**).

matteBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow color (that is, specified by **Mwm*bottomShadowColor** or **Mwm*client*bottomShadowColor**).

matteBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow pixmap of the matte, when **matteWidth** is positive. The default value is the client bottom shadow pixmap (that is, specified by **Mwm*bottomShadowPixmap** or **Mwm*client*bottomShadowPixmap**).

matteForeground (class **Foreground**)
This resource specifies the foreground color of the matte, when **matteWidth** is positive. The default value of this resource is the client foreground color (that is, specified by **Mwm*foreground** or **Mwm*client*foreground**).

matteTopShadowColor (class **Background**)
This resource specifies the top shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow color (that is, specified by **Mwm*topShadowColor** or **Mwm*client*topShadowColor**).

matteTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow pixmap of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow pixmap (that is, specified by **Mwm*topShadowPixmap** or **Mwm*client*topShadowPixmap**).

matteWidth (class **MatteWidth**)
This resource specifies the width of the optional matte. The default value is 0, which effectively disables the matte.

maximumClientSize (class **MaximumClientSize**)
This resource is either a size specification or a direction that indicates how a client window is to be maximized. The resource value can be specified as a size specification *width***x***height*. The width and height are interpreted in the units that the client uses (for example, for terminal emulators this is generally characters). Alternately, **vertical** or **horizontal** can be specified to indicate the direction in which the client maximizes.

If this resource is not specified, the maximum size from the WM_NORMAL_HINTS property is used if set. Otherwise the default value is the size where the client window with window management borders fills the screen. When the maximum client size is not determined by the **maximumClientSize** resource, the maximumMaximumSize resource value is used as a constraint on the maximum size.

useClientIcon (class **UseClientIcon**)
If the value given for this resource is True, a client-supplied icon image takes precedence over a user-supplied icon image. The default value is False, giving the user-supplied icon image higher precedence than the client-supplied icon image.

usePPosition (class **UsePPosition**)
This resource specifies whether MWM honors the program specified position **PPosition** specified in the WM_NORMAL_HINTS property in

the absence of a user specified position. Setting this resource to on causes **mwm** to always honor program specified position. Setting this resource to off causes **mwm** to always ignore program specified position. Setting this resource to the default value of nonzero causes **mwm** to honor program specified position other than (0,0).

windowMenu (class **WindowMenu**)
This resource indicates the name of the menu pane that is posted when the window menu is popped up (usually by pressing button 1 on the window menu button on the client window frame). Menu panes are specified in the MWM resource description file. Window menus can be customized on a client class basis by specifying resources of the form **Mwm***_client_name_or_class_***windowMenu** (see "Mwm Resource Description File Syntax"). The default value of this resource is "DefaultWindowMenu".

Resource Description File

The MWM resource description file is a supplementary resource file that contains resource descriptions that are referred to by entries in the defaults files (**.Xdefaults**, **app-defaults/Mwm**). It contains descriptions of resources that are to be used by **mwm**, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular **mwm** resource description file can be selected using the **configFile** resource.

The following types of resources can be described in the **mwm** resource description file:

Buttons Window manager functions can be bound (associated) with button events.

Keys Window manager functions can be bound (associated) with key press events.

Menus Menu panes can be used for the window menu and other menus posted with key bindings and button bindings.

mwm Resource Description File Syntax

The **mwm** resource description file is a standard text file that contains items of information separated by blanks, tabs, and newline characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (for example, the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes ("). Single characters can be quoted by preceding them with the \ (backslash). All text from an unquoted # (pound sign) to the end of the line is regarded as a comment and is not interpreted as part of a resource description. If ! (exclamation point) is the first character in a line, the line is regarded as a comment. If a line ends in \, the next line is considered a continuation of that line.

Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for button and key binding sets, and menu panes. The function specification has the following syntax:

function = function_name [function_args]
function_name = window manager function
function_args = {quoted_item | unquoted_item}

The following functions are supported. If a function is specified that is not one of the supported functions, then it is interpreted by **mwm** as **f.nop**.

f.beep This function causes a beep.

f.circle_down [icon | window]
This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it no longer obscures any other window or icon). This function affects only those windows and icons that obscure other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an **icon** function argument is specified, the function applies only to icons. If a **window** function argument is specified, the function applies only to windows.

f.circle_up [icon | window]
This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that obscure other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. If an **icon** function argument is specified, the function

applies only to icons. If a **window** function argument is specified, the function applies only to windows.

f.exec or **!**

This function causes *command* to be executed (using the value of the MWMSHELL environment variable if it is set, otherwise the value of the SHELL environment variable if it is set, otherwise **/bin/sh**). The **!** notation can be used in place of the **f.exec** function name.

f.focus_color

This function sets the colormap focus to a client window. If this function is done in a root context, the default colormap (set up by the *X Window System* for the screen where MWM is running) is installed and there is no specific client window colormap focus. This function is treated as **f.nop** if colormapFocusPolicy is not explicit.

f.focus_key

This function sets the keyboard input focus to a client window or icon. This function is treated as **f.nop** if **keyboardFocusPolicy** is not explicit or the function is executed in a root context.

f.kill This function is used to terminate a client. If the WM_DELETE_WINDOW protocol is set up, the client is sent a client message event, indicating that the client window should be deleted. If the WM_SAVE_YOURSELF protocol is set up, the client is sent a client message event, indicating that the client needs to prepare to be terminated. If the client does not have the WM_DELETE_WINDOW or WM_SAVE_YOURSELF protocol set up, this function causes a client's X connection to be terminated (usually resulting in termination of the client). Refer to the description of the **quitTimeout** resource and the WM_PROTOCOLS property.

f.lower [*-client* | **within** | **freeFamily**]

This function lowers a primary window to the bottom of the global window stack (where it obscures no other window) and lowers the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive.

The *client* argument indicates the name or class of a client to lower. If the *client* argument is not specified, the context that the function was invoked in indicates the window or icon to lower.

Specifying **within** lowers the secondary window within the family (staying above the parent) but does not lower the client family in the global window stack.

Specifying **freeFamily** lowers the window to the bottom of the global windows stack from its local family stack.

f.maximize
This function causes a client window to be displayed with its maximum size.

f.menu This function associates a cascading (pull-right) menu with a menu pane entry or a menu with a button or key binding. The *menu_name* function argument identifies the menu to be used.

f.minimize
This function causes a client window to be minimized (iconified). When a window is minimized when no icon box is used, its icon is placed on the bottom of the window stack (so that it obscures no other window). If an icon box is used, the client's icon changes to its iconified form inside the icon box. Secondary windows (that is, transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move This function causes a client window to be interactively moved.

f.next_cmap
This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [icon | window | transient]
This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if **keyboardFocusPolicy** is not explicit. The keyboard input focus is moved only to windows that do not have an associated secondary window that is application modal. If the **transient** argument is specified, transient (secondary) windows are traversed (otherwise, if only **window** is specified, traversal is done only to the window that last had focus in a transient group). If an **icon** function argument is specified, the function applies only to icons. If a **window** function argument is specified, the function applies only to windows.

f.nop This function does nothing.

f.normalize

This function causes a client window to be displayed with its normal size. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.normalize_and_raise

This function causes the corresponding client window to be displayed with its normal size and raised to the top of the window stack. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.pack_icons

This function is used to re-layout icons (based on the layout policy being used) on the root window or in the icon box. In general this causes icons to be "packed" into the icon grid.

f.pass_keys

This function is used to enable/disable (toggle) processing of key bindings for window manager functions. When it disables key binding processing, all keys are passed on to the window with the keyboard input focus and no window manager functions are invoked. If the **f.pass_keys** function is invoked with a key binding to disable key-binding processing, the same key binding can be used to enable key-binding processing.

f.post_wmenu

This function is used to post the window menu. If a key is used to post the window menu and a window menu button is present, the window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window. If no window menu button is present, the window menu is placed at the top-left corner of the client window.

f.prev_cmap

This function installs the previous colormap in the list of colormaps for the window with the colormap focus.

f.prev_key [icon | window | transient]

This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if **keyboardFocusPolicy** is not explicit. The keyboard input focus is moved only to windows that do not have an associated secondary window that is application modal. If the **transient** argument is specified, transient (secondary) windows are traversed (otherwise, if only **window** is specified, traversal is done only to the last

focused window in a transient group). If an **icon** function argument is specified, the function applies only to icons. If an **window** function argument is specified, the function applies only to windows.

f.quit_mwm
This function terminates **mwm** (but *not* the X window system).

f.raise [*-client* | **within** | **freeFamily**]
This function raises a primary window to the top of the global window stack (where it is obscured by no other window) and raises the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive.

The *client* argument indicates the name or class of a client to lower. If the *client* is not specified, the context that the function was invoked in indicates the window or icon to lower.

Specifying **within** raises the secondary window within the family but does not raise the client family in the global window stack.

Specifying **freeFamily** raises the window to the top of its local family stack and raises the family to the top of the global window stack.

f.raise_lower [**within** | **freeFamily**]
This function raises a primary window to the top of the global window stack if it is partially obscured by another window; otherwise, it lowers the window to the bottom of the window stack. The arguments to this function are mutually exclusive.

Specifying **within** raises a secondary window within the family (staying above the parent window), if it is partially obscured by another window in the application's family; otherwise, it lowers the window to the bottom of the family stack. It has no effect on the global window stacking order.

Specifying **freeFamily** raises the window to the top of its local family stack, if obscured by another window, and raises the family to the top of the global window stack; otherwise, it lowers the window to the bottom of its local family stack and lowers the family to the bottom of the global window stack.

f.refresh This function causes all windows to be redrawn.

f.refresh_win
This function causes a client window to be redrawn.

f.resize This function causes a client window to be interactively resized.

f.restore This function restores the previous state of an icon's associated window. If a maximized window is iconified, then **f.restore** restores it to its maximized state. If a normal window is iconified, then **f.restore** restores it to its normalized state.

f.restore_and_raise
This function restores the previous state of an icon's associated window and raises the window to the top of the window stack. If a maximized window is iconified, then **f.restore_and_raise** restores it to its maximized state and raises it to the top of the window stack. If a normal window is iconified, then **f.restore_and_raise** restores it to its normalized state and raises it to the top of the window stack.

f.restart This function causes **mwm** to be restarted (effectively terminated and re-executed).

f.screen [**next** | **prev** | **back** *screen_number*]
This function causes the pointer to warp to a specific screen number or to the **next**, **previous**, or last visited (**back**) screen. The arguments to this function are mutually exclusive.

The *screen_number* argument indicates the screen number that the pointer is to warp to. Screens are numbered starting from screen 0.

Specifying **next** cause the pointer to warp to the next managed screen (skipping over any unmanaged screens).

Specifying **prev** cause the pointer to warp to the previous managed screen (skipping over any unmanaged screens).

Specifying **back** cause the pointer to warp to the last visited screen.

f.send_msg *message_number*
This function sends a client message of the type _MOTIF_WM_MESSAGES with the *message_type* indicated by the *message_number* function argument. The client message is sent only if *message_number* is included in the client's _MOTIF_WM_MESSAGES property. A menu item label is grayed out if the menu item is used to do an **f.send_msg** of a message that is not included in the client's _MOTIF_WM_MESSAGES property.

f.separator
This function causes a menu separator to be put in the menu pane at the specified location (the label is ignored).

f.set_behavior
This function causes the window manager to restart with the default behavior (if a custom behavior is configured) or revert to the custom behavior. By default this is bound to **<Shift> <Ctrl> <Meta> <*Key*>!**.

f.title This function inserts a title in the menu pane at the specified location.

Each function may be constrained as to which resource types can specify the function (for example, menu pane) and also what context the function can be used in (for example, the function is done to the selected client window). Function contexts are

root No client window or icon has been selected as an object for the function.

window A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the window is in its normalized state (for example, **f.maximize**) or its maximized state (for example, **f.normalize**).

icon An icon has been selected as an object for the function.

If a function's context has been specified as **icon|window** and the function is invoked in an icon box, the function applies to the icon box, not to the icons inside.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply, the function is treated as **f.nop**. The following table indicates the resource types and function contexts in which window manager functions apply.

Function	Contexts	Resources
f.beep	root, icon, window	button, key, menu
f.circle_down	root, icon, window	button, key, menu
f.circle_up	root, icon, window	button, key, menu
f.exec	root, icon, window	button, key, menu
f.focus_color	root, icon, window	button, key, menu
f.focus_key	root, icon, window	button, key, menu
f.kill	icon, window	button, key, menu
f.lower	icon, window	button, key, menu
f.maximize	icon, window(normal)	button, key, menu
f.menu	root, icon, window	button, key, menu
f.minimize	window	button, key, menu
f.move	icon, window	button, key, menu
f.next_cmap	root, icon, window	button, key, menu
f.next_key	root, icon, window	button, key, menu
f.nop	root, icon, window	button, key, menu
f.normalize	icon, window(maximized)	button, key, menu
f.normalize_and_raise	icon, window	button, key, menu
f.pack_icons	root, icon, window	button, key, menu
f.pass_keys	root, icon, window	button, key, menu
f.post_wmenu	root, icon, window	button, key
f.prev_cmap	root, icon, window	button, key, menu
f.prev_key	root, icon, window	button, key, menu
f.quit_mwm	root, icon, window	button, key, menu (root only)
f.raise	icon, window	button, key, menu
f.raise_lower	icon, window	button, key, menu
f.refresh	root, icon, window	button, key, menu
f.refresh_win	window	button, key, menu
f.resize	window	button, key, menu
f.restore	icon, window	button, key, menu

Function	Contexts	Resources
f.restore_and_raise	icon, window	button, key, menu
f.restart	root, icon, window	button, key, menu (root only)
f.screen	root, icon, window	button, key, menu
f.send_msg	icon, window	button, key, menu
f.separator	root, icon, window	menu
f.set_behavior	root, icon, window	button, key, menu
f.title	root, icon, window	menu

Window Manager Event Specification

Events are indicated as part of the specifications for button and key-binding sets, and menu panes.

Button events have the following syntax:

button = [*modifier_list*]<*button_event_name*>
modifier_list = *modifier_name* {*modifier_name*}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the button event occurs). The following table indicates the values that can be used for *modifier_name*. The **<Alt>** key is frequently labeled **<Extend>** or **<Meta>**. **<Alt>** and **<Meta>** can be used interchangeably in event specification.

Modifier	Description
<Ctrl>	Control Key
<Shift>	Shift Key
<Alt>	Alt/Meta Key
<Meta>	Meta/Alt Key
<Lock>	Lock Key
<Mod1>	Modifier1
<Mod2>	Modifier2
<Mod3>	Modifier3
<Mod4>	Modifier4
<Mod5>	Modifier5

The following table indicates the values that can be used for *button_event_name*.

Button	Description
Btn1Down	Button 1 Press
Btn1Up	Button 1 Release
Btn1Click	Button 1 Press and Release
Btn1Click2	Button 1 Double-Click
Btn2Down	Button 2 Press
Btn2Up	Button 2 Release
Btn2Click	Button 2 Press and Release
Btn2Click2	Button 2 Double-Click
Btn3Down	Button 3 Press
Btn3Up	Button 3 Release
Btn3Click	Button 3 Press and Release
Btn3Click2	Button 3 Double-Click
Btn4Down	Button 4 Press
Btn4Up	Button 4 Release
Btn4Click	Button 4 Press and Release
Btn4Click2	Button 4 Double-Click
Btn5Down	Button 5 Press
Btn5Up	Button 5 Release
Btn5Click	Button 5 Press and Release
Btn5Click2	Button 5 Double-Click

Key events that are used by the window manager for menu mnemonics and for binding to window manager functions are single key presses; key releases are ignored. Key events have the following syntax:

key = [*modifier_list*]<*key*>*key_name*
modifier_list = *modifier_name* {*modifier_name*}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). Modifiers for keys are the same as those that apply to buttons. The *key_name* is an X11 keysym name. Keysym names can be found in the **keysymdef.h** file (remove the **XK_** prefix).

Button Bindings

The **buttonBindings** resource value is the name of a set of button bindings that are used to configure window manager behavior. A window manager function can be done when a button press occurs with the pointer over a framed client window, an icon, or the root window. The context for indicating where the button press applies is also the context for invoking the window manager function when the button press is done (significant for functions that are context sensitive).

The button binding syntax is

Buttons *bindings_set_name*
{
button context function
button context function
.
.
button context function
}

The syntax for the *context* specification is

context = *object*[| *context*]
object = **root | icon | window | title | frame | border | app**

The context specification indicates where the pointer must be for the button binding to be effective. For example, a context of **window** indicates that the pointer must be over a client window or window management frame for the button binding to be effective. The **frame** context is for the window management frame around a client window (including the border and titlebar), the **border** context is for the border part of the window management frame (not including the titlebar), the **title** context is for the title area of the window management frame, and the **app** context is for the application window (not including the window management frame).

If an **f.nop** function is specified for a button binding, the button binding is not done.

Key Bindings

The **keyBindings** resource value is the name of a set of key bindings that are used to configure window manager behavior. A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings.

The key binding syntax is

Keys *bindings_set_name*
{
  key context function
  key context function
  .
  .
  key context function
}

If an **f.nop** function is specified for a key binding, the key binding is not done. If an **f.post_wmenu** or **f.menu** function is bound to a key, **mwm** will automatically use the same key for removing the menu from the screen after it has been popped up.

The *context* specification syntax is the same as for button bindings. For key bindings, the **frame**, **title**, **border**, and **app** contexts are equivalent to the **window** context. The context for a key event is the window or icon that has the keyboard input focus (**root** if no window or icon has the keyboard input focus).

Menu Panes

Menus can be popped up using the **f.post_wmenu** and **f.menu** window manager functions. The context for window manager functions that are done from a menu is **root**, **icon** or **window** depending on how the menu was popped up. In the case of the **window** menu or menus popped up with a key binding, the location of the keyboard input focus indicates the context. For menus popped up using a button binding, the context of the button binding is the context of the menu.

The menu pane specification syntax is

Menu *menu_name*
{
  label [*mnemonic*] [*accelerator*] *function*
  label [*mnemonic*] [*accelerator*] *function*
  .
  .
  label [*mnemonic*] [*accelerator*] *function*
}

Each line in the **Menu** specification identifies the label for a menu item and the function to be done if the menu item is selected. Optionally a menu button mnemonic and a menu button keyboard accelerator may be specified. Mnemonics are functional only when the menu is posted and keyboard traversal applies.

The *label* may be a string or a bitmap file. The label specification has the following syntax:

label = *text* | *bitmap_file*
bitmap_file = *@file_name*
text = *quoted_item* | *unquoted_item*

The string encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the **f.nop** function or an invalid function or a function that does not apply in the current context.

A **mnemonic** specification has the following syntax

mnemonic = *_character*

The first matching *character* in the label is underlined. If there is no matching *character* in the label, no mnemonic is registered with the window manager for that label. Although the *character* must exactly match a character in the label, the mnemonic does not execute if any modifier (such as Shift) is pressed with the character key.

The **accelerator** specification is a key event specification with the same syntax as is used for key bindings to window manager functions.

Environment

mwm uses the environment variable HOME for specifying the user's home directory.

mwm uses the environment variable LANG for specifying the user's choice of language for the **mwm** message catalog and the **mwm** resource description file.

mwm uses the environment variables XFILESEARCHPATH, XUSERFILESEARCHPATH, XAPPLRESDIR, XENVIRONMENT, LANG, and HOME in determining search paths for resource defaults files. **mwm** may also use XBMLANGPATH to search for bitmap files.

mwm reads the **$HOME/.motifbind** file if it exists to install a virtual key bindings property on the root window. For more information on the content of the **.motifbind** file, see **VirtualBindings(3X)**.

mwm uses the environment variable MWMSHELL (or SHELL, if MWMSHELL is not set), for specifying the shell to use when executing commands with the **f.exec** function.

Files

/usr/lib/X11/$LANG/system.mwmrc
/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/Mwm
$HOME/.Xdefaults
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
$HOME/.motifbind

Related Information

VendorShell(3X), **VirtualBindings(3X)**, **X(1)**, and **XmInstallImage(3X)**.

uil—The user interface language compiler

Synopsis

uil [*options*] *file*

Description

The **uil** command invokes the UIL compiler. The User Interface Language (UIL) is a specification language for describing the initial state of a user interface for a Motif application. The specification describes the objects (menus, dialog boxes, labels, push buttons, and so on) used in the interface and specifies the routines to be called when the interface changes state as a result of user interaction.

file Specifies the file to be compiled through the UIL compiler.

options Specifies one or more of the following options:

-I*pathname*
This option causes the compiler to look for include files in the directory specified if the include files have not been found in the paths that already were searched. Specify this option followed by a pathname, with no intervening spaces.

-m Machine code is listed. This directs the compiler to place in the listing file a description of the records that it added to the User Interface Database (UID). This helps you isolate errors. The default is no machine code.

-o *file* Directs the compiler to produce a UID. By default, UIL creates a UID with the name **a.uid**. The file specifies the filename for the UID. No UID is produced if the compiler issues any diagnostics categorized as error or severe.

-s Directs the compiler to set the locale before compiling any files. The locale is set in an implementation-dependent manner. On ANSI C-based systems, the locale is usually set by calling **setlocale(LC_ALL, "")**. If this option is not specified, the compiler does not set the locale.

-v *file* Directs the compiler to generate a listing. The file specifies the filename for the listing. If the **-v** option is not present, no listing is generated by the compiler. The default is no listing.

-w Specifies that the compiler suppress all warning and informational messages. If this option is not present, all messages are generated, regardless of the severity.

-wmd *file* Specifies a binary widget meta-language description file to be used in place of the default WML description.

For more information about UIL syntax, see the *OSF/Motif Programmer's Guide*.

Related Information

X(1X) and **Uil(3X)**.

xmbind—Configures virtual key bindings

Synopsis

xmbind [*options*] [*file*]

Description

xmbind is an X Window System client that configures the virtual key bindings for Motif applications. This action is performed by **mwm** at its startup, so the **xmbind** client is only needed when **mwm** is not in use, or when you want to change bindings without restarting **mwm**. If a file is specified, its contents are used as the virtual key bindings. If a file is not specified, the file **.motifbind** in the user's home directory is used. If this file is not found, **xmbind** loads the default virtual key bindings, as described in **VirtualBindings(3X)**.

Options

-display This option specifies the display to use; see X(1).

Related Information

VirtualBindings(3X) and **X(1X)**.

ApplicationShell—The ApplicationShell widget class

Synopsis

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
```

Description

ApplicationShell is used as the main top-level window for an application. An application should have more than one ApplicationShell only if it implements multiple logical applications.

Classes

ApplicationShell inherits behavior and resources from **Core**, **Composite**, **Shell**, **WMShell**, **VendorShell**, and **TopLevelShell**.

The class pointer is **applicationShellWidgetClass**.

The class name is **ApplicationShell**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

ApplicationShell Resource Set

Name Class	Default Type	Access
XmNargc XmCArgc	0 int	CSG
XmNargv XmCArgv	NULL String *	CSG

XmNargc Specifies the number of arguments given in the **XmNargv** resource. The function **XtInitialize** sets this resource on the shell widget instance it creates by using its parameters as the values.

XmNargv Specifies the argument list required by a session manager to restart the application if it is killed. This list should be updated at appropriate points by the application if a new state has been reached that can be directly restarted. The function **XtInitialize** sets this resource on the shell widget instance it creates by using its parameters as the values.

Inherited Resources

ApplicationShell inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the reference page for that superclass.

TopLevelShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNiconic XmCIconic	False Boolean	CSG
XmNiconName XmCIconName	NULL String	CSG
XmNiconNameEncoding XmCIconNameEncoding	dynamic Atom	CSG

VendorShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaudibleWarning XmCAudibleWarning	XmBELL unsigned char	CSG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNdefaultFontList XmCDefaultFontList	dynamic XmFontList	CG
XmNdeleteResponse XmCDeleteResponse	XmDESTROY unsigned char	CSG
XmNinputMethod XmCInputMethod	NULL String	CSG
XmNkeyboardFocusPolicy XmCKeyboardFocusPolicy	XmEXPLICIT unsigned char	CSG
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmwmDecorations XmCMwmDecorations	-1 int	CSG
XmNmwmFunctions XmCMwmFunctions	-1 int	CSG
XmNmwmInputMode XmCMwmInputMode	-1 int	CSG
XmNmwmMenu XmCMwmMenu	NULL String	CSG
XmNpreeditType XmCPreeditType	dynamic String	CSG
XmNshellUnitType XmCShellUnitType	XmPIXELS unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNuseAsyncGeometry XmCUseAsyncGeometry	False Boolean	CSG

WMShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbaseHeight XmCBaseHeight	XtUnspecifiedShellInt int	CSG
XmNbaseWidth XmCBaseWidth	XtUnspecifiedShellInt int	CSG
XmNheightInc XmCHeightInc	XtUnspecifiedShellInt int	CSG
XmNiconMask XmCIconMask	NULL Pixmap	CSG
XmNiconPixmap XmCIconPixmap	NULL Pixmap	CSG
XmNiconWindow XmCIconWindow	NULL Window	CSG
XmNiconX XmCIconX	-1 int	CSG
XmNiconY XmCIconY	-1 int	CSG
XmNinitialState XmCInitialState	NormalState int	CSG
XmNinput XmCInput	True Boolean	CSG
XmNmaxAspectX XmCMaxAspectX	XtUnspecifiedShellInt int	CSG
XmNmaxAspectY XmCMaxAspectY	XtUnspecifiedShellInt int	CSG
XmNmaxHeight XmCMaxHeight	XtUnspecifiedShellInt int	CSG
XmNmaxWidth XmCMaxWidth	XtUnspecifiedShellInt int	CSG
XmNminAspectX XmCMinAspectX	XtUnspecifiedShellInt int	CSG

Name Class	Default Type	Access
XmNminAspectY XmCMinAspectY	XtUnspecifiedShellInt int	CSG
XmNminHeight XmCMinHeight	XtUnspecifiedShellInt int	CSG
XmNminWidth XmCMinWidth	XtUnspecifiedShellInt int	CSG
XmNtitle XmCTitle	dynamic String	CSG
XmNtitleEncoding XmCTitleEncoding	dynamic Atom	CSG
XmNtransient XmCTransient	False Boolean	CSG
XmNwaitForWm XmCWaitForWm	True Boolean	CSG
XmNwidthInc XmCWidthInc	XtUnspecifiedShellInt int	CSG
XmNwindowGroup XmCWindowGroup	dynamic Window	CSG
XmNwinGravity XmCWinGravity	dynamic int	CSG
XmNwmTimeout XmCWmTimeout	5000 ms int	CSG

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	False Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	False Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for ApplicationShell.

Related Information

Composite(3X), **Core(3X)**, **Shell(3X)**, **WMShell(3X)**, **VendorShell(3X)**, and **TopLevelShell(3X)**.

Composite—The Composite widget class

Synopsis

#include <Xm/Xm.h>

Description

Composite widgets are intended to be containers for other widgets and can have an arbitrary number of children. Their responsibilities (implemented either directly by the widget class or indirectly by Intrinsics functions) include:

- Overall management of children from creation to destruction.
- Destruction of descendants when the composite widget is destroyed.
- Physical arrangement (geometry management) of a displayable subset of managed children.
- Mapping and unmapping of a subset of the managed children. Instances of composite widgets need to specify the order in which their children are kept. For example, an application may want a set of command buttons in some logical order grouped by function, and it may want buttons that represent filenames to be kept in alphabetical order.

Classes

Composite inherits behavior and resources from **Core**.

The class pointer is **compositeWidgetClass**.

The class name is **Composite**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

XmNchildren
A read-only list of the children of the widget.

XmNinsertPosition
Points to the **XtOrderProc** function described below.

XmNnumChildren
A read-only resource specifying the length of the list of children in **XmNchildren**.

The following procedure pointer in a composite widget instance is of type **XtOrderProc**:

Cardinal (* XtOrderProc) (*widget*)
Widget *w*;

w Specifies the widget.

Composite widgets that allow clients to order their children (usually homogeneous boxes) can call their widget instance's **insert_position** procedure from the class's **insert_child** procedure to determine where a new child should go in its children array. Thus, a client of a composite class can apply different sorting criteria to widget instances of the class, passing in a different **insert_position** procedure when it creates each composite widget instance.

The return value of the **insert_position** procedure indicates how many children should go before the widget. A value of 0 (zero) indicates that the widget should go before all other children; returning **num_children** indicates that it should go after all other children. The default **insert_position** function returns **num_children** and can be overridden by a specific composite widget's resource list or by the argument list provided when the composite widget is created.

Inherited Resources

Composite inherits behavior and resources from the superclass described in the following table. For a complete description of each resource, refer to the reference page for that superclass.

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for Composite.

Related Information

Core(3X).

Constraint—The Constraint widget class

Synopsis

#include <Xm/Xm.h>

Description

Constraint widgets maintain additional state data for each child. For example, client-defined constraints on the child's geometry may be specified.

When a constrained composite widget defines constraint resources, all of that widget's children inherit all of those resources as their own. These constraint resources are set and read just the same as any other resources defined for the child. This resource inheritance extends exactly one generation down, which means only the first-generation children of a constrained composite widget inherit the parent widget's constraint resources.

Because constraint resources are defined by the parent widgets and not the children, the child widgets never directly use the constraint resource data. Instead, the parents use constraint resource data to attach child-specific data to children.

Classes

Constraint inherits behavior and resources from **Composite** and **Core**.

The class pointer is **constraintWidgetClass**.

The class name is **Constraint**.

New Resources

Constraint defines no new resources.

Inherited Resources

Constraint inherits behavior and resources from **Composite** and **Core**. The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for Constraint.

Related Information

Composite(3X) and **Core(3X)**.

Core—The Core widget class

Synopsis

#include <Xm/Xm.h>

Description

Core is the Xt Intrinsic base class for windowed widgets. The **Object** and **RectObj** classes provide support for windowless widgets.

Classes

All widgets are built from **Core**.

The class pointer is **widgetClass**.

The class name is **Core**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

XmNaccelerators
Specifies a translation table that is bound with its actions in the context of a particular widget. The accelerator table can then be installed on some destination widget.

XmNancestorSensitive
Specifies whether the immediate parent of the widget receives input events. Use the function **XtSetSensitive** to change the argument to preserve data integrity (see **XmNsensitive**). For shells, the default is copied from the parent's **XmNancestorSensitive** resource if there is a parent; otherwise, it is True. For other widgets, the default is the bitwise AND of the parent's **XmNsensitive** and **XmNancestorSensitive** resources.

XmNbackground
Specifies the background color for the widget.

XmNbackgroundPixmap
Specifies a pixmap for tiling the background. The first tile is placed at the upper left corner of the widget's window.

XmNborderColor
Specifies the color of the border in a pixel value.

XmNborderPixmap
Specifies a pixmap to be used for tiling the border. The first tile is placed at the upper left corner of the border.

XmNborderWidth
Specifies the width of the border that surrounds the widget's window on all four sides. The width is specified in pixels. A width of zero means that no border shows.

XmNcolormap
Specifies the colormap that is used for conversions to the type **Pixel** for this widget instance. When this resource is changed, previously generated pixel values are not affected, but newly generated values are in the new colormap. For shells without parents, the default is the default colormap of the widget's screen. Otherwise, the default is copied from the parent.

XmNdepth
Specifies the number of bits that can be used for each pixel in the widget's window. Applications should not change or set the value of this resource as it is set by the Xt Intrinsics when the widget is created. For shells without parents, the default is the default depth of the widget's screen. Otherwise, the default is copied from the parent.

XmNdestroyCallback
Specifies a list of callbacks that is called when the widget is destroyed.

XmNheight
Specifies the inside height (excluding the border) of the widget's window.

XmNinitialResourcesPersistent
Specifies whether or not resources are reference counted. If the value is True when the widget is created, the resources referenced by the widget are not reference counted, regardless of how the resource type converter is registered. An application that expects to destroy the widget and wants to have resources deallocated should specify a value of False. The default is True, implying an assumption that the widget will not be destroyed during the life of the application.

XmNmappedWhenManaged
If this resource is set to True, it maps the widget (makes it visible) as soon as it is both realized and managed. If this resource is set to False, the client is responsible for mapping and unmapping the widget. If the value is changed from True to False after the widget has been realized and managed, the widget is unmapped.

XmNscreen
Specifies the screen on which a widget instance resides. It is read only. When the Toolkit is initialized, the top-level widget obtains its default value from the default screen of the display. Otherwise, the default is copied from the parent.

XmNsensitive
Determines whether a widget receives input events. If a widget is sensitive, the Xt Intrinsics' Event Manager dispatches to the widget all keyboard, mouse button, motion, window enter/leave, and focus events. Insensitive widgets do not receive these events. Use the function **XtSetSensitive** to change the sensitivity argument. Using **XtSetSensitive** ensures that if a parent widget has **XmNsensitive** set to False, the ancestor-sensitive flag of all its children is appropriately set.

XmNtranslations
Points to a translations list. A translations list is a list of events and actions that are to be performed when the events occur.

XmNwidth
Specifies the inside width (excluding the border) of the widget's window.

XmNx Specifies the x-coordinate of the upper left outside corner of the widget's window. The value is relative to the upper left inside corner of the parent window.

XmNy Specifies the y-coordinate of the upper left outside corner of the widget's window. The value is relative to the upper left inside corner of the parent window.

Translations
There are no translations for Core.

Related Information

Object(3X) and **RectObj(3X)**.

MrmCloseHierarchy—Closes a UID hierarchy

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmCloseHierarchy(hierarchy_id)
        MrmHierarchy hierarchy_id;
```

Description

The **MrmCloseHierarchy** function closes a UID hierarchy previously opened by **MrmOpenHierarchyPerDisplay**. All files associated with the hierarchy are closed by the Motif Resource Manager (MRM) and all associated memory is returned.

hierarchy_id Specifies the ID of a previously opened UID hierarchy. The *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

Return Value

This function returns one of these status return constants:

MrmSUCCESS The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmFAILURE The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3X).

MrmFetchBitmapLiteral—Fetches a bitmap literal from a hierarchy

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchBitmapLiteral(hierarchy_id, index, screen, display, pixmap_return,
 width, height)
        MrmHierarchy hierarchy_id;
        String       index;
        Screen       *screen;
        Display      *display;
        Pixmap       *pixmap_return;
        Dimension    *width;
        Dimension    *height;
```

Description

The **MrmFetchBitmapLiteral** function fetches a bitmap literal from an MRM hierarchy, and converts the bitmap literal to an X pixmap of depth 1. The function returns this pixmap and its width and height.

hierarchy_id
: Specifies the ID of the UID hierarchy that contains the specified icon literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index
: Specifies the UIL name of the bitmap literal to fetch.

screen
: Specifies the screen used for the pixmap. The *screen* argument specifies a pointer to the Xlib structure **Screen** which contains the information about that screen and is linked to the **Display** structure. For more information on the **Display** and **Screen** structures, see the Xlib function **XOpenDisplay** and the associated screen information macros.

display
: Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

pixmap_return
: Returns the resulting X pixmap value.

width
: Specifies a pointer to the width of the pixmap.

height
: Specifies a pointer to the height of the pixmap.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS
The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND
The bitmap literal was not found in the hierarchy.

MrmWRONG_TYPE
The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE
The function failed.

Related Information

MrmFetchIconLiteral(3X), **MrmFetchLiteral(3X)**, and **XOpenDisplay(3X)**.

MrmFetchColorLiteral—Fetches a named color literal from a UID file

Synopsis

```
#include <Mrm/MrmPublic.h>

int MrmFetchColorLiteral(hierarchy_id, index, display, colormap_id, pixel)
        MrmHierarchy hierarchy_id;
        String       index;
        Display      *display;
        Colormap     colormap_id;
        Pixel        *pixel;
```

Description

The **MrmFetchColorLiteral** function fetches a named color literal from a UID file, and converts the color literal to a pixel color value.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the color literal to fetch. You must define this name in UIL as an exported value.

display Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

colormap_id Specifies the ID of the color map. If *colormap_id* is NULL, the default color map is used.

pixel Returns the ID of the color literal.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND The color literal was not found in the UIL file.

MrmWRONG_TYPE
The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE The function failed.

Related Information

MrmFetchBitmapLiteral(3X), **MrmOpenHierarchyPerDisplay(3X)**, **MrmFetchIconLiteral(3X)**, **MrmFetchLiteral(3X)**, and **XOpenDisplay(3X)**.

MrmFetchIconLiteral—Fetches an icon literal from a hierarchy

Synopsis

```
#include <Mrm/MrmPublic.h>

int MrmFetchIconLiteral(hierarchy_id, index, screen, display, fgpix, bgpix, pixmap)
        MrmHierarchy hierarchy_id;
        String       index;
        Screen       *screen;
        Display      *display;
        Pixel        fgpix;
        Pixel        bgpix;
        Pixmap       *pixmap;
```

Description

The **MrmFetchIconLiteral** function fetches an icon literal from an MRM hierarchy, and converts the icon literal to an X pixmap.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified icon literal. The *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the icon literal to fetch.

screen Specifies the screen used for the pixmap. The *screen* argument specifies a pointer to the Xlib structure **Screen**, which contains the information about that screen and is linked to the **Display** structure. For more information on the **Display** and **Screen** structures, see the Xlib function **XOpenDisplay** and the associated screen information macros.

display Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

fgpix Specifies the foreground color for the pixmap.

bgpix Specifies the background color for the pixmap.

pixmap Returns the resulting X pixmap value.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND The icon literal was not found in the hierarchy.

MrmWRONG_TYPE
The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE The function failed.

Related Information

MrmFetchBitmapLiteral(3X), **MrmOpenHierarchyPerDisplay(3X)**, **MrmFetchLiteral(3X)**, **MrmFetchColorLiteral(3X)**, and **XOpenDisplay(3X)**.

MrmFetchLiteral—Fetches a literal from a UID file

Synopsis

```
#include <Mrm/MrmPublic.h>

int MrmFetchLiteral(hierarchy_id, index, display, value, type)
        MrmHierarchy hierarchy_id;
        String       index;
        Display      *display;
        XtPointer    *value;
        MrmCode      *type;
```

Description

The **MrmFetchLiteral** function reads and returns the value and type of a literal (named value) that is stored as a public resource in a single UID file. This function returns a pointer to the value of the literal. For example, an integer is always returned as a pointer to an integer, and a string is always returned as a pointer to a string.

Applications should not use **MrmFetchLiteral** for fetching icon or color literals. If this is attempted, **MrmFetchLiteral** returns an error.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the literal (pixmap) to fetch. You must define this name in UIL as an exported value.

display Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the **Display** structure, see the Xlib function **XOpenDisplay**.

value Returns the ID of the named literal's value.

type Returns the named literal's data type. Types are defined in the include file **Mrm/MrmPublic.h**.

Return Value

This function returns one of these status return constants:

MrmSUCCESS The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND The literal was not found in the UIL file.

MrmWRONG_TYPE
The caller tried to fetch a literal of a type not supported by this function.

MrmFAILURE The function failed.

Related Information

MrmFetchBitmapLiteral(3X), **MrmOpenHierarchyPerDisplay(3X)**, **MrmFetchIconLiteral(3X)**, **MrmFetchColorLiteral(3X)**, and **XOpenDisplay(3X)**.

MrmFetchSetValues—Fetches the values to be set from literals stored in UID files

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchSetValues(hierarchy_id, widget, args, num_args)
        MrmHierarchy hierarchy_id;
        Widget       widget;
        ArgList      args;
        Cardinal     num_args;
```

Description

The **MrmFetchSetValues** function is similar to **XtSetValues**, except that the values to be set are defined by the UIL named values that are stored in the UID hierarchy. **MrmFetchSetValues** fetches the values to be set from literals stored in UID files.

hierarchy_id Specifies the ID of the UID hierarchy that contains the specified literal. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

widget Specifies the widget that is modified.

args Specifies an argument list that identifies the widget arguments to be modified as well as the index (UIL name) of the literal that defines the value for that argument. The name part of each argument (*args[n].name*) must begin with the string **XmN** followed by the name that uniquely identifies this attribute tag. For example, **XmNwidth** is the attribute name associated with the core argument *width*. The value part (*args[n].value*) must be a string that gives the index (UIL name) of the literal. You must define all literals in UIL as exported values.

num_args Specifies the number of entries in *args*.

This function sets the values on a widget, evaluating the values as public literal resource references resolvable from a UID hierarchy. Each literal is fetched from the hierarchy, and its value is modified and converted as required. This value is then placed in the argument list and used as the actual value for an **XtSetValues** call. **MrmFetchSetValues** allows a widget to be modified after creation using UID file values the same way creation values are used in **MrmFetchWidget**.

As in **MrmFetchWidget**, each argument whose value can be evaluated from the UID hierarchy is set in the widget. Values that are not found or values in which conversion errors occur are not modified.

Each entry in the argument list identifies an argument to be modified in the widget. The name part identifies the tag, which begins with **XmN**. The value part must be a string whose value is the index of the literal. Thus, the following code would modify the label resource of the widget to have the value of the literal accessed by the index **OK_button_label** in the hierarchy:

```
args[n].name = XmNlabel;
args[n].value = "OK_button_label";
```

Return Value

This function returns one of the following status return constants:

MrmSUCCESS The function executed successfully.

MrmPARTIAL_SUCCESS
At least one literal was successfully fetched.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmFAILURE The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3X), **XtSetValues(3X)**.

MrmFetchWidget—Fetches and creates an indexed (UIL named) application widgets and its children

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchWidget(hierarchy_id, index, parent_widget, widget, class)
        MrmHierarchy hierarchy_id;
        String       index;
        Widget       parent_widget;
        Widget       *widget;
        MrmType      *class;
```

Description

The **MrmFetchWidget** function fetches and creates an indexed application widget and its children. The indexed application widget is any widget that is named in UIL. In fetch operations, the fetched widget's subtree is also fetched and created. This widget must not appear as the child of a widget within its own subtree. **MrmFetchWidget** does not execute **XtManageChild** for the newly created widget.

hierarchy_id Specifies the ID of the **UID** hierarchy that contains the interface definition. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the widget to fetch.

parent_widget
Specifies the parent widget ID.

widget Returns the widget ID of the created widget.

class Returns the class code identifying MRM's widget class. The widget class code for the main window widget, for example, is **MRMwcMainWindow**. Literals identifying MRM widget class codes are defined in **Mrm.h**.

An application can fetch any named widget in the **UID** hierarchy using **MrmFetchWidget**. **MrmFetchWidget** can be called at any time to fetch a widget that was not fetched at application startup. **MrmFetchWidget** can be used to defer fetching pop-up widgets until they are first referenced (presumably in a callback), and then used to fetch them once.

MrmFetchWidget can also create multiple instances of a widget (and its subtree). In this case, the **UID** definition functions as a template; a widget definition can be fetched any number of times. An application can use this template to make multiple instances of a widget, for example, in a dialog box box or menu.

The index (UIL name) that identifies the widget must be known to the application.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS The function executed successfully.

MrmBAD_HIERARCHY
The hierarchy ID was invalid.

MrmNOT_FOUND The widget was not found in UID hierarchy.

MrmFAILURE The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3X), **MrmFetchWidgetOverride(3X)**.

MrmFetchWidgetOverride—Fetches any indexed (UIL named) application widget. It overrides the arguments specified for this application widget in UIL

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmFetchWidgetOverride(*hierarchy_id, index, parent_widget, override_name, override_args, override_num_args, widget, class*)

MrmHierarchy *hierarchy_id*;
String *index*;
Widget *parent_widget*;
String *override_name*;
ArgList *override_args*;
Cardinal *override_num_args*;
Widget **widget*;
MrmType **class*;

Description

The **MrmFetchWidgetOverride** function is the extended version of **MrmFetchWidget**. It is identical to **MrmFetchWidget**, except that it allows the caller to override the widget's name and any arguments that **MrmFetchWidget** would otherwise retrieve from the UID file or one of the defaulting mechanisms. That is, the override argument list is not limited to those arguments in the UID file.

The override arguments apply only to the widget fetched and returned by this function. Its children (subtree) do not receive any override parameters.

hierarchy_id Specifies the ID of the UID hierarchy that contains the interface definition. The value of *hierarchy_id* was returned in a previous call to **MrmOpenHierarchyPerDisplay**.

index Specifies the UIL name of the widget to fetch.

parent_widget
Specifies the parent widget ID.

override_name
Specifies the name to override the widget name. Use a NULL value if you do not want to override the widget name.

override_args Specifies the override argument list, exactly as given to **XtCreateWidget** (conversion complete and so forth). Use a NULL value if you do not want to override the argument list.

override_num_args
Specifies the number of arguments in *override_args*.

widget	Returns the widget ID of the created widget.
class	Returns the class code identifying MRM's widget class. Literals identifying MRM widget class codes are defined in the include file **Mrm/MrmPublic.h**.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS	The function executed successfully.
MrmBAD_HIERARCHY	The hierarchy ID was invalid.
MrmNOT_FOUND	The widget was not found in UID hierarchy.
MrmFAILURE	The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3X), **MrmFetchWidget(3X)**.

MrmInitialize—Prepares an application to use MRM widget-fetching facilities

Synopsis

void MrmInitialize()

Description

The **MrmInitialize** function must be called to prepare an application to use MRM widget-fetching facilities. You must call this function prior to fetching a widget. However, it is good programming practice to call **MrmInitialize** prior to performing any MRM operations.

MrmInitialize initializes the internal data structures that MRM needs to successfully perform type conversion on arguments and to successfully access widget creation facilities. An application must call **MrmInitialize** before it uses other MRM functions.

MrmOpenHierarchy—Allocates a hierarchy ID and opens all the UID files in the hierarchy

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchy(*num_files, file_names_list, ancillary_structures_list, hierarchy_id*)

MrmCount	*num_files*;
String	*file_names_list*[];
MrmOsOpenParamPtr	**ancillary_structures_list*;
MrmHierarchy	**hierarchy_id*;

Description

This routine is obsolete and exists for compatibility with previous releases. It is replaced by **MrmOpenHierarchyPerDisplay**. **MrmOpenHierarchy** is identical to **MrmOpenHierarchyPerDisplay** except that **MrmOpenHierarchy** does not take a *display* argument.

num_files Specifies the number of files in the name list.

file_names_list
Specifies an array of character strings that identify the UID files.

ancillary_structures_list
A list of operating-system-dependent ancillary structures corresponding to items such as filenames, clobber flags, and so forth. This argument should be NULL for most operations. If you need to reference this structure, see the definition of **MrmOsOpenParamPtr** in the **MrmPublic.h** header file for more information.

hierarchy_id Returns the search hierarchy ID. The search hierarchy ID identifies the list of UID files that MRM searches (in order) when performing subsequent fetch calls.

Each UID file string in *file_names_list* can specify either a full pathname or a filename. If a UID file string has a leading slash (/), it specifies a full pathname, and MRM opens the file as specified. Otherwise, the UID file string specifies a filename. In this case, MRM looks for the file along a search path specified by the **UIDPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set.

The **UIDPATH** environment variable specifies a search path and naming conventions associated with UID files. It can contain the substitution field **%U**, where the UID file string from the *file_names_list* argument to

MrmOpenHierarchyPerDisplay is substituted for **%U**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to **uid**. The entire path is first searched with **%S** mapped to **.uid**. If no file is found, is searched again with **%S** mapped to NULL.

If no display is set prior to calling this function, the result of this function's call to **XtResolvePathname** is undefined.

For example, the following **UIDPATH** value and **MrmOpenHierarchy** call cause MRM to open two separate UID files:

```
UIDPATH=/uidlib/%L/%U.uid:/uidlib/%U/%L
  static char *uid_files[] = {"/usr/users/me/test.uid", "test2"};
  MrmHierarchy  *Hierarchy_id;
  MrmOpenHierarchy((MrmCount)2,uid_files, NULL, Hierarchy_id)
```

MRM opens the first file, **/usr/users/me/test.uid**, as specified in the *file_names_list* argument to **MrmOpenHierarchy**, because the UID file string in the *file_names_list* argument specifies a full pathname. MRM looks for the second file, **test2**, first as **/uidlib/%L/test2.uid** and second as **/uidlib/test2/%L**, where the display's language string is substituted for **%L**.

After **MrmOpenHierarchy** opens the UID hierarchy, you should not delete or modify the UID files until you close the UID hierarchy by calling **MrmCloseHierarchy**.

If **UIDPATH** is not set but the environment variable **XAPPLRESDIR** is set, MRM searches the following pathnames:

- %U%S
- $XAPPLRESDIR/%L/uid/%N/%U%S
- $XAPPLRESDIR/%l/uid/%N/%U%S
- $XAPPLRESDIR/uid/%N/%U%S
- $XAPPLRESDIR/%L/uid/%U%S
- $XAPPLRESDIR/%l/uid/%U%S
- $XAPPLRESDIR/uid/%U%S
- $HOME/uid/%U%S
- $HOME/%U%S
- /usr/lib/X11/%L/uid/%N/%U%S
- /usr/lib/X11/%l/uid/%N/%U%S

- /usr/lib/X11/uid/%N/%U%S
- /usr/lib/X11/%L/uid/%U%S
- /usr/lib/X11/%l/uid/%U%S
- /usr/lib/X11/uid/%U%S
- /usr/include/X11/uid/%U%S

If neither **UIDPATH** nor **XAPPLRESDIR** is set, MRM searches the following pathnames:

- %U%S
- HOME/%L/uid/%N/%U%S
- HOME/%l/uid/%N/%U%S
- $HOME/uid/%N/%U%S
- $HOME/%L/uid/%U%S
- $HOME/%l/uid/%U%S
- $HOME/uid/%U%S
- $HOME/%U%S
- /usr/lib/X11/%L/uid/%N/%U%S
- /usr/lib/X11/%l/uid/%N/%U%S
- /usr/lib/X11/uid/%N/%U%S
- /usr/lib/X11/%L/uid/%U%S
- /usr/lib/X11/%l/uid/%U%S
- /usr/lib/X11/uid/%U%S
- /usr/include/X11/uid/%U%S

These paths are defaults that vendors may change. For example, a vendor may use different directories for **/usr/lib/X11** and **/usr/include/X11**.

The following substitutions are used in these paths:

%U The UID file string, from the *file_names_list* argument.

%N The class name of the application.

%L	The display's language string.
%l	The language component of the display's language string.
%S	The suffix to the filename. The entire path is first searched with a suffix of **.uil**. If no file is found, it is searched again with a NULL suffix.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS	The function executed successfully.
MrmNOT_FOUND	File not found.
MrmFAILURE	The function failed.

Related Information

MrmOpenHierarchyPerDisplay(3X) and **MrmCloseHierarchy(3X)**.

MrmOpenHierarchyPerDisplay—Allocates a hierarchy ID and opens all the UID files in the hierarchy

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchyPerDisplay (display, num_files, file_names_list,
ancillary_structures_list, hierarchy_id)
        Display             *display;
        MrmCount            num_files;
        String              file_names_list[];
        MrmOsOpenParamPtr   *ancillary_structures_list;
        MrmHierarchy        *hierarchy_id;
```

Description

MrmOpenHierarchyPerDisplay allows you to specify the list of UID files that MRM searches in subsequent fetch operations. All subsequent fetch operations return the first occurrence of the named item encountered while traversing the UID hierarchy from the first list element (UID file specification) to the last list element. This function also allocates a hierarchy ID and opens all the UID files in the hierarchy. It initializes the optimized search lists in the hierarchy. If **MrmOpenHierarchyPerDisplay** encounters any errors during its execution, any files that were opened are closed.

The application must call **XtAppInitialize** before calling **MrmOpenHierarchyPerDisplay**.

display Specifies the connection to the X server and the value to pass to **XtResolvePathname**. For more information on the Display structure, see the Xlib function **XOpenDisplay**.

num_files Specifies the number of files in the name list.

file_names_list
Specifies an array of character strings that identify the UID files.

ancillary_structures_list
A list of operating-system-dependent ancillary structures corresponding to items such as filenames, clobber flags, and so forth. This argument should be NULL for most operations. If you need to reference this structure, see the definition of **MrmOsOpenParamPtr** in the **MrmPublic.h** header file for more information.

hierarchy_id Returns the search hierarchy ID. The search hierarchy ID identifies the list of UID files that MRM searches (in order) when performing subsequent fetch calls.

Each UID file string in *file_names_list* can specify either a full pathname or a filename. If a UID file string has a leading / (slash), it specifies a full pathname, and MRM opens the file as specified. Otherwise, the UID file string specifies a filename. In this case MRM looks for the file along a search path specified by the **UIDPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set.

The **UIDPATH** environment variable specifies a search path and naming conventions associated with UID files. It can contain the substitution field **%U**, where the UID file string from the *file_names_list* argument to **MrmOpenHierarchyPerDisplay** is substituted for **%U**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to **uid**. The entire path is searched first with **%S** mapped to **.uid**. If no file is found, it is searched again with %S mapped to NULL. For example, the following **UIDPATH** value and **MrmOpenHierarchyPerDisplay** call cause MRM to open two separate UID files:

```
UIDPATH=/uidlib/%L/%U.uid:/uidlib/%U/%L
  static char *uid_files[] = {"/usr/users/me/test.uid", "test2"};
  MrmHierarchy  *Hierarchy_id;
  MrmOpenHierarchyPerDisplay((MrmCount)2,uid_files, NULL, Hierarchy_id)
```

MRM opens the first file, **/usr/users/me/test.uid**, as specified in the *file_names_list* argument to **MrmOpenHierarchyPerDisplay**, because the UID file string in the *file_names_list* argument specifies a full pathname. MRM looks for the second file, **test2**, first as **/uidlib/%L/test2.uid** and second as **/uidlib/test2/%L**, where the display's language string is substituted for **%L**.

After **MrmOpenHierarchyPerDisplay** opens the UID hierarchy, you should not delete or modify the UID files until you close the UID hierarchy by calling **MrmCloseHierarchy**.

If **UIDPATH** is not set, but the environment variable **XAPPLRESDIR** is set, MRM searches the following pathnames:

- %U%S
- $XAPPLRESDIR/%L/uid/%N/%U%S
- $XAPPLRESDIR/%l/uid/%N/%U%S
- $XAPPLRESDIR/uid/%N/%U%S

- $XAPPLRESDIR/%L/uid/%U%S
- $XAPPLRESDIR/%l/uid/%U%S
- $XAPPLRESDIR/uid/%U%S
- $HOME/uid/%U%S
- $HOME/%U%S
- /usr/lib/X11/%L/uid/%N/%U%S
- /usr/lib/X11/%l/uid/%N/%U%S
- /usr/lib/X11/uid/%N/%U%S
- /usr/lib/X11/%L/uid/%U%S
- /usr/lib/X11/%l/uid/%U%S
- /usr/lib/X11/uid/%U%S
- /usr/include/X11/uid/%U%S

If neither **UIDPATH** nor **XAPPLRESDIR** is set, MRM searches the following pathnames:

- %U%S
- $HOME/%L/uid/%N/%U%S
- $HOME/%l/uid/%N/%U%S
- $HOME/uid/%N/%U%S
- $HOME/%L/uid/%U%S
- $HOME/%l/uid/%U%S
- $HOME/uid/%U%S
- $HOME/%U%S
- /usr/lib/X11/%L/uid/%N/%U%S
- /usr/lib/X11/%l/uid/%N/%U%S
- /usr/lib/X11/uid/%N/%U%S
- /usr/lib/X11/%L/uid/%U%S

- /usr/lib/X11/%l/uid/%U%S
- /usr/lib/X11/uid/%U%S
- /usr/include/X11/uid/%U%S

These paths are defaults that vendors may change. For example, a vendor may use different directories for **/usr/lib/X11** and **/usr/include/X11**.

The following substitutions are used in these paths:

%U	The UID file string, from the *file_names_list* argument.
%N	The class name of the application.
%L	The display's language string.
%l	The language component of the display's language string.
%S	The suffix to the filename. The entire path is first searched with a suffix of **.uil**. If no file is found, it is searched again with a NULL suffix.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS	The function executed successfully.
MrmNOT_FOUND	File not found.
MrmFAILURE	The function failed.

Related Information

MrmCloseHierarchy(3X).

MrmRegisterClass—Saves the information needed for MRM to access the widget creation function for user-defined widgets

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterClass(*class_code, class_name, create_name, create_proc, class_record*)

```
MrmType      class_code;
String       class_name;
String       create_name;
Widget       (*create_proc) ();
WidgetClass  class_record;
```

Description

The **MrmRegisterClass** function allows MRM to access user-defined widget classes. This function registers the necessary information for MRM to create widgets of this class. You must call **MrmRegisterClass** prior to fetching any user-defined class widget.

MrmRegisterClass saves the information needed to access the widget creation function and to do type conversion of argument lists by using the information in MRM databases.

class_code This argument is ignored; it is present for compatibility with previous releases.

class_name This argument is ignored; it is present for compatibility with previous releases.

create_name Specifies the case-sensitive name of the low-level widget creation function for the class. An example from the Motif Toolkit is **XmCreateLabel**. Arguments are *parent_widget*, *name*, *override_arglist*, and *override_argcount*.

For user-defined widgets, *create_name* is the creation procedure in the UIL that defines this widget.

create_proc Specifies the address of the creation function that you named in *create_name*.

class_record Specifies a pointer to the class record.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS	The function executed successfully.
MrmFAILURE	The function failed.

MrmRegisterNames—Registers the values associated with the names referenced in UIL (for example, UIL callback function names or UIL identifier names)

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterNames(register_list, register_count)
        MrmRegisterArglist      register_list;
        MrmCount                register_count;
```

Description

The **MrmRegisterNames** function registers a vector of names and associated values for access in MRM. The values can be callback functions, pointers to user-defined data, or any other values. The information provided is used to resolve symbolic references occurring in UID files to their run-time values. For callbacks, this information provides the procedure address required by the Motif Toolkit. For names used as identifiers in UIL, this information provides any run-time mapping the application needs.

This function is similar to **MrmRegisterNamesInHierarchy**, except that the scope of the names registered by **MrmRegisterNamesInHierarchy** is limited to the hierarchy specified in the call to that function, whereas the names registered by **MrmRegisterNames** have global scope. When MRM looks up a name, it first tries to find the name among those registered for the given hierarchy. If that lookup fails, it tries to find the name among those registered globally.

register_list Specifies a list of name/value pairs for the names to be registered. Each name is a case-sensitive, NULL-terminated ASCII string. Each value is a 32-bit quantity, interpreted as a procedure address if the name is a callback function, and uninterpreted otherwise.

register_count
Specifies the number of entries in *register_list*.

The names in the list are case-sensitive. The list can be either ordered or unordered.

Callback functions registered through **MrmRegisterNames** can be either regular or creation callbacks. Regular callbacks have declarations determined by Motif Toolkit and user requirements. Creation callbacks have the same format as any other callback:

```
void CallBackProc(widget_id, tag, callback_data)
        Widget                  *widget_id;
        Opaque                  tag
        XmAnyCallbackStruct     *callback_data;
```

widget_id Specifies the widget ID associated with the widget performing the callback (as in any callback function).

tag Specifies the tag value (as in any callback function).

callback_data Specifies a widget-specific data structure. This data structure has a minimum of two members: event and reason. The reason member is always set to **MrmCR_CREATE**.

Note that the widget name and parent are available from the widget record accessible through *widget_id*.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS The function executed successfully.

MrmFAILURE The function failed.

MrmRegisterNamesInHierarchy—Registers the values associated with the names referenced in UIL within a single hierarchy (for example, UIL callback function names or UIL identifier names)

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterNamesInHierarchy(hierarchy_id, register_list, register_count)
        MrmHierarchy            hierarchy_id;
        MrmRegisterArglist      register_list;
        MrmCount                register_count;
```

Description

The **MrmRegisterNamesInHierarchy** function registers a vector of names and associated values for access in MRM. The values can be callback functions, pointers to user-defined data, or any other values. The information provided is used to resolve symbolic references occurring in UID files to their run-time values.For callbacks, this information provides the procedure address required by the Motif Toolkit. For names used as identifiers in UIL, this information provides any run-time mapping the application needs.

This function is similar to **MrmRegisterNames**, except that the scope of the names registered by **MrmRegisterNamesInHierarchy** is limited to the hierarchy specified by *hierarchy_id*, whereas the names registered by **MrmRegisterNames** have global scope. When MRM looks up a name, it first tries to find the name among those registered for the given hierarchy. If that lookup fails, it tries to find the name among those registered globally.

hierarchy_id Specifies the hierarchy with which the names are to be associated.

register_list Specifies a list of name/value pairs for the names to be registered. Each name is a case-sensitive, NULL-terminated ASCII string. Each value is a 32-bit quantity, interpreted as a procedure address if the name is a callback function, and uninterpreted otherwise.

register_count
Specifies the number of entries in *register_list.*

The names in the list are case-sensitive. The list can be either ordered or unordered.

Callback functions registered through **MrmRegisterNamesInHierarchy** can be either regular or creation callbacks. Regular callbacks have declarations determined by Motif Toolkit and user requirements.

Creation callbacks have the same format as any other callback:

void CallBackProc(*widget_id, tag, callback_data*)
Widget **widget_id*;
Opaque *tag*;
XmAnyCallbackStruct **callback_data*;

widget_id Specifies the widget ID associated with the widget performing the callback (as in any callback function).

tag Specifies the tag value (as in any callback function).

callback_data
Specifies a widget-specific data structure. This data structure has a minimum of two members: event and reason. The reason member is always set to **MrmCR_CREATE**.

Note that the widget name and parent are available from the widget record accessible through *widget_id*.

Return Value

This function returns one of the following status return constants:

MrmSUCCESS The function executed successfully.

MrmFAILURE The function failed.

Object—The Object widget class

Synopsis

#include <Xm/Xm.h>

Description

Object is never instantiated. Its sole purpose is as a supporting superclass for other widget classes.

Classes

The class pointer is **objectClass**.

The class name is **Object**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

XmNdestroyCallback

Specifies a list of callbacks that is called when the gadget is destroyed.

Translations

There are no translation for Object.

OverrideShell—The OverrideShell widget class

Synopsis

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
```

Description

OverrideShell is used for shell windows that completely bypass the window manager, for example, PopupMenu shells.

Classes

OverrideShell inherits behavior and resources from **Core**, **Composite**, and **Shell**.

The class pointer is **overrideShellWidgetClass**.

The class name is **OverrideShell**.

New Resources

OverrideShell defines no new resources, but overrides the **XmNoverrideRedirect** and **XmNsaveUnder** resources in the **Shell** class.

Inherited Resources

OverrideShell inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the reference page for that superclass.

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	True Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	True Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for OverrideShell.

Related Information

Composite(3X), **Core(3X)**, and **Shell(3X)**.

RectObj—The RectObj widget class

Synopsis

#include <Xm/Xm.h>

Description

RectObj is never instantiated. Its sole purpose is as a supporting superclass for other widget classes.

Classes

RectObj inherits behavior and a resource from **Object**.

The class pointer is **rectObjClass**.

The class name is **RectObj**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

RectObj Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

XmNancestorSensitive

Specifies whether the immediate parent of the gadget receives input events. Use the function **XtSetSensitive** if you are changing the argument to preserve data integrity (see **XmNsensitive**). The default is the bitwise AND of the parent's **XmNsensitive** and **XmNancestorSensitive** resources.

XmNborderWidth

Specifies the width of the border placed around the RectObj's rectangular display area.

XmNheight Specifies the inside height (excluding the border) of the RectObj's rectangular display area.

XmNsensitive

Determines whether a RectObj receives input events. If a RectObj is sensitive, the parent dispatches to the gadget all keyboard, mouse button, motion, window enter/leave, and focus events. Insensitive gadgets do not receive these events. Use the function **XtSetSensitive** to change the sensitivity argument. Using **XtSetSensitive** ensures that if a parent widget has **XmNsensitive** set to False, the ancestor-sensitive flag of all its children is appropriately set.

XmNwidth Specifies the inside width (excluding the border) of the RectObj's rectangular display area.

XmNx Specifies the x-coordinate of the upper left outside corner of the RectObj's rectangular display area. The value is relative to the upper left inside corner of the parent window.

XmNy Specifies the y-coordinate of the upper left outside corner of the RectObj's rectangular display area. The value is relative to the upper left inside corner of the parent window.

Inherited Resources

RectObj inherits behavior and a resource from **Object**. For a description of this resource, refer to the **Object** reference page.

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Translations

There are no translations for RectObj.

Related Information

Object(3X).

Shell—The Shell widget class

Synopsis

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
```

Description

Shell is a top-level widget (with only one managed child) that encapsulates the interaction with the window manager.

At the time the shell's child is managed, the child's width is used for both widgets if the shell is unrealized and no width has been specified for the shell. Otherwise, the shell's width is used for both widgets. The same relations hold for the height of the shell and its child.

Classes

Shell inherits behavior and resources from **Composite** and **Core**.

The class pointer is **shellWidgetClass**.

The class name is **Shell**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	False Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	False Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

XmNallowShellResize

Specifies that if this resource is False, the Shell widget instance returns **XtGeometryNo** to all geometry requests from its children.

XmNcreatePopupChildProc

Specifies the pointer to a function that is called when the Shell widget instance is popped up by **XtPopup**. The function creates the child widget when the shell is popped up instead of when the application starts up. This can be used if the child needs to be reconfigured each time the shell is popped up. The function takes one argument, the popup shell, and returns no result. It is called after the popup callbacks specified by **XmNpopupCallback**.

XmNgeometry

Specifies the desired geometry for the widget instance. This resource is examined only when the widget instance is unrealized and the number of its managed children is changed. It is used to change the values of the **XmNx**, **XmNy**, **XmNwidth**, and **XmNheight** resources.

XmNoverrideRedirect
If True, specifies that the widget instance is a temporary window which should be ignored by the window manager. Applications and users should not normally alter this resource.

XmNpopdownCallback
Specifies a list of callbacks that is called when the widget instance is popped down by **XtPopdown**.

XmNpopupCallback
Specifies a list of callbacks that is called when the widget instance is popped up by **XtPopup**.

XmNsaveUnder
If True, specifies that iit is desirable to save the contents of the screen beneath this widget instance, avoiding expose events when the instance is unmapped. This is a hint, and an implementation may save contents whenever it desires, including always or never.

XmNvisual Specifies the visual used in creating the widget.

Inherited Resources

Shell inherits behavior and resources from the superclass described in the following table. For a complete description of each resource, refer to the reference page for that superclass.

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for Shell.

Related Information

Composite(3X) and **Core(3X)**.

TopLevelShell—The TopLevelShell widget class

Synopsis

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
```

Description

TopLevelShell is used for normal top-level windows such as any additional top-level widgets an application needs.

Classes

TopLevelShell inherits behavior and resources from **Core**, **Composite**, **Shell**, **WMShell**, and **VendorShell**.

The class pointer is **topLevelShellWidgetClass**.

The class name is **TopLevelShell**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

TopLevelShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNiconic XmCIconic	False Boolean	CSG
XmNiconName XmCIconName	NULL String	CSG
XmNiconNameEncoding XmCIconNameEncoding	dynamic Atom	CSG

XmNiconic If True when the widget instance is realized, specifies that the widget instance indicates to the window manager that the application wishes to start as an icon, regardless of the **XmNinitialState** resource.

XmNiconName

Specifies the short form of the application name to be displayed by the window manager when the application is iconified.

XmNiconNameEncoding

Specifies a property type that represents the encoding of the **XmNiconName** string. If a language procedure has been set, the default is None; otherwise, the default is XA_STRING. When the widget is realized, if the value is None, the corresponding name is assumed to be in the current locale. The name is passed to **XmbTextListToTextProperty** with an encoding style of **XStdICCTextStyle**. The resulting encoding is STRING if the name is fully convertible to STRING, otherwise COMPOUND_TEXT. The values of the encoding resources are not changed; they remain None.

Inherited Resources

TopLevelShell inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the reference page for that superclass.

VendorShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaudibleWarning XmCAudibleWarning	XmBELL unsigned char	CSG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNdefaultFontList XmCDefaultFontList	dynamic XmFontList	CG
XmNdeleteResponse XmCDeleteResponse	XmDESTROY unsigned char	CSG
XmNinputMethod XmCInputMethod	NULL String	CSG
XmNkeyboardFocusPolicy XmCKeyboardFocusPolicy	XmEXPLICIT unsigned char	CSG
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmwmDecorations XmCMwmDecorations	-1 int	CSG
XmNmwmFunctions XmCMwmFunctions	-1 int	CSG
XmNmwmInputMode XmCMwmInputMode	-1 int	CSG
XmNmwmMenu XmCMwmMenu	NULL String	CSG
XmNpreeditType XmCPreeditType	dynamic String	CSG
XmNshellUnitType XmCShellUnitType	XmPIXELS unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNuseAsyncGeometry XmCUseAsyncGeometry	False Boolean	CSG

WMShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbaseHeight XmCBaseHeight	XtUnspecifiedShellInt int	CSG
XmNbaseWidth XmCBaseWidth	XtUnspecifiedShellInt int	CSG
XmNheightInc XmCHeightInc	XtUnspecifiedShellInt int	CSG
XmNiconMask XmCIconMask	NULL Pixmap	CSG
XmNiconPixmap XmCIconPixmap	NULL Pixmap	CSG
XmNiconWindow XmCIconWindow	NULL Window	CSG
XmNiconX XmCIconX	-1 int	CSG
XmNiconY XmCIconY	-1 int	CSG
XmNinitialState XmCInitialState	NormalState int	CSG
XmNinput XmCInput	True Boolean	CSG
XmNmaxAspectX XmCMaxAspectX	XtUnspecifiedShellInt int	CSG
XmNmaxAspectY XmCMaxAspectY	XtUnspecifiedShellInt int	CSG
XmNmaxHeight XmCMaxHeight	XtUnspecifiedShellInt int	CSG
XmNmaxWidth XmCMaxWidth	XtUnspecifiedShellInt int	CSG
XmNminAspectX XmCMinAspectX	XtUnspecifiedShellInt int	CSG

Name Class	Default Type	Access
XmNminAspectY XmCMinAspectY	XtUnspecifiedShellInt int	CSG
XmNminHeight XmCMinHeight	XtUnspecifiedShellInt int	CSG
XmNminWidth XmCMinWidth	XtUnspecifiedShellInt int	CSG
XmNtitle XmCTitle	dynamic String	CSG
XmNtitleEncoding XmCTitleEncoding	dynamic Atom	CSG
XmNtransient XmCTransient	False Boolean	CSG
XmNwaitForWm XmCWaitForWm	True Boolean	CSG
XmNwidthInc XmCWidthInc	XtUnspecifiedShellInt int	CSG
XmNwindowGroup XmCWindowGroup	dynamic Window	CSG
XmNwinGravity XmCWinGravity	dynamic int	CSG
XmNwmTimeout XmCWmTimeout	5000 ms int	CSG

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	False Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	False Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for TopLevelShell.

Related Information

Composite(3X), **Core(3X)**, **Shell(3X)**, **WMShell(3X)**, and **VendorShell(3X)**.

TransientShell—The TransientShell widget class

Synopsis

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
```

Description

TransientShell is used for shell windows that can be manipulated by the window manager, but are not allowed to be iconified separately. For example, DialogBoxes make no sense without their associated application. They are iconified by the window manager only if the main application shell is iconified.

Classes

TransientShell inherits behavior and resources from **Core**, **Composite**, **Shell**, **WMShell**, and **VendorShell**.

The class pointer is **transientShellWidgetClass**.

The class name is **TransientShell**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

In addition to these new resources, **TransientShell** overrides the **XmNsaveUnder** resource in **Shell** and the **XmNtransient** resource in **WMShell**.

TransientShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNtransientFor XmCTransientFor	NULL Widget	CSG

XmNtransientFor

Specifies a widget that the shell acts as a pop-up for. If this resource is NULL or is a widget that has not been realized, the **XmNwindowGroup** is used instead.

Inherited Resources

TransientShell inherits behavior and resources from the superclasses described in the following tables, which define sets of widget resources used by the programmer to specify data. For a complete description of each resource, refer to the reference page for that superclass.

The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

VendorShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaudibleWarning XmCAudibleWarning	XmBELL unsigned char	CSG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNdefaultFontList XmCDefaultFontList	dynamic XmFontList	CG
XmNdeleteResponse XmCDeleteResponse	XmDESTROY unsigned char	CSG
XmNinputMethod XmCInputMethod	NULL String	CSG
XmNkeyboardFocusPolicy XmCKeyboardFocusPolicy	XmEXPLICIT unsigned char	CSG
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmwmDecorations XmCMwmDecorations	-1 int	CSG
XmNmwmFunctions XmCMwmFunctions	-1 int	CSG
XmNmwmInputMode XmCMwmInputMode	-1 int	CSG
XmNmwmMenu XmCMwmMenu	NULL String	CSG
XmNpreeditType XmCPreeditType	dynamic String	CSG
XmNshellUnitType XmCShellUnitType	XmPIXELS unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNuseAsyncGeometry XmCUseAsyncGeometry	False Boolean	CSG

WMShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbaseHeight XmCBaseHeight	XtUnspecifiedShellInt int	CSG
XmNbaseWidth XmCBaseWidth	XtUnspecifiedShellInt int	CSG
XmNheightInc XmCHeightInc	XtUnspecifiedShellInt int	CSG
XmNiconMask XmCIconMask	NULL Pixmap	CSG
XmNiconPixmap XmCIconPixmap	NULL Pixmap	CSG
XmNiconWindow XmCIconWindow	NULL Window	CSG
XmNiconX XmCIconX	-1 int	CSG
XmNiconY XmCIconY	-1 int	CSG
XmNinitialState XmCInitialState	NormalState int	CSG
XmNinput XmCInput	True Boolean	CSG
XmNmaxAspectX XmCMaxAspectX	XtUnspecifiedShellInt int	CSG
XmNmaxAspectY XmCMaxAspectY	XtUnspecifiedShellInt int	CSG
XmNmaxHeight XmCMaxHeight	XtUnspecifiedShellInt int	CSG
XmNmaxWidth XmCMaxWidth	XtUnspecifiedShellInt int	CSG
XmNminAspectX XmCMinAspectX	XtUnspecifiedShellInt int	CSG

Name Class	Default Type	Access
XmNminAspectY XmCMinAspectY	XtUnspecifiedShellInt int	CSG
XmNminHeight XmCMinHeight	XtUnspecifiedShellInt int	CSG
XmNminWidth XmCMinWidth	XtUnspecifiedShellInt int	CSG
XmNtitle XmCTitle	dynamic String	CSG
XmNtitleEncoding XmCTitleEncoding	dynamic Atom	CSG
XmNtransient XmCTransient	True Boolean	CSG
XmNwaitForWm XmCWaitForWm	True Boolean	CSG
XmNwidthInc XmCWidthInc	XtUnspecifiedShellInt int	CSG
XmNwindowGroup XmCWindowGroup	dynamic Window	CSG
XmNwinGravity XmCWinGravity	dynamic int	CSG
XmNwmTimeout XmCWmTimeout	5000 ms int	CSG

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	False Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	True Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for TransientShell.

Related Information

Composite(3X), **Core(3X)**, **Shell(3X)**, **VendorShell(3X)**, and **WMShell(3X)**.

Uil—Invokes the UIL compiler from within an application

Synopsis

```
#include <uil/UilDef.h>

Uil_status_type Uil (command_desc, compile_desc, message_cb, message_data,
status_cb, status_data)
    Uil_command_type        *command_desc;
    Uil_compile_desc_type   *compile_desc;
    Uil_continue_type       (*message_cb) ();
    char                    *message_data;
    Uil_continue_type       (*status_cb) ();
    char                    *status_data;
```

Description

The **Uil** function provides a callable entry point for the UIL compiler. The **Uil** callable interface can be used to process a UIL source file and to generate UID files, as well as return a detailed description of the UIL source module in the form of a symbol table (parse tree).

command_desc
Specifies the **uil** command line.

compile_desc Returns the results of the compilation.

message_cb Specifies a callback function that is called when the compiler encounters errors in the UIL source.

message_data
Specifies user data that is passed to the message callback function (*message_cb*). Note that this argument is not interpreted by UIL, and is used exclusively by the calling application.

status_cb Specifies a callback function that is called to allow X applications to service X events such as updating the screen. This function is called at various check points, which have been hard coded into the UIL compiler. The *status_update_delay* argument in *command_desc* specifies the number of check points to be passed before the *status_cb* function is invoked.

status_data Specifies user data that is passed to the status callback function (*status_cb*). Note that this argument is not interpreted by the UIL compiler and is used exclusively by the calling application.

Following are the data structures **Uil_command_type** and **Uil_compile_desc_type**:

```
typedef struct Uil_command_type {
char *source_file;
    /* single source to compile */
char *resource_file; /* name of output file */
char *listing_file; /* name of listing file */
unsigned int *include_dir_count;
    /* number of dirs. in include_dir */
char *((*include_dir) []);
    /* dir. to search for include files */
unsigned listing_file_flag: 1;
    /* produce a listing */
unsigned resource_file_flag: 1;
    /* generate UID output */
unsigned machine_code_flag: 1;
    /* generate machine code */
unsigned report_info_msg_flag: 1;
    /* report info messages */
unsigned report_warn_msg_flag: 1;
    /* report warnings */
unsigned parse_tree_flag: 1;
    /* generate parse tree */
unsigned int status_update_delay;
    /* number of times a status point is */
    /* passed before calling status_cb */
    /* function 0 means called every time */
char *database;
    /* name of database file */
unsigned database_flag: 1;
    /* read a new database file */
unsigned use_setlocale_flag: 1;
    /* enable calls to setlocale */
};
```

```
typedef struct Uil_compile_desc_type {
unsigned int compiler_version;
    /* version number of compiler */
unsigned int data_version;
    /* version number of structures */
char *parse_tree_root; /* parse tree output */
unsigned int message_count [Uil_k_max_status+1];
/* array of severity counts */
};
```

Following is a description of the message callback function specified by *message_cb*:

Uil_continue_type (**message_cb*) (*message_data, message_number, severity, msg_buffer, src_buffer, ptr_buffer, loc_buffer, message_count*)

char	**message_data*;
int	*message_number*;
int	*severity*;
char	**msg_buffer*, **src_buffer*;
char	**ptr_buffer*, **loc_buffer*;
int	*message_count*[];

This function specifies a callback function that UIL invokes instead of printing an error message when the compiler encounters an error in the UIL source. The callback should return one of the following values:

Uil_k_terminate
: Terminate processing of the source file

Uil_k_continue
: Continue processing the source file

The arguments are

message_data
: Data supplied by the application as the *message_data* argument to the **Uil** function. UIL does not interpret this data in any way; it just passes it to the callback.

message_number
: An index into a table of error messages and severities for internal use by UIL.

severity
: An integer that indicates the severity of the error. The possible values are the status constants returned by the **Uil** function. See **Return Value** for more information.

msg_buffer A string that describes the error.

src_buffer A string consisting of the source line where the error occurred. This string is not always available. In this case, the argument is NULL.

ptr_buffer A string consisting of whitespace and a printing character in the character position corresponding to the column of the source line where the error occurred. This string may be printed beneath the source line to provide a visual indication of the column where the error occurred. This string is not always available. In this case, the argument is NULL.

loc_buffer A string identifying the line number and file of the source line where the error occurred. This is not always available; the argument is then NULL.

message_count
An array of integers containing the number of diagnostic messages issued thus far for each severity level. To find the number of messages issued for the current severity level, use the *severity* argument as the index into this array.

Following is a description of the status callback function specified by *status_cb*:

Uil_continue_type (**status_cb*) (*status_data, percent_complete, lines_processed, current_file, message_count*)

```
char        *status_data;
int         percent_complete;
int         lines_processed;
char        *current_file;
int         message_count[];
```

This function specifies a callback function that is invoked to allow X applications to service X events such as updating the screen. The callback should return one of the following values:

Uil_k_terminate
Terminate processing of the source file

Uil_k_continue
Continue processing the source file

The arguments are

status_data
Data supplied by the application as the *status_data* argument to the **Uil** function. UIL does not interpret this data in any way; it just passes it to the callback.

percent_complete
An integer indicating what percentage of the current source file has been processed so far.

lines_processed
An integer indicating how many lines of the current source file have been read so far.

current_file A string containing the pathname of the current source file.

message_count
An array of integers containing the number of diagnostic messages issued thus far for each severity level. To find the number of messages issued for a given severity level, use the severity level as the index into this array. The possible severity levels are the status constants returned by the **Uil** function. See **Return Value** for more information.

Return Value

This function returns one of the following status return constants:

Uil_k_success_status
The operation succeeded.

Uil_k_info_status The operation succeeded. An informational message is returned.

Uil_k_warning_status
The operation succeeded. A warning message is returned.

Uil_k_error_status The operation failed due to an error.

Uil_k_severe_status The operation failed due to an error.

Related Information

UilDumpSymbolTable(3X) and **uil(1X)**.

UilDumpSymbolTable—Dumps the contents of a named UIL symbol table to standard output

Synopsis

```
#include <uil/UilDef.h>

void UilDumpSymbolTable (root_ptr)
        sym_entry_type *root_ptr;
```

Description

The **UilDumpSymbolTable** function dumps the contents of a UIL symbol table pointer to standard output.

root_ptr Specifies a pointer to the the symbol table root entry. This value can be taken from the **parse_tree_root** part of the **Uil_compile_desc_type** data structure returned by **Uil**.

By following the link from the root entry, you can traverse the entire parse tree. Symbol table entries are in the following format:

hex.address
symbol.type
symbol.data
prev.source.position
source.position
modification.record

where:

hex.address Specifies the hexadecimal address of this entry in the symbol table.

symbol.type Specifies the type of this symbol table entry. Some possible types are **root**, **module**, **value**, **procedure**, and **widget**.

symbol.data Specifies data for the symbol table entry. The data varies with the type of the entry. Often it contains pointers to other symbol table entries, or the actual data for the data type.

prev.source.position
Specifies the end point in the source code for the previous source item.

source.position
Specifies the range of positions in the source code for this symbol.

The exact data structures for each symbol type are defined in the include file **UilSymDef.h**. Note that this file is automatically included when an application includes the file **UilDef.h**.

Related Information

Uil(3X)

VendorShell—The VendorShell widget class

Synopsis

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
```

Description

VendorShell is a Motif widget class used as a supporting superclass for all shell classes that are visible to the window manager and that are not override redirect. It contains resources that describe the MWM-specific look and feel. It also manages the MWM-specific communication needed by all VendorShell subclasses. See the **mwm** reference page for more information.

If an application uses the **XmNmwmDecorations**, **XmNmwmFunctions**, or **XmNmwmInputMode** resource, it should include the file **Xm/MwmUtil.h**.

Setting **XmNheight**, **XmNwidth**, or **XmNborderWidth** for either a VendorShell or its managed child usually sets that resource to the same value in both the parent and the child. When an off-the-spot input method exists, the height and width of the shell may be greater than those of the managed child in order to accommodate the input method. In this case, setting **XmNheight** or **XmNwidth** for the shell does not necessarily set that resource to the same value in the managed child, and setting **XmNheight** or **XmNwidth** for the child does not necessarily set that resource to the same value in the shell.

For the managed child of a VendorShell, regardless of the value of the shell's **XmNallowShellResize**, setting **XmNx** or **XmNy** sets the corresponding resource of the parent but does not change the child's position relative to the parent. **XtGetValues** for the child's **XmNx** or **XmNy** yields the value of the corresponding resource in the parent. The x and y-coordinates of the child's upper left outside corner relative to the parent's upper left inside corner are both 0 (zero) minus the value of **XmNborderWidth**.

Note that the *Inter-Client Communication Conventions Manual* (ICCM) allows a window manager to change or control the border width of a reparented top-level window.

Classes

VendorShell inherits behavior and resources from the **Core**, **Composite**, **Shell**, and **WMShell** classes.

The class pointer is **vendorShellWidgetClass**.

The class name is **VendorShell**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a subresource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a subresource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given subresource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

VendorShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaudibleWarning XmCAudibleWarning	XmBELL unsigned char	CSG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNdefaultFontList XmCDefaultFontList	dynamic XmFontList	CG
XmNdeleteResponse XmCDeleteResponse	XmDESTROY unsigned char	CSG
XmNinputMethod XmCInputMethod	NULL String	CSG
XmNkeyboardFocusPolicy XmCKeyboardFocusPolicy	XmEXPLICIT unsigned char	CSG
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmwmDecorations XmCMwmDecorations	-1 int	CSG
XmNmwmFunctions XmCMwmFunctions	-1 int	CSG
XmNmwmInputMode XmCMwmInputMode	-1 int	CSG
XmNmwmMenu XmCMwmMenu	NULL String	CSG
XmNpreeditType XmCPreeditType	dynamic String	CSG
XmNshellUnitType XmCShellUnitType	XmPIXELS unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNuseAsyncGeometry XmCUseAsyncGeometry	False Boolean	CSG

XmNaudibleWarning

Determines whether an action activates its associated audible cue. The possible values are **XmBELL** and **XmNONE**.

XmNbuttonFontList

Specifies the font list used for VendorShell's button descendants. If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNbuttonFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the BulletinBoard, VendorShell, or MenuShell widget class. If such an ancestor is found, **XmNbuttonFontList** is initialized to the **XmNbuttonFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

XmNdefaultFontList

Specifies a default font list for VendorShell's descendants. This resource is obsolete and exists for compatibility with earlier releases. It has been replaced by **XmNbuttonFontList**, **XmNlabelFontList**, and **XmNtextFontList**.

XmNdeleteResponse

Determines what action the shell takes in response to a WM_DELETE_WINDOW message. The setting can be one of three values: **XmDESTROY**, **XmUNMAP**, and **XmDO_NOTHING**. The resource is scanned, and the appropriate action is taken after the WM_DELETE_WINDOW callback list (if any) that is registered with the Protocol manager has been called.

XmNinputMethod

Specifies the string that sets the locale modifier for the input method.

XmNkeyboardFocusPolicy

Determines allocation of keyboard focus within the widget hierarchy rooted at this shell. The X keyboard focus must be directed to somewhere in the hierarchy for this client-side focus management to take effect. Possible values are **XmEXPLICIT**, specifying a click-to-type policy, and **XmPOINTER**, specifying a pointer-driven policy.

XmNlabelFontList

Specifies the font list used for VendorShell's label descendants (Labels and LabelGadgets). If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNlabelFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or

XmMenuShell widget class. If such an ancestor is found, **XmNlabelFontList** is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

XmNmwmDecorations

Specifies the decoration flags (specific decorations to add or remove from the window manager frame) for the **_MOTIF_WM_HINTS** property. If any decoration flags are specified by the **_MOTIF_WM_HINTS** property, only decorations indicated by both that property and the MWM **clientDecoration** and **transientDecoration** resources are displayed. If no decoration flags are specified by the **_MOTIF_WM_HINTS** property, decorations indicated by the MWM **clientDecoration** and **transientDecoration** resources are displayed. The default for the **XmNmwmDecorations** resource is not to specify any decoration flags for the **_MOTIF_WM_HINTS** property.

The value of this resource is the bitwise inclusive OR of one or more flag bits. The possible flag bit constants, defined in the include file **Xm/MwmUtil.h**, are

MWM_DECOR_ALL
: All decorations *except* those specified by other flag bits that are set

MWM_DECOR_BORDER
: Client window border

MWM_DECOR_RESIZEH
: Resize frame handles

MWM_DECOR_TITLE
: Title bar

MWM_DECOR_MENU
: Window menu button

MWM_DECOR_MINIMIZE
: Minimize window button

MWM_DECOR_MAXIMIZE
: Maximize window button

XmNmwmFunctions

Specifies the function flags (specific window manager functions to apply or not apply to the client window) for the

_MOTIF_WM_HINTS property. If any function flags are specified by the **_MOTIF_WM_HINTS** property, only functions indicated by both that property and the MWM **clientFunctions** and **transientFunctions** resources are applied. If no function flags are specified by the **_MOTIF_WM_HINTS** property, functions indicated by the MWM **clientFunctions** and **transientFunctions** resources are applied. The default for the **XmNmwmFunctions** resource is not to specify any function flags for the **_MOTIF_WM_HINTS** property.

The value of this resource is the bitwise inclusive OR of one or more flag bits. The possible flag bit constants, defined in the include file **Xm/MwmUtil.h**, are

MWM_FUNC_ALL	All functions *except* those specified by other flag bits that are set
MWM_FUNC_RESIZE	**f.resize**
MWM_FUNC_MOVE	**f.move**
MWM_FUNC_MINIMIZE	**f.minimize**
MWM_FUNC_MAXIMIZE	**f.maximize**
MWM_FUNC_CLOSE	**f.kill**

XmNmwmInputMode

Specifies the input mode flag (application modal or system modal input constraints) for the **_MOTIF_WM_HINTS** property. If no input mode flag is specified by the **_MOTIF_WM_HINTS** property, no input constraints are applied, and input goes to any window. The default for the **XmNmwmInputMode** resource is not to specify any input mode flag for the **_MOTIF_WM_HINTS** property.

An application that sets input constraints on a dialog usually uses the BulletinBoard's **XmNdialogStyle** resource rather than the parent DialogShell's **XmNmwmInputMode** resource.

The possible values for this resource, defined in the include file **Xm/MwmUtil.h**, are

MWM_INPUT_MODELESS
Input goes to any window.

MWM_INPUT_PRIMARY_APPLICATION_MODAL
Input does not go to ancestors of this window.

MWM_INPUT_SYSTEM_MODAL
Input goes only to this window.

MWM_INPUT_FULL_APPLICATION_MODAL
Input does not go to other windows in this application.

XmNmwmMenu
Specifies the menu items that the Motif window manager should add to the end of the window menu. The string contains a list of items separated by **\n** with the following format:

label [mnemonic] [accelerator] function

If more than one item is specified, the items should be separated by a newline character.

XmNpreeditType
Specifies the input method style or styles available to the input manager. The syntax, possible values, and default value are implementation dependent.

XmNshellUnitType
Determines geometric resource interpretation. The following values are allowed:

XmPIXELS All values provided to the widget are treated as normal pixel values.

Xm100TH_MILLIMETERS
All values provided to the widget are treated as 1/100 of a millimeter.

Xm1000TH_INCHES
All values provided to the widget are treated as 1/1000 of an inch.

Xm100TH_POINTS
All values provided to the widget are treated as 1/100 of a point. A point is a unit used in text processing applications and is defined as 1/72 inch.

Xm100TH_FONT_UNITS
All values provided to the widget are treated as 1/100 of a font unit. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

XmNtextFontList
Specifies the font list used for VendorShell's Text and List descendants. If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNtextFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the BulletinBoard or VendorShell widget class. If such an ancestor is found, **XmNtextFontList** is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

XmNuseAsyncGeometry
Specifies whether the geometry manager should wait for confirmation of a geometry request to the window manager. When the value of this resource is True, the geometry manager forces **XmNwaitForWm** to False and **XmNwmTimeout** to 0, and it relies on asynchronous notification. When the value of this resource is False, **XmNwaitForWm** and **XmNwmTimeout** are unaffected. The default is False.

Inherited Resources

VendorShell inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

WMShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbaseHeight XmCBaseHeight	XtUnspecifiedShellInt int	CSG
XmNbaseWidth XmCBaseWidth	XtUnspecifiedShellInt int	CSG
XmNheightInc XmCHeightInc	XtUnspecifiedShellInt int	CSG
XmNiconMask XmCIconMask	NULL Pixmap	CSG
XmNiconPixmap XmCIconPixmap	NULL Pixmap	CSG
XmNiconWindow XmCIconWindow	NULL Window	CSG
XmNiconX XmCIconX	-1 int	CSG
XmNiconY XmCIconY	-1 int	CSG
XmNinitialState XmCInitialState	NormalState int	CSG
XmNinput XmCInput	True Boolean	CSG
XmNmaxAspectX XmCMaxAspectX	XtUnspecifiedShellInt int	CSG
XmNmaxAspectY XmCMaxAspectY	XtUnspecifiedShellInt int	CSG
XmNmaxHeight XmCMaxHeight	XtUnspecifiedShellInt int	CSG
XmNmaxWidth XmCMaxWidth	XtUnspecifiedShellInt int	CSG
XmNminAspectX XmCMinAspectX	XtUnspecifiedShellInt int	CSG

Name Class	Default Type	Access
XmNminAspectY XmCMinAspectY	XtUnspecifiedShellInt int	CSG
XmNminHeight XmCMinHeight	XtUnspecifiedShellInt int	CSG
XmNminWidth XmCMinWidth	XtUnspecifiedShellInt int	CSG
XmNtitle XmCTitle	dynamic String	CSG
XmNtitleEncoding XmCTitleEncoding	dynamic Atom	CSG
XmNtransient XmCTransient	False Boolean	CSG
XmNwaitForWm XmCWaitForWm	True Boolean	CSG
XmNwidthInc XmCWidthInc	XtUnspecifiedShellInt int	CSG
XmNwindowGroup XmCWindowGroup	dynamic Window	CSG
XmNwinGravity XmCWinGravity	dynamic int	CSG
XmNwmTimeout XmCWmTimeout	5000 ms int	CSG

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	False Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	False Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for VendorShell.

Related Information

Composite(3X), **Core(3X)**, **mwm(1X)**, **Shell(3X)**, **WMShell(3X)**, **XmActivateProtocol(3X)**, **XmActivateWMProtocol(3X)**, **XmAddProtocolCallback(3X)**, **XmAddWMProtocolCallback(3X)**, **XmAddProtocols(3X)**, **XmAddWMProtocols(3X)**, **XmDeactivateProtocol(3X)**, **XmDeactivateWMProtocol(3X)**, **XmGetAtomName(3X)**, **XmInternAtom(3X)**, **XmIsMotifWMRunning(3X)**, **XmRemoveProtocolCallback(3X)**, **XmRemoveWMProtocolCallback(3X)**, **XmRemoveProtocols(3X)**, **XmRemoveWMProtocols(3X)**, **XmScreen(3X)**, **XmSetProtocolHooks(3X)**, and **XmSetWMProtocolHooks(3X)**.

VirtualBindings—Bindings for virtual mouse and key events

Description

The OSF/Motif reference pages describe translations in terms of *virtual bindings*, based on those described in the *OSF/Motif Style Guide*. Mouse events are described in terms of *virtual buttons*, and key events are described in terms of *virtual keys*. The term *virtual* implies that the events as described do not necessarily correspond to a fixed set of X Window System events. Instead, virtual buttons and keys are linked to actual events by means of virtual bindings.

Virtual Modifiers

Both virtual buttons and virtual keys may contain *virtual modifiers*. Each virtual modifier corresponds to one or more actual modifiers. The following table lists the bindings of virtual modifiers to actual modifiers in OSF/Motif.

Virtual Modifier Bindings	
Virtual Modifier	**Actual Modifiers**
MAlt	**<Mod1>**
MCopy	**<Ctrl>**
MCtrl	**<Ctrl>**
MLink	**<Ctrl> <Shift>**
MMove	**<Shift>**
MShift	**<Shift>**

Mod1 refers to the first modifier key. OSF/Motif requires that it correspond to either **<Alt>** or **<Meta>**.

The virtual modifier **MAny** indicates that any modifier can be used. If **MAny** is not specified and the user presses an actual modifier that is not explicitly included in a translation, that modifier may prevent the translation from being matched.

Virtual Buttons

Each virtual button corresponds to one or more actual button event descriptions. Each button event description contains a button name and possibly modifiers. These button event descriptions, appropriately ordered and possibly further modified, are used in translation tables. The following table lists the bindings of virtual buttons to actual button event descriptions in OSF/Motif.

Virtual Button Bindings	
Virtual Button	**Actual Button Events**
BCustom	**Btn3**
BTransfer	**Btn2**
BExtend	**<Shift>Btn1**
BMenu	**Btn3**
BSelect	**Btn1**
BToggle	**<Ctrl>Btn1**

Virtual Keys

Each virtual key corresponds to one or more actual key event descriptions. Each key event description contains a keysym name and possibly modifiers. These key event descriptions, appropriately ordered and possibly further modified, are used in translation tables. The following table lists the bindings of virtual keys to actual key event descriptions in OSF/Motif.

Virtual Key Bindings	
Virtual Key	**Actual Key Events**
KActivate	<Key><Return> <Ctrl><Key><Return> <Key>osfActivate
KAddMode	<Key>osfAddMode
KBackSpace	<Key>osfBackSpace
KBackTab	<Shift><Key><Tab>
KBeginData	<Ctrl><Key>osfBeginLine
KBeginLine	<Key>osfBeginLine
KCancel	<Key>osfCancel
KClear	<Key>osfClear
KCopy	<Key>osfCopy <Ctrl><Key>osfInsert
KCut	<Key>osfCut <Shift><Key>osfDelete
KDelete	<Key>osfDelete
KDeselectAll	<Ctrl><Key>backslash
KDown	<Key>osfDown
KEndData	<Ctrl><Key>osfEndLine
KEndLine	<Key>osfEndLine
KEnter	<Key><Return>
KEscape	<Key><Escape>
KExtend	<Ctrl> <Shift><Key>space <Shift><Key>osfSelect
KHelp	<Key>osfHelp
KInsert	<Key>osfInsert
KLeft	<Key>osfLeft

Virtual Key Bindings	
Virtual Key	**Actual Key Events**
KMenu	<Key>osfMenu
KMenuBar	<Key>osfMenuBar
KNextField	<Key><Tab> <Ctrl><Key><Tab>
KNextMenu	<Ctrl><Key>osfDown <Ctrl><Key>osfRight
KPageDown	<Key>osfPageDown
KPageLeft	<Ctrl><Key>osfPageUp <Key>osfPageLeft
KPageRight	<Ctrl><Key>osfPageDown <Key>osfPageRight
KPageUp	<Key>osfPageUp
KPaste	<Key>osfPaste <Shift><Key>osfInsert
KPrevField	<Shift><Key><Tab> <Ctrl> <Shift><Key><Tab>
KPrevMenu	<Ctrl><Key>osfUp <Ctrl><Key>osfLeft
KPrimaryCopy	<Ctrl><Key>osfPrimaryPaste Mod1<Key>osfCopy Mod1 <Ctrl><Key>osfInsert
KPrimaryCut	Mod1<Key>osfPrimaryPaste Mod1<Key>osfCut Mod1 <Shift><Key>osfDelete
KPrimaryPaste	<Key>osfPrimaryPaste

Virtual Key Bindings	
Virtual Key	**Actual Key Events**
KQuickCopy	**<Ctrl><Key>osfQuickPaste**
KQuickCut	**Mod1<Key>osfQuickPaste**
KQuickExtend	**<Shift><Key>osfQuickPaste**
KQuickPaste	**<Key>osfQuickPaste**
KReselect	**<Ctrl> <Shift><Key>osfSelect**
KRestore	**<Ctrl> <Shift><Key>osfInsert**
KRight	**<Key>osfRight**
KSelect	**<Key>space** **<Ctrl><Key>space** **<Key>osfSelect**
KSelectAll	**<Ctrl><Key>slash**
KSpace	**<Key>space**
KTab	**<Key><Tab>**
KUndo	**<Key>osfUndo** **Mod1<Key>osfBackSpace**
KUp	**<Key>osfUp**
KAny	**<Key>**

Bindings for osf Keysyms

Keysym strings that begin with **osf** are not part of the X server's keyboard mapping. Instead, these keysyms are produced on the client side at run time. They are interpreted by the routine **XmTranslateKey**, and are used by the translation manager when the server delivers an actual key event. For each application, a mapping is maintained between **osf** keysyms and keysyms that correspond to actual keys. This mapping is based on information obtained at application startup from one of the following sources, listed in order of precedence:

- A **defaultVirtualBindings** application resource in the resource database.
- A property on the root window, which can be set by **mwm** on startup, or by the **xmbind** client, or on prior startup of a Motif application.

- The file **.motifbind** in the user's home directory.
- A set of bindings based on the vendor string and optionally the vendor release of the X server. Motif searches for these bindings in the following steps:

  1. If the file **xmbind.alias** exists in the user's home directory, Motif searches this file for a pathname associated with the vendor string or with the vendor string and vendor release. If it finds such a pathname and if that file exists, Motif loads the bindings contained in that file.
  2. If it has found no bindings, Motif next looks for the file **xmbind.alias** in the directory specified by the environment variable **XMBINDDIR**, if **XMBINDDIR** is set, or in the directory **/usr/lib/Xm/bindings** if **XMBINDDIR** is not set. If this file exists Motif searches it for a pathname associated with the vendor string or with the vendor string and vendor release. If it finds such a pathname and if that file exists, Motif loads the bindings contained in that file.
  3. If it still has found no bindings, Motif loads a set of hard-coded fallback bindings.

The **xmbind.alias** file contains zero or more lines of the following form:

"vendor_string[vendor_release]"bindings_file

where *vendor_string* is the X server vendor name as returned by the X client **xdpyinfo** or the Xlib function **XServerVendor**, and must appear in double quotes. If *vendor_release* is included, it is the X server vendor release number as returned by the X client **xdpyinfo** or the Xlib function **XVendorRelease**, and must also be contained within the double quotes separated by one space from *vendor_string*. The *vendor_release* argument is provided to allow support of changes in keyboard hardware from a vendor, assuming that the vendor increments the release number to flag such changes. Alternatively, the vendor may simply use a unique vendor string for each different keyboard.

The *bindings_file* argument is the pathname of the file containing the bindings themselves. It can be a relative or absolute pathname. If it it is a relative pathname, it is relative to the location of the **xmbind.alias** file.

Comment lines in the **xmbind.alias** file begin with ! (exclamation point).

The bindings found in either the **.motifbind** file or the vendor mapping are placed in a property on the root window. This property is used to determine the bindings for subsequent Motif applications.

On startup **mwm** attempts to load the file **.motifbind** in the user's home directory. If this is unsuccessful, it loads the vendor bindings as described previously. It places the bindings it loads in a property on the root window for use by subsequent Motif applications.

The **xmbind** function loads bindings from a file if that file is specified on the command line. If no file is specified on the command line, it attempts to load the file **.motifbind** in the user's home directory. If this fails, it loads the vendor bindings as described previously. It places the bindings it loads in a property on the root window for use by subsequent Motif applications.

The format of the specification for mapping **osf** keysyms to actual keysyms is similar to that of a specification for an event translation. The syntax is specified here in EBNF notation using the following conventions:

[*a*]	Means either nothing or *a*
{*a*}	Means zero or more occurrences of *a*

Terminals are enclosed in double quotation marks.

The syntax of an **osf** keysym binding specification is as follows:

binding_spec	=	{line "\n"} [line]
line	=	virtual_keysym ":" key_event
key_event	=	{modifier_name} "<Key>" actual_keysym
virtual_keysym	=	keysym
actual_keysym	=	keysym
keysym	=	A valid X11 keysym name that is mapped by **XStringToKeysym**

As with event translations, more specific event descriptions must precede less specific descriptions. For example, an event description for a key with a modifier must precede a description for the same key without the same modifier.

Following is an example of a specification for the **defaultVirtualBindings** resource in a resource file:

```
*defaultVirtualBindings: \
      osfBackSpace      :     <Key>BackSpace \n\
      osfInsert         :     <Key>InsertChar\n\
...
      osfDelete         :     <Key>DeleteChar
```

The format of a **.motifbind** file or of a file containing vendor bindings is the same, except that the binding specification for each keysym is placed on a separate line. The previous example specification appears as follows in a **.motifbind** or vendor bindings file:

```
osfBackSpace      :     <Key>BackSpace
osfInsert         :     <Key>InsertChar
...
osfDelete         :     <Key>DeleteChar
```

The following table lists the fixed fallback default bindings for **osf** keysyms.

Fallback Default Bindings for osf Keysyms	
osf Keysym	**Fallback Default Binding**
osfActivate	<unbound>
osfAddMode	<Shift> F8
osfBackSpace	Backspace
osfBeginLine	Home
osfClear	Clear
osfCopy	unbound
osfCut	unbound
osfDelete	Delete
osfDown	Down
osfEndLine	End
osfCancel	<Escape>
osfHelp	F1
osfInsert	Insert
osfLeft	Left
osfMenu	F4
osfMenuBar	F10
osfPageDown	Next
osfPageLeft	unbound
osfPageRight	unbound
osfPageUp	Prior
osfPaste	unbound
osfPrimaryPaste	unbound
osfQuickPaste	unbound
osfRight	Right
osfSelect	Select
osfUndo	Undo
osfUp	Up

Related Information

xmbind(1X)

WMShell—The WMShell widget class

Synopsis

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
```

Description

WMShell is a top-level widget that encapsulates the interaction with the window manager.

Classes

WMShell inherits behavior and resources from the **Core**, **Composite**, and **Shell** classes.

The class pointer is **wmShellWidgetClass**.

The class name is **WMShell**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

WMShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbaseHeight XmCBaseHeight	XtUnspecifiedShellInt int	CSG
XmNbaseWidth XmCBaseWidth	XtUnspecifiedShellInt int	CSG
XmNheightInc XmCHeightInc	XtUnspecifiedShellInt int	CSG
XmNiconMask XmCIconMask	NULL Pixmap	CSG
XmNiconPixmap XmCIconPixmap	NULL Pixmap	CSG
XmNiconWindow XmCIconWindow	NULL Window	CSG
XmNiconX XmCIconX	-1 int	CSG
XmNiconY XmCIconY	-1 int	CSG
XmNinitialState XmCInitialState	NormalState int	CSG
XmNinput XmCInput	False Boolean	CSG
XmNmaxAspectX XmCMaxAspectX	XtUnspecifiedShellInt int	CSG
XmNmaxAspectY XmCMaxAspectY	XtUnspecifiedShellInt int	CSG
XmNmaxHeight XmCMaxHeight	XtUnspecifiedShellInt int	CSG
XmNmaxWidth XmCMaxWidth	XtUnspecifiedShellInt int	CSG
XmNminAspectX XmCMinAspectX	XtUnspecifiedShellInt int	CSG

Name Class	Default Type	Access
XmNminAspectY XmCMinAspectY	XtUnspecifiedShellInt int	CSG
XmNminHeight XmCMinHeight	XtUnspecifiedShellInt int	CSG
XmNminWidth XmCMinWidth	XtUnspecifiedShellInt int	CSG
XmNtitle XmCTitle	dynamic String	CSG
XmNtitleEncoding XmCTitleEncoding	dynamic Atom	CSG
XmNtransient XmCTransient	False Boolean	CSG
XmNwaitForWm XmCWaitForWm	True Boolean	CSG
XmNwidthInc XmCWidthInc	XtUnspecifiedShellInt int	CSG
XmNwindowGroup XmCWindowGroup	dynamic Window	CSG
XmNwinGravity XmCWinGravity	dynamic int	CSG
XmNwmTimeout XmCWmTimeout	5000 ms int	CSG

XmNbaseHeight

Specifies the base for a progression of preferred heights for the window manager to use in sizing the widget. The preferred heights are **XmNbaseHeight** plus integral multiples of **XmNheightInc**, with a minimum of **XmNminHeight** and a maximum of **XmNmaxHeight**. If an initial value is not supplied for **XmNbaseHeight** but is supplied for **XmNbaseWidth**, the value of **XmNbaseHeight** is set to 0 (zero) when the widget is realized.

XmNbaseWidth

Specifies the base for a progression of preferred widths for the window manager to use in sizing the widget. The preferred widths are **XmNbaseWidth** plus integral multiples of **XmNwidthInc**, with a minimum of **XmNminWidth** and a maximum of **XmNmaxWidth**. If an initial value is not supplied for **XmNbaseWidth** but is supplied

for **XmNbaseHeight**, the value of **XmNbaseWidth** is set to 0 (zero) when the widget is realized.

XmNheightInc
Specifies the increment for a progression of preferred heights for the window manager to use in sizing the widget. The preferred heights are **XmNbaseHeight** plus integral multiples of **XmNheightInc**, with a minimum of **XmNminHeight** and a maximum of **XmNmaxHeight**. If an initial value is not supplied for **XmNheightInc** but is supplied for **XmNwidthInc**, the value of **XmNheightInc** is set to 1 when the widget is realized.

XmNiconMask
Specifies a bitmap that could be used by the window manager to clip the **XmNiconPixmap** bitmap to make the icon nonrectangular.

XmNiconPixmap
Specifies a bitmap that could be used by the window manager as the application's icon.

XmNiconWindow
Specifies the ID of a window that could be used by the window manager as the application's icon.

XmNiconX Specifies a suitable place to put the application's icon; this is a hint to the window manager in root window coordinates. Because the window manager controls icon placement policy, this resource may be ignored. If no initial value is specified, the value is set to -1 when the widget is realized.

XmNiconY Specifies a suitable place to put the application's icon; this is a hint to the window manager in root window coordinates. Because the window manager controls icon placement policy, this resource may be ignored. If no initial value is specified, the value is set to -1 when the widget is realized.

XmNinitialState
Specifies the state the application wants the widget instance to start in. It must be one of the constants **NormalState** or **IconicState**.

XmNinput Specifies the application's input model for this widget and its descendants. The meaning of a True or False value for this resource depends on the presence or absence of a WM_TAKE_FOCUS atom in the WM_PROTOCOLS property:

Input Model	XmNinput	WM_TAKE_FOCUS
No input	False	Absent
Passive	True	Absent
Locally active	True	Present
Globally active	False	Present

For more information on input models, see the X Consortium Standard *Inter-Client Communication Conventions Manual* (ICCM).

XmNmaxAspectX
Specifies the numerator of the maximum aspect ratio (X/Y) that the application wants the widget instance to have.

XmNmaxAspectY
Specifies the denominator of the maximum aspect ratio (X/Y) that the application wants the widget instance to have.

XmNmaxHeight
Specifies the maximum height that the application wants the widget instance to have. If an initial value is not supplied for **XmNmaxHeight** but is supplied for **XmNmaxWidth**, the value of **XmNmaxHeight** is set to 32767 when the widget is realized.

XmNmaxWidth
Specifies the maximum width that the application wants the widget instance to have. If an initial value is not supplied for **XmNmaxWidth** but is supplied for **XmNmaxHeight**, the value of **XmNmaxWidth** is set to 32767 when the widget is realized.

XmNminAspectX
Specifies the numerator of the minimum aspect ratio (X/Y) that the application wants the widget instance to have.

XmNminAspectY
Specifies the denominator of the minimum aspect ratio (X/Y) that the application wants the widget instance to have.

XmNminHeight

Specifies the minimum height that the application wants the widget instance to have. If an initial value is not supplied for **XmNminHeight** but is supplied for **XmNminWidth**, the value of **XmNminHeight** is set to 1 when the widget is realized.

XmNminWidth

Specifies the minimum width that the application wants the widget instance to have. If an initial value is not supplied for **XmNminWidth** but is supplied for **XmNminHeight**, the value of **XmNminWidth** is set to 1 when the widget is realized.

XmNtitle Specifies the application name to be displayed by the window manager. The default is the icon name, if specified; otherwise, it is the name of the application.

XmNtitleEncoding

Specifies a property type that represents the encoding of the **XmNtitle** string. If a language procedure has been set, the default is None; otherwise, the default is XA_STRING. When the widget is realized, if the value is None, the corresponding name is assumed to be in the current locale. The name is passed to **XmbTextListToTextProperty** with an encoding style of **XStdICCTextStyle**. The resulting encoding is STRING if the name is fully convertible to STRING, otherwise COMPOUND_TEXT. The values of the encoding resources are not changed; they remain None.

XmNtransient

Specifies a Boolean value that is True if the widget instance is transient, typically a popup on behalf of another widget. The window manager may treat a transient widget's window differently from other windows. For example, a window manager may not iconify a transient window separately from its associated application. Applications and users should not normally alter this resource.

XmNwaitForWm

When True, specifies that the Intrinsics waits the length of time given by the **XmNwmTimeout** resource for the window manager to respond to certain actions before assuming that there is no window manager present. This resource is altered by the Intrinsics as it receives, or fails to receive, responses from the window manager.

XmNwidthInc

Specifies the base for a progression of preferred widths for the window manager to use in sizing the widget. The preferred widths are **XmNbaseWidth** plus integral multiples of **XmNwidthInc**, with a minimum of **XmNminWidth** and a maximum of **XmNmaxWidth**. If an initial value is not supplied for **XmNwidthInc** but is supplied for **XmNheightInc**, the value of **XmNwidthInc** is set to 1 when the widget is realized.

XmNwindowGroup

Specifies the ID of a window with which this widget instance is associated. By convention, this window is the "leader" of a group of windows. A window manager may treat all windows in a group in some way; for example, it may always move or iconify them together.

If no initial value is specified, the value is set to the window of the first realized ancestor widget in the parent hierarchy when the widget is realized. If a value of **XtUnspecifiedWindowGroup** is specified, no window group is set.

XmNwinGravity

Specifies the window gravity for use by the window manager in positioning the widget. If no initial value is specified, the value is set when the widget is realized. If **XmNgeometry** is not NULL, **XmNwinGravity** is set to the window gravity returned by **XWMGeometry**. Otherwise, **XmNwinGravity** is set to **NorthWestGravity**.

XmNwmTimeout

Specifies the length of time that the Intrinsics waits for the window manager to respond to certain actions before assuming that there is no window manager present. The value is in milliseconds and must not be negative.

Inherited Resources

WMShell inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	False Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	False Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for WMShell.

Related Information

Composite(3X), **Core(3X)**, and **Shell(3X)**.

XmActivateProtocol—A VendorShell function that activates a protocol

Synopsis

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

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

void XmActivateWMProtocol (shell, protocol)
        Widget      shell;
        Atom        protocol;
```

Description

XmActivateProtocol activates a protocol. It updates the handlers and the *property* if the *shell* is realized. It is sometimes useful to allow a protocol's state information (callback lists, and so on) to persist, even though the client may choose to temporarily resign from the interaction. This is supported by allowing a *protocol* to be in one of two states: active or inactive. If the *protocol* is active and the *shell* is realized, the *property* contains the *protocol* **Atom**. If the *protocol* is inactive, the **Atom** is not present in the *property*.

XmActivateWMProtocol is a convenience interface. It calls **XmActivateProtocol** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

property Specifies the protocol property

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**)

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmActivateWMProtocol(3X)** and **XmInternAtom(3X)**.

XmActivateWMProtocol—A VendorShell convenience interface that activates a protocol

Synopsis

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

void XmActivateWMProtocol (shell, protocol)
        Widget      shell;
        Atom        protocol;
```

Description

XmActivateWMProtocol is a convenience interface. It calls **XmActivateProtocol** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**)

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmActivateProtocol(3X)**, and **XmInternAtom(3X)**.

XmAddProtocolCallback—A VendorShell function that adds client callbacks for a protocol

Synopsis

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

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

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

Description

XmAddProtocolCallback adds client callbacks for a protocol. It checks if the protocol is registered, and if it is not, calls **XmAddProtocols**. It then adds the callback to the internal list. These callbacks are called when the corresponding client message is received.

XmAddWMProtocolCallback is a convenience interface. It calls **XmAddProtocolCallback** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

property Specifies the protocol property

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**)

callback Specifies the procedure to call when a protocol message is received

closure Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmAddProtocols(3X)**, **XmAddWMProtocolCallback(3X)**, and **XmInternAtom(3X)**.

XmAddProtocols—A VendorShell function that adds the protocols to the protocol manager and allocates the internal tables

Synopsis

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

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

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

Description

XmAddProtocols adds the protocols to the protocol manager and allocates the internal tables.

XmAddWMProtocols is a convenience interface. It calls **XmAddProtocols** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

property Specifies the protocol property

protocols Specifies the protocol **Atoms** (or **int** types cast to **Atom**)

num_protocols
Specifies the number of elements in *protocols*

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmAddWMProtocols(3X)**, and **XmInternAtom(3X)**.

XmAddTabGroup—A function that adds a manager or a primitive widget to the list of tab groups

Synopsis

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

void XmAddTabGroup (tab_group)
        Widget          tab_group;
```

Description

This function is obsolete and its behavior is replaced by setting **XmNnavigationType** to **XmEXCLUSIVE_TAB_GROUP**. When using the keyboard to traverse through a widget hierarchy, primitive or manager widgets are grouped together into what are known as **tab groups**. Any manager or primitive widget can be a tab group. Within a tab group, move the focus to the next widget within the tab group by using the arrow keys. To move to another tab group, use **KNextField** or **KPrevField**.

Tab groups are ordinarily specified by the **XmNnavigationType** resource. **XmAddTabGroup** is called to control the order of traversal of tab groups. The widget specified by *tab_group* is appended to the list of tab groups to be traversed, and the widget's **XmNnavigationType** is set to **XmEXCLUSIVE_TAB_GROUP**.

tab_group Specifies the manager or primitive widget ID

Related Information

XmManager(3X), **XmGetTabGroup(3X)**, **XmPrimitive(3X)**, and **XmRemoveTabGroup(3X)**.

XmAddWMProtocolCallback—A VendorShell convenience interface that adds client callbacks for a protocol

Synopsis

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

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

Description

XmAddWMProtocolCallback is a convenience interface. It calls **XmAddProtocolCallback** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**)

callback Specifies the procedure to call when a protocol message is received

closure Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmAddProtocolCallback(3X)**, and **XmInternAtom(3X)**.

XmAddWMProtocols—A VendorShell convenience interface that adds the protocols to the protocol manager and allocates the internal tables

Synopsis

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

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

Description

XmAddWMProtocols is a convenience interface. It calls **XmAddProtocols** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

protocols Specifies the protocol **Atoms** (or **int** types cast to **Atom**)

num_protocols
Specifies the number of elements in *protocols*

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmAddProtocols(3X)**, and **XmInternAtom(3X)**.

XmArrowButton—The ArrowButton widget class

Synopsis

#include <Xm/ArrowB.h>

Description

ArrowButton consists of a directional arrow surrounded by a border shadow. When it is selected, the shadow changes to give the appearance that the ArrowButton has been pressed in. When the ArrowButton is unselected, the shadow reverts to give the appearance that the ArrowButton is released, or out.

Classes

ArrowButton inherits behavior and resources from the **Core** and **XmPrimitive** classes.

The class pointer is **xmArrowButtonWidgetClass**.

The class name is **XmArrowButton**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmArrowButton Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNarmCallback XmCCallback	NULL XtCallbackList	C
XmNarrowDirection XmCArrowDirection	XmARROW_UP unsigned char	CSG
XmNdisarmCallback XmCCallback	NULL XtCallbackList	C
XmNmultiClick XmCMultiClick	dynamic unsigned char	CSG

XmNactivateCallback

Specifies a list of callbacks that is called when the ArrowButton is activated. To activate the button, press and release **BSelect** while the pointer is inside the ArrowButton widget. Activating the ArrowButton also disarms it. The reason sent by this callback is **XmCR_ACTIVATE**.

XmNarmCallback

Specifies a list of callbacks that is called when the ArrowButton is armed. To arm this widget, press **BSelect** while the pointer is inside the ArrowButton. The reason sent by this callback is **XmCR_ARM**.

XmNarrowDirection

Sets the arrow direction. The values for this resource are

- **XmARROW_UP**
- **XmARROW_DOWN**
- **XmARROW_LEFT**
- **XmARROW_RIGHT**

XmNdisarmCallback

Specifies a list of callbacks that is called when the ArrowButton is disarmed. To disarm this widget, press and release **BSelect** while the pointer is inside the ArrowButton. The reason for this callback is **XmCR_DISARM**.

XmNmultiClick

If a button click is followed by another button click within the time span specified by the display's multiclick time, and this resource is set to **XmMULTICLICK_DISCARD**, the second click. is not processed. If this resource is set to **XmMULTICLICK_KEEP**, the event is processed and *click_count* is incremented in the callback structure. When the button is not in a menu, the default value is **XmMULTICLICK_KEEP**.

Inherited Resources

ArrowButton inherits behavior and resources from the superclasses described in the following table. For a complete description of each resource, refer to the reference page for that superclass.

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    int         click_count;
} XmArrowButtonCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

click_count This value is valid only when the reason is **XmCR_ACTIVATE**. It contains the number of clicks in the last multiclick sequence if the **XmNmultiClick** resource is set to **XmMULTICLICK_KEEP**; otherwise it contains 1. The activate callback is invoked for each click if **XmNmultiClick** is set to **XmMULTICLICK_KEEP**.

Translations

XmArrowButton includes translations for XmPrimitive. The following list describes additional XmArrowButton translations. These translations may not directly correspond to a translation table.

BSelect Press:	**Arm()**
BSelect Click:	**Activate()**
	Disarm()
BSelect Release:	**Activate()**
	Disarm()
BSelect Press 2+:	**MultiArm()**
BSelect Release 2+:	**MultiActivate()**
KSelect:	**ArmAndActivate()**
KHelp:	**Help()**

Action Routines

The XmArrowButton action routines are

Activate(): Draws the shadow in the unselected state. If the pointer is within the ArrowButton, calls the callbacks for **XmNactivateCallback**.

Arm(): Draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**.

ArmAndActivate():
Draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**. Arranges for the shadow to be drawn in the unselected state and the callbacks for **XmNactivateCallback** and **XmNdisarmCallback** to be called, either immediately or at a later time.

Disarm(): Draws the shadow in the unselected state and calls the callbacks for **XmNdisarmCallback**.

Help(): Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MultiActivate():

If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action increments *click_count* in the callback structure and draws the shadow in the unselected state. If the pointer is within the ArrowButton, this action calls the callbacks for **XmNactivateCallback** and **XmNdisarmCallback**.

MultiArm(): If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing. If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**.

Additional Behavior

This widget has the following additional behavior:

<EnterWindow>:

Draws the ArrowButton shadow in its selected state if the pointer leaves and re-enters the window while **BSelect** is pressed.

<LeaveWindow>:

Draws the ArrowButton shadow in its unselected state if the pointer leaves the window while **BSelect** is pressed.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateArrowButton(3X)**, and **XmPrimitive(3X)**.

XmArrowButtonGadget—The ArrowButtonGadget widget class

Synopsis

#include <Xm/ArrowBG.h>

Description

ArrowButtonGadget consists of a directional arrow surrounded by a border shadow. When it is selected, the shadow changes to give the appearance that the ArrowButtonGadget has been pressed in. When it is unselected, the shadow reverts to give the appearance that the button is released, or out.

Classes

ArrowButtonGadget inherits behavior and resources from the **Object**, **RectObj**, and **XmGadget** classes.

The class pointer is **xmArrowButtonGadgetClass**.

The class name is **XmArrowButtonGadget**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

ArrowButtonGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNarmCallback XmCCallback	NULL XtCallbackList	C
XmNarrowDirection XmCArrowDirection	XmARROW_UP unsigned char	CSG
XmNdisarmCallback XmCCallback	NULL XtCallbackList	C
XmNmultiClick XmCMultiClick	dynamic unsigned char	CSG

XmNactivateCallback

Specifies a list of callbacks that is called when the ArrowButtonGadget is activated. To activate the button, press and release **BSelect** while the pointer is inside the ArrowButtonGadget. Activating the ArrowButtonGadget also disarms it. The reason sent by this callback is **XmCR_ACTIVATE**.

XmNarmCallback

Specifies a list of callbacks that is called when the ArrowButtonGadget is armed. To arm this widget, press **BSelect** while the pointer is inside the ArrowButtonGadget. The reason sent by this callback is **XmCR_ARM**.

XmNarrowDirection

Sets the arrow direction. The values for this resource are

- **XmARROW_UP**
- **XmARROW_DOWN**
- **XmARROW_LEFT**
- **XmARROW_RIGHT**

XmNdisarmCallback

Specifies a list of callbacks that is called when the ArrowButtonGadget is disarmed. To disarm this widget, press and release **BSelect** while the pointer is inside the ArrowButtonGadget. The reason sent by this callback is **XmCR_DISARM**.

XmNmultiClick

If a button click is followed by another button click within the time span specified by the display's multiclick time and this resource is set to **XmMULTICLICK_DISCARD**, the second click is not processed. If this resource is set to **XmMULTICLICK_KEEP**, the event is processed and *click_count* is incremented in the callback structure. When the ArrowButtonGadget is not in a menu, the default value is **XmMULTICLICK_KEEP**.

Inherited Resources

ArrowButtonGadget inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	G
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	G
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	G
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

RectObj Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	0 Dimension	N/A
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    int         click_count;
} XmArrowButtonCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

click_count This value is valid only when the reason is **XmCR_ACTIVATE**. It contains the number of clicks in the last multiclick sequence if the **XmNmultiClick** resource is set to **XmMULTICLICK_KEEP**, otherwise it contains **1**. The activate callback is invoked for each click if **XmNmultiClick** is set to **XmMULTICLICK_KEEP**.

Behavior

XmArrowButtonGadget includes behavior from **XmGadget**. The following list describes additional XmArrowButtonGadget behavior:

BSelect Press:
Draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**.

BSelect Click or BSelectRelease:
Draws the shadow in the unselected state. If the pointer is within the ArrowButtonGadget, calls the callbacks for **XmNactivateCallback**. Calls the callbacks for **XmNdisarmCallback**.

BSelect Press 2+:
If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing. If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**.

BSelect Release 2+:
If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action increments *click_count* in the callback structure and draws the shadow in the unselected state. If the pointer is within the ArrowButtonGadget, this action calls the callbacks for **XmNactivateCallback** and **XmNdisarmCallback**.

KSelect: Draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**. Arranges for the shadow to be drawn in the unselected state and the callbacks for **XmNactivateCallback** and **XmNdisarmCallback** to be called, either immediately or at a later time.

KHelp: Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

<Enter>: Draws the ArrowButtonGadget shadow in its selected state if the pointer leaves and re-enters the gadget while BSelect is pressed.

<Leave>: Draws the ArrowButtonGadget shadow in its unselected state if the pointer leaves the gadget while BSelect is pressed.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Object(3X), **RectObj(3X)**, **XmCreateArrowButtonGadget(3X)**, and **XmGadget(3X)**.

XmBulletinBoard—The BulletinBoard widget class

Synopsis

#include <Xm/BulletinB.h>

Description

BulletinBoard is a composite widget that provides simple geometry management for children widgets. It does not force positioning on its children, but can be set to reject geometry requests that result in overlapping children. BulletinBoard is the base widget for most dialog widgets and is also used as a general container widget.

Modal and modeless dialogs are implemented as collections of widgets that include a DialogShell, a BulletinBoard (or subclass) child of the shell, and various dialog components (buttons, labels, and so on) that are children of BulletinBoard. BulletinBoard defines callbacks useful for dialogs (focus, map, unmap), which are available for application use. If its parent is a DialogShell, BulletinBoard passes title and input mode (based on dialog style) information to the parent, which is responsible for appropriate communication with the window manager.

The default value for **XmNinitialFocus** is the value of **XmNdefaultButton**.

Classes

BulletinBoard inherits behavior and resources from the **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is **xmBulletinBoardWidgetClass**.

The class name is **XmBulletinBoard**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmBulletinBoard Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowOverlap XmCAllowOverlap	True Boolean	CSG
XmNautoUnmanage XmCAutoUnmanage	True Boolean	CG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNcancelButton XmCWidget	NULL Widget	SG
XmNdefaultButton XmCWidget	NULL Widget	SG
XmNdefaultPosition XmCDefaultPosition	True Boolean	CSG
XmNdialogStyle XmCDialogStyle	dynamic unsigned char	CSG
XmNdialogTitle XmCDialogTitle	NULL XmString	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	10 Dimension	CSG
XmNmarginWidth XmCMarginWidth	10 Dimension	CSG
XmNnoResize XmCNoResize	False Boolean	CSG
XmNresizePolicy XmCResizePolicy	XmRESIZE_ANY unsigned char	CSG

Name Class	Default Type	Access
XmNshadowType XmCShadowType	XmSHADOW_OUT unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNtextTranslations XmCTranslations	NULL XtTranslations	C
XmNunmapCallback XmCCallback	NULL XtCallbackList	C

XmNallowOverlap

Controls the policy for overlapping children widgets. If True, BulletinBoard allows geometry requests that result in overlapping children.

XmNautoUnmanage

Controls whether or not BulletinBoard is automatically unmanaged after a button is activated. If this resource is True on initialization and if the BulletinBoard's parent is a DialogShell, BulletinBoard adds a callback to button children (PushButtons, PushButtonGadgets, and DrawnButtons) that unmanages the BulletinBoard when a button is activated. If this resource is False on initialization or if the BulletinBoard's parent is not a DialogShell, the BulletinBoard is not automatically unmanaged. For BulletinBoard subclasses with Apply or Help buttons, activating those buttons does not automatically unmanage the BulletinBoard.

XmNbuttonFontList

Specifies the font list used for BulletinBoard's button descendants. If this value is NULL at initialization, the font list is initialized by looking up the parent hierarchy of the widget for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or XmMenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNbuttonFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on the creation and structure of a font list.

XmNcancelButton

Specifies the widget ID of the **Cancel** button. BulletinBoard's subclasses, which define a **Cancel** button, set this resource. BulletinBoard does not directly provide any behavior for that button.

XmNdefaultButton

Specifies the widget ID of the default button. Some BulletinBoard subclasses, which define a default button, set this resource. BulletinBoard defines translations and installs accelerators that activate that button when **KActivate** is pressed and the keyboard focus is not in another button.

XmNdefaultPosition

Controls whether or not the BulletinBoard is automatically positioned by its parent. If True, and the parent of the BulletinBoard is a DialogShell, the BulletinBoard is centered within or around the parent of the DialogShell when the BulletinBoard is mapped and managed. If False, the BulletinBoard is not automatically positioned.

XmNdialogStyle

Indicates the dialog style associated with the BulletinBoard. If the parent of the BulletinBoard is a DialogShell, the parent's **XmNmwmInputMode** is set according to the value of this resource. This resource can be set only if the BulletinBoard is unmanaged. Possible values for this resource include the following:

XmDIALOG_SYSTEM_MODAL

Used for dialogs that must be responded to before any other interaction in any application.

XmDIALOG_PRIMARY_APPLICATION_MODAL

Used for dialogs that must be responded to before some other interactions in ancestors of the widget.

XmDIALOG_APPLICATION_MODAL

Used for dialogs that must be responded to before some other interactions in ancestors of the widget. This value is the same as **XmDIALOG_PRIMARY_APPLICATION_MODAL**, and remains for compatibility.

XmDIALOG_FULL_APPLICATION_MODAL

Used for dialogs that must be responded to before some other interactions in the same application.

XmDIALOG_MODELESS
Used for dialogs that do not interrupt interaction of any application. This is the default when the parent of the BulletinBoard is a DialogShell.

XmDIALOG_WORK_AREA
Used for BulletinBoard widgets whose parents are not DialogShells. **XmNdialogStyle** is forced to have this value when the parent of the BulletinBoard is not a DialogShell.

XmNdialogTitle
Specifies the dialog title. If this resource is not NULL, and the parent of the BulletinBoard is a subclass of WMShell, BulletinBoard sets the **XmNtitle** and **XmNtitleEncoding** of its parent. If the only character set in **XmNdialogTitle** is ISO8859-1, **XmNtitle** is set to the string of the title, and **XmNtitleEncoding** is set to **STRING**. If **XmNdialogTitle** contains character sets other than ISO8859-1, **XmNtitle** is set to the string of the title converted to a compound text string, and **XmNtitleEncoding** is set to **COMPOUND_TEXT**.

XmNfocusCallback
Specifies the list of callbacks that is called when the BulletinBoard widget or one of its descendants accepts the input focus. The callback reason is **XmCR_FOCUS**.

XmNlabelFontList
Specifies the font list used for BulletinBoard's label descendants (Labels and LabelGadgets). If this value is NULL at initialization, the font list is initialized by looking up the parent hierarchy of the widget for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or XmMenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on the creation and structure of a font list.

XmNmapCallback
Specifies the list of callbacks that is called only when the parent of the BulletinBoard is a DialogShell. In this case, this callback list is invoked when the BulletinBoard widget is mapped. The callback reason is **XmCR_MAP**. DialogShells are usually mapped when the DialogShell is managed.

XmNmarginHeight
Specifies the minimum spacing in pixels between the top or bottom edge of BulletinBoard and any child widget.

XmNmarginWidth
Specifies the minimum spacing in pixels between the left or right edge of BulletinBoard and any child widget.

XmNnoResize
Controls whether or not resize controls are included in the window manager frame around the BulletinBoard's parent. If this resource is set to True, **mwm** does not include resize controls in the window manager frame containing the parent of the BulletinBoard if the parent is a subclass of VendorShell. If this resource is set to False, the window manager frame does include resize controls. Other controls provided by **mwm** can be included or excluded through the **mwm** resources provided by VendorShell.

XmNresizePolicy
Controls the policy for resizing BulletinBoard widgets. Possible values include

XmRESIZE_NONE	Fixed size
XmRESIZE_ANY	Shrink or grow as needed
XmRESIZE_GROW	Grow only

XmNshadowType
Describes the shadow drawing style for BulletinBoard. This resource can have the following values:

XmSHADOW_IN
Draws the BulletinBoard shadow so that it appears inset. This means that the bottom shadow visuals and top shadow visuals are reversed.

XmSHADOW_OUT
Draws the BulletinBoard shadow so that it appears outset.

XmSHADOW_ETCHED_IN
Draws the BulletinBoard shadow using a double line giving the effect of a line etched into the window, similar to the Separator widget.

XmSHADOW_ETCHED_OUT
Draws the BulletinBoard shadow using a double line giving the effect of a line coming out of the window, similar to the Separator widget.

XmNtextFontList
Specifies the font list used for BulletinBoard's Text and List descendants. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard or VendorShell widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on the creation and structure of a font list.

XmNtextTranslations
Adds translations to any Text widget or Text widget subclass that is added as a child of BulletinBoard.

XmNunmapCallback
Specifies the list of callbacks that is called only when the parent of the BulletinBoard is a DialogShell. In this case, this callback list is invoked when the BulletinBoard widget is unmapped. The callback reason is **XmCR_UNMAP**. DialogShells are usually unmapped when the DialogShell is unmanaged.

Inherited Resources

BulletinBoard inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	dynamic Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	N/A
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

Translations

XmBulletinBoard includes the translations from **XmManager**.

Additional Behavior

The **XmBulletinBoard** widget has the following additional behavior:

MAny KCancel:
Calls the activate callbacks for the cancel button if it is sensitive. If no cancel button exists and if the parent of the BulletinBoard is a manager, passes the event to the parent.

KActivate: Calls the activate callbacks for the button with the keyboard focus. If no button has the keyboard focus, calls the activate callbacks for the default button if it is sensitive. In a List widget or single-line Text widget, the List or Text action associated with **KActivate** is called before the BulletinBoard actions associated with **KActivate**.

In a multiline Text widget, any **KActivate** event except **KEnter** calls the Text action associated with **KActivate**, then the BulletinBoard actions associated with **KActivate**. If no button has the focus, no default button exists, and the parent of the BulletinBoard is a manager, passes the event to the parent.

<FocusIn>: Calls the callbacks for **XmNfocusCallback**. When the focus policy is **XmPOINTER**, the callbacks are called when the pointer enters the window. When the focus policy is **XmEXPLICIT**, the callbacks are called when the user traverses to the widget.

<Map>: Calls the callbacks for **XmNmapCallback**.

<Unmap>: Calls the callbacks for **XmNunmapCallback**.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreateBulletinBoard(3X)**, **XmCreateBulletinBoardDialog(3X)**, **XmDialogShell(3X)**, and **XmManager(3X)**.

XmCascadeButton—The CascadeButton widget class

Synopsis

#include <Xm/CascadeB.h>

Description

CascadeButton links two MenuPanes or a MenuBar to a MenuPane.

It is used in menu systems and must have a RowColumn parent with its **XmNrowColumnType** resource set to **XmMENU_BAR**, **XmMENU_POPUP** or **XmMENU_PULLDOWN**.

It is the only widget that can have a Pulldown MenuPane attached to it as a submenu. The submenu is displayed when this widget is activated within a MenuBar, a PopupMenu, or a PulldownMenu. Its visuals can include a label or pixmap and a cascading indicator when it is in a Popup or Pulldown MenuPane; or it can include only a label or a pixmap when it is in a MenuBar.

The default behavior associated with a CascadeButton depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the CascadeButton. In addition, **BMenu** controls the behavior of the CascadeButton if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

A CascadeButton's visuals differ from most other button gadgets. When the button becomes armed, its visuals change from a 2-D to a 3-D look, and it displays the submenu that has been attached to it. If no submenu is attached, it simply changes its visuals.

When a CascadeButton within a Pulldown or Popup MenuPane is armed as the result of the user moving the mouse pointer into the widget, it does not immediately display its submenu. Instead, it waits a short amount of time to see if the arming was temporary (that is, the user was simply passing through the widget), or whether the user really wanted the submenu posted. This time delay is configurable using **XmNmappingDelay**.

CascadeButton provides a single mechanism for activating the widget from the keyboard. This mechanism is referred to as a keyboard mnemonic. If a mnemonic has been specified for the widget, the user may activate the CascadeButton by simply typing the mnemonic while the CascadeButton is visible. If the CascadeButton is in a MenuBar and the MenuBar does not have the focus, the **MAlt** modifier must be pressed with the mnemonic. Mnemonics are typically used to interact with a menu using the keyboard interface.

If in a Pulldown or Popup MenuPane and there is a submenu attached, the **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight**, and **XmNmarginTop** resources may enlarge to accommodate **XmNcascadePixmap**. **XmNmarginWidth** defaults to 6 if this resource is in a MenuBar; otherwise, it takes Label's default, which is 2.

Classes

CascadeButton inherits behavior and resources from **Core**, **XmPrimitive**, and **XmLabel** classes.

The class pointer is **xmCascadeButtonWidgetClass.**

The class name is **XmCascadeButton.**

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmCascadeButton Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNcascadePixmap XmCPixmap	dynamic Pixmap	CSG
XmNcascadingCallback XmCCallback	NULL XtCallbackList	C
XmNmappingDelay XmCMappingDelay	180 ms int	CSG
XmNsubMenuId XmCMenuWidget	NULL Widget	CSG

XmNactivateCallback

Specifies the list of callbacks that is called when the user activates the CascadeButton widget and there is no submenu attached to pop up. The activation occurs when a mouse button is released or when

the mnemonic associated with the widget is typed. The specific mouse button depends on information in the RowColumn parent. The reason sent by the callback is **XmCR_ACTIVATE**.

XmNcascadePixmap
Specifies the cascade pixmap displayed on one end of the widget when a CascadeButton is used within a Popup or Pulldown MenuPane and a submenu is attached. The Label class resources **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight**, and **XmNmarginTop** may be modified to ensure that room is left for the cascade pixmap. The default cascade pixmap is an arrow pointing to the side of the menu where the submenu will appear.

XmNcascadingCallback
Specifies the list of callbacks that is called just prior to the mapping of the submenu associated with CascadeButton. The reason sent by the callback is **XmCR_CASCADING**.

XmNmappingDelay
Specifies the amount of time, in milliseconds, between when a CascadeButton becomes armed and when it maps its submenu. This delay is used only when the widget is within a Popup or Pulldown MenuPane. The value must not be negative.

XmNsubMenuId
Specifies the widget ID for the Pulldown MenuPane to be associated with this CascadeButton. The specified MenuPane is displayed when the CascadeButton becomes armed. The MenuPane must have been created with the appropriate parentage depending on the type of menu used. See **XmCreateMenuBar(3X)**, **XmCreatePulldownMenu(3X)**, and **XmCreatePopupMenu(3X)** for more information on the menu systems.

Inherited Resources

CascadeButton inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmLabel Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	N/A
XmNacceleratorText XmCAcceleratorText	NULL XmString	N/A
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	dynamic Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	0 Dimension	CSG
XmNmarginRight XmCMarginRight	dynamic Dimension	CSG
XmNmarginTop XmCMarginTop	dynamic Dimension	CSG
XmNmarginWidth XmCMarginWidth	dynamic Dimension	CSG

Name Class	Default Type	Access
XmNmnemonic XmCMnemonic	NULL KeySym	CSG

Name Class	Default Type	Access
XmNmnemonicCharSet XmCMnemonicCharSet	XmFONTLIST_DEFAULT_TAG String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	0 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	dynamic Boolean	G
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback or is NULL if this callback was not triggered due to an **XEvent**

Translations

XmCascadeButton includes translations from **XmPrimitive**. **XmCascadeButton** includes the menu traversal translations from **XmLabel**. These translations may not directly correspond to a translation table.

Note that altering translations in **#override** or **#augment** mode is undefined.

The following list describes the translations for a CascadeButton in a MenuBar. These translations may not directly correspond to a translation table.

BSelect Press: **MenuBarSelect()**

BSelect Release: **DoSelect()**

KActivate: **KeySelect()**

KSelect: **KeySelect()**

KHelp: **Help()**

MAny KCancel: **CleanupMenuBar()**

The following list describes the translations for a CascadeButton in a PullDown or Popup MenuPane. In a Popup menu system, **BMenu** also performs the **BSelect** actions. These translations may not directly correspond to a translation table.

BSelect Press: **StartDrag()**

BSelect Release: **DoSelect()**

KActivate: **KeySelect()**

KSelect: **KeySelect()**

KHelp: **Help()**

MAny KCancel: **CleanupMenuBar()**

Action Routines

The XmCascadeButton action routines are

CleanupMenuBar():

In a MenuBar, disarms the CascadeButton and the menu and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the menu was entered.

In a toplevel Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu.

In a Popup MenuPane, unposts the menu and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget from which the menu was posted.

DoSelect(): Calls the callbacks in **XmNcascadingCallback**, posts the submenu attached to the CascadeButton and enables keyboard traversal within the menu. If the CascadeButton does not have a submenu attached, this action calls the callbacks in **XmNactivateCallback**, activates the CascadeButton, and unposts all posted menus in the cascade.

Help(): Unposts all menus in the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the menu system was entered. Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

KeySelect(): Calls the callbacks in **XmNcascadingCallback**, and posts the submenu attached to the CascadeButton if keyboard traversal is enabled in the menu. If the CascadeButton does not have a submenu attached, this action calls the callbacks in **XmNactivateCallback**, activates the CascadeButton, and unposts all posted menus in the cascade.

MenuBarSelect():
Unposts any menus posted by the parent menu. Arms both the CascadeButton and the MenuBar, posts the associated submenu, and enables mouse traversal. If the menu is already active, this event disables keyboard traversal for the menu and returns the menu to mouse traversal mode.

StartDrag(): Arms the CascadeButton, posts the associated submenu, and enables mouse traversal. If the menu is already active, this event disables keyboard traversal for the menu and returns the menu to mouse traversal mode.

Additional Behavior

Posting a submenu calls the **XmNcascadingCallback** callbacks. This widget has the following additional behavior:

<EnterWindow>:
If keyboard traversal is enabled, does nothing. Otherwise, in a MenuBar that is armed, unposts any MenuPanes associated with another MenuBar entry, arms the CascadeButton, and posts the associated submenu. In other menus, arms the CascadeButton and posts the associated submenu after the delay specified by **XmNmappingDelay**.

<LeaveWindow>:

If keyboard traversal is enabled does nothing. Otherwise, in a MenuBar that is armed, disarms the CascadeButton if the submenu associated with the CascadeButton is not currently posted or if there is no submenu associated with the CascadeButton.

In other menus, if the pointer moves anywhere except into a submenu associated with the CascadeButton, the CascadeButton is disarmed and its submenu is unposted.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCascadeButtonHighlight(3X)**, **XmCreateCascadeButton(3X)**,**XmCreateMenuBar(3X)**, **XmCreatePulldownMenu(3X)**, **XmCreatePopupMenu(3X)**, **XmLabel(3X)**, **XmPrimitive(3X)**, and **XmRowColumn(3X)**.

XmCascadeButtonGadget—The CascadeButtonGadget widget class

Synopsis

#include <Xm/CascadeBG.h>

Description

CascadeButtonGadget links two MenuPanes, a MenuBar to a MenuPane, or an OptionMenu to a MenuPane.

It is used in menu systems and must have a RowColumn parent with its **XmNrowColumnType** resource set to **XmMENU_BAR**, **XmMENU_POPUP**, **XmMENU_PULLDOWN**, or **XmMENU_OPTION**.

It is the only gadget that can have a Pulldown MenuPane attached to it as a submenu. The submenu is displayed when this gadget is activated within a PopupMenu, a PulldownMenu, or an OptionMenu. Its visuals can include a label or pixmap and a cascading indicator when it is in a Popup or Pulldown MenuPane; or it can include only a label or a pixmap when it is in an OptionMenu.

The default behavior associated with a CascadeButtonGadget depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the CascadeButtonGadget. In addition, **BMenu** controls the behavior of the CascadeButtonGadget if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

A CascadeButtonGadget's visuals differ from most other button gadgets. When the button becomes armed, its visuals change from a 2-D to a 3-D look, and it displays the submenu that has been attached to it. If no submenu is attached, it simply changes its visuals.

When a CascadeButtonGadget within a Pulldown or Popup MenuPane is armed as the result of the user moving the mouse pointer into the gadget, it does not immediately display its submenu. Instead, it waits a short time to see if the arming was temporary (that is, the user was simply passing through the gadget), or the user really wanted the submenu posted. This delay is configurable using **XmNmappingDelay**.

CascadeButtonGadget provides a single mechanism for activating the gadget from the keyboard. This mechanism is referred to as a keyboard mnemonic. If a mnemonic has been specified for the gadget, the user may activate it by simply typing the mnemonic while the CascadeButtonGadget is visible. If the CascadeButtonGadget is in a MenuBar and the MenuBar does not have focus, the **MAlt** modifier must be pressed with the mnemonic. Mnemonics are typically used to interact with a menu using the keyboard.

If a CascadeButtonGadget is in a Pulldown or Popup MenuPane and there is a submenu attached, the **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight**, and **XmNmarginTop** resources may enlarge to accommodate **XmNcascadePixmap**. **XmNmarginWidth** defaults to 6 if this resource is in a MenuBar; otherwise, it takes LabelGadget's default, which is 2.

Classes

CascadeButtonGadget inherits behavior and resources from the **Object**, **RectObj**, **XmGadget**, and **XmLabelGadget** classes.

The class pointer is **xmCascadeButtonGadgetClass**.

The class name is **XmCascadeButtonGadget**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmCascadeButtonGadget		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNcascadePixmap XmCPixmap	dynamic Pixmap	CSG
XmNcascadingCallback XmCCallback	NULL XtCallbackList	C
XmNmappingDelay XmCMappingDelay	180 ms int	CSG
XmNsubMenuId XmCMenuWidget	NULL Widget	CSG

XmNactivateCallback

Specifies the list of callbacks that is called when the user activates the CascadeButtonGadget, and there is no submenu attached to pop up. The activation occurs when a mouse button is released or when

the mnemonic associated with the gadget is typed. The specific mouse button depends on information in the RowColumn parent. The reason sent by the callback is **XmCR_ACTIVATE**.

XmNcascadePixmap

Specifies the cascade pixmap displayed on one end of the gadget when a CascadeButtonGadget is used within a Popup or Pulldown MenuPane and a submenu is attached. The LabelGadget class resources **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight**, and **XmNmarginTop** may be modified to ensure that room is left for the cascade pixmap. The default cascade pixmap in menus other than option menus is an arrow pointing to the side of the menu where the submenu will appear. The default for the CascadeButtonGadget in an option menu is **XmUNSPECIFIED_PIXMAP**.

XmNcascadingCallback

Specifies the list of callbacks that is called just prior to the mapping of the submenu associated with the CascadeButtonGadget. The reason sent by the callback is **XmCR_CASCADING**.

XmNmappingDelay

Specifies the amount of time, in milliseconds, between when a CascadeButtonGadget becomes armed and when it maps its submenu. This delay is used only when the gadget is within a Popup or Pulldown MenuPane. The value must not be negative.

XmNsubMenuId

Specifies the widget ID for the Pulldown MenuPane to be associated with this CascadeButtonGadget. The specified MenuPane is displayed when the CascadeButtonGadget becomes armed. The MenuPane must have been created with the appropriate parentage depending on the type of menu used. See **XmCreatePulldownMenu(3X)**, **XmCreatePopupMenu(3X)**, and **XmCreateOptionMenu(3X)** for more information on the menu systems.

Inherited Resources

CascadeButtonGadget inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the reference page for that superclass.

XmLabelGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	N/A
XmNacceleratorText XmCAcceleratorText	NULL XmString	N/A
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	dynamic Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	0 Dimension	CSG
XmNmarginRight XmCMarginRight	dynamic Dimension	CSG
XmNmarginTop XmCMarginTop	dynamic Dimension	CSG
XmNmarginWidth XmCMarginWidth	dynamic Dimension	CSG

Name Class	Default Type	Access
XmNmnemonic XmCMnemonic	NULL KeySym	CSG
XmNmnemonicCharSet XmCMnemonicCharSet	dynamic String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	G
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	G
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	0 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	G

Name Class	Default Type	Access
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

RectObj Resource Set

Name Class	Default Type	Access
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	0 Dimension	N/A
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Object Resource Set

Name Class	Default Type	Access
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback or is NULL if this callback was not triggered by an **XEvent**

Behavior

XmCascadeButtonGadget includes behavior from **XmGadget**. **XmCascadeButton** includes the menu traversal behavior from **XmLabel**. Additional **XmCascadeButtonGadget** behavior is described in the following list (in a Popup menu system, **BMenu** also performs the **BSelect** actions).

BSelect Press:
Unposts any menus posted by the parent menu. Arms the CascadeButtonGadget, posts the associated submenu, enables mouse traversal, and, in a MenuBar, arms the MenuBar. If the menu is already active, this event disables keyboard traversal for the menu and returns the menu to mouse traversal mode.

BSelect Release:
Calls the callbacks in **XmNcascadingCallback**, posts the submenu attached to the CascadeButtonGadget and enables keyboard traversal within the menu. If the CascadeButtonGadget does not have a submenu attached, this action calls the callbacks in **XmNactivateCallback**, activates the CascadeButtonGadget, and unposts all posted menus in the cascade.

KActivate: Calls the callbacks in **XmNcascadingCallback**, and posts the submenu attached to the CascadeButtonGadget if keyboard traversal is enabled in the menu. If the CascadeButtonGadget does not have a submenu attached, this action calls the callbacks in **XmNactivateCallback**, activates the CascadeButtonGadget, and unposts all posted menus in the cascade. This action applies only to gadgets in MenuBars, PulldownMenus, and PopupMenus. For a CascadeButtonGadget in an OptionMenu, if the parent is a manager, this action passes the event to the parent.

KSelect: Calls the callbacks in **XmNcascadingCallback**, and posts the submenu attached to the CascadeButtonGadget if keyboard traversal is enabled in the menu. If the CascadeButtonGadget does not have a submenu attached, this action calls the callbacks in **XmNactivateCallback**, activates the CascadeButtonGadget, and unposts all posted menus in the cascade.

KHelp: Unposts all menus in the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the menu system was entered. Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MAny KCancel:

In a MenuBar, disarms the CascadeButtonGadget and the menu and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the menu was entered. For a CascadeButtonGadget in an OptionMenu, if the parent is a manager, this action passes the event to the parent.

In a toplevel Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu.

In a Popup MenuPane, unposts the menu and restores keyboard focus to the widget from which the menu was posted.

<Enter>: If keyboard traversal is enabled does nothing. Otherwise, in a MenuBar, unposts any MenuPanes associated with another MenuBar entry, arms the CascadeButtonGadget, and posts the associated submenu. In other menus, arms the CascadeButtonGadget and posts the associated submenu after the delay specified by **XmNmappingDelay**.

<Leave>: If keyboard traversal is enabled does nothing. Otherwise, in a MenuBar, disarms the CascadeButtonGadget if the submenu associated with the CascadeButtonGadget is not currently posted or if there is no submenu associated with the CascadeButtonGadget.

In other menus, if the pointer moves anywhere except into a submenu associated with the CascadeButtonGadget, the CascadeButtonGadget is disarmed and its submenu is unposted.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Object(3X), **RectObj(3X)**, **XmCascadeButtonHighlight(3)**, **XmCreateCascadeButtonGadget(3X)**, **XmCreatePulldownMenu(3X)**, **XmCreatePopupMenu(3X)**, **XmCreateOptionMenu(3X)**,**XmGadget(3X)**, **XmLabelGadget(3X)**, and **XmRowColumn(3X)**.

XmCascadeButtonGadgetHighlight—A CascadeButtonGadget function that sets the highlight state

Synopsis

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

void XmCascadeButtonGadgetHighlight (cascadeButtonGadget, highlight)
        Widget      cascadeButtonGadget;
        Boolean     highlight;
```

Description

XmCascadeButtonGadgetHighlight either draws or erases the shadow highlight around the CascadeButtonGadget.

cascadeButtonGadget
Specifies the CascadeButtonGadget to be highlighted or unhighlighted

highlight Specifies whether to highlight (True) or to unhighlight (False)

For a complete definition of CascadeButtonGadget and its associated resources, see **XmCascadeButtonGadget(3X)**.

Related Information

XmCascadeButton(3X), **XmCascadeButtonGadget(3X)**, and **XmCascadeButtonHighlight(3X)**.

XmCascadeButtonHighlight—A CascadeButton and CascadeButtonGadget function that sets the highlight state

Synopsis

```
#include <Xm/CascadeB.h>
#include <Xm/CascadeBG.h>

void XmCascadeButtonHighlight (cascadeButton, highlight)
        Widget      cascadeButton;
        Boolean     highlight;
```

Description

XmCascadeButtonHighlight either draws or erases the shadow highlight around the CascadeButton or the CascadeButtonGadget.

cascadeButton
Specifies the CascadeButton or CascadeButtonGadget to be highlighted or unhighlighted

highlight Specifies whether to highlight (True) or to unhighlight (False)

For a complete definition of CascadeButton or CascadeButtonGadget and their associated resources, see **XmCascadeButton(3X)** or **XmCascadeButtonGadget(3X)**.

Related Information

XmCascadeButton(3X), **XmCascadeButtonGadget(3X)** and **XmCascadeButtonGadgetHighlight(3X)**.

XmChangeColor—Recalculates all associated colors of a widget

Synopsis

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

void XmChangeColor (widget, background)
        Widget      widget;
        Pixel       background;
```

Description

XmChangeColor handles all color modifications for the specified widget when a new background pixel value is specified. This function recalculates the foreground, select, and shadow colors based on the new background color and sets the corresponding resources for the widget. If a color calculation procedure has been set by a call to **XmSetColorCalculation**, **XmChangeColor** uses that procedure to calculate the new colors. Otherwise, the routine uses a default procedure.

widget Specifies the widget ID whose colors will be updated

background Specifies the background color pixel value

Related Information

XmGetColorCalculation(3X), **XmGetColors(3X)**, and **XmSetColorCalculation(3X)**.

XmClipboardCancelCopy—A clipboard function that cancels a copy to the clipboard

Synopsis

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

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

Description

XmClipboardCancelCopy cancels the copy to clipboard that is in progress and frees up temporary storage. When a copy is to be performed, **XmClipboardStartCopy** allocates temporary storage for the clipboard data. **XmClipboardCopy** copies the appropriate data into the the temporary storage. **XmClipboardEndCopy** copies the data to the clipboard structure and frees up the temporary storage structures. If **XmClipboardCancelCopy** is called, the **XmClipboardEndCopy** function does not have to be called. A call to **XmClipboardCancelCopy** is valid only after a call to **XmClipboardStartCopy** and before a call to **XmClipboardEndCopy**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies a widget's window ID that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to **XmClipboardStartCopy**.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardFail
The function failed because **XmClipboardStartCopy** was not called or because the data item contains too many formats.

Related Information

XmClipboardCopy(3X), **XmClipboardEndCopy(3X)**, and **XmClipboardStartCopy(3X)**.

XmClipboardCopy—A clipboard function that copies a data item to temporary storage for later copying to clipboard

Synopsis

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

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

Description

XmClipboardCopy copies a data item to temporary storage. The data item is moved from temporary storage to the clipboard data structure when a call to **XmClipboardEndCopy** is made. Additional calls to **XmClipboardCopy** before a call to **XmClipboardEndCopy** add additional data item formats to the same data item or append data to an existing format. Formats are described in the *Inter-Client Communication Conventions Manual* (ICCCM) as targets.

NOTE: Do not call **XmClipboardCopy** before a call to **XmClipboardStartCopy** has been made. The latter function allocates temporary storage required by **XmClipboardCopy**.

If the *buffer* argument is NULL, the data is considered to be passed by name. When data that has been passed by name is later requested by another application, the application that owns the data receives a callback with a request for the data. The application that owns the data must then transfer the data to the clipboard with the **XmClipboardCopyByName** function. When a data item that was passed by name is deleted from the clipboard, the application that owns the data receives a callback stating that the data is no longer needed.

For information on the callback function, see the callback argument description for **XmClipboardStartCopy**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to **XmClipboardStartCopy**.

format_name Specifies the name of the format in which the data item is stored on the clipboard. The format was known as target in the ICCCM.

buffer Specifies the buffer from which the clipboard copies the data.

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

private_id Specifies the private data that the application wants to store with the data item.

data_id Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This argument is required only for data that is passed by name.

Return Value

ClipboardSuccess
The function is successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardFail
The function failed because **XmClipboardStartCopy** was not called or because the data item contains too many formats.

Related Information

XmClipboardCopyByName(3X), **XmClipboardEndCopy(3X)**, and **XmClipboardStartCopy(3X)**.

XmClipboardCopyByName—A clipboard function that copies a data item passed by name

Synopsis

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

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

Description

XmClipboardCopyByName copies the actual data for a data item that was previously passed by name to the clipboard. Data is considered to be passed by name when a call to **XmClipboardCopy** is made with a NULL buffer parameter. Additional calls to this function append new data to the existing data.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained by through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.

data_id Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This number was assigned by **XmClipboardCopy** to the data item.

buffer Specifies the buffer from which the clipboard copies the data.

length Specifies the number of bytes in the data item.

private_id Specifies the private data that the application wants to store with the data item.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardCopy(3X), **XmClipboardLock(3X)**, **XmClipboardStartCopy(3X)**, and **XmClipboardUnlock(3X)**.

XmClipboardEndCopy—A clipboard function that ends a copy to the clipboard

Synopsis

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

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

Description

XmClipboardEndCopy locks the clipboard from access by other applications, places data in the clipboard data structure, and unlocks the clipboard. Data items copied to the clipboard by **XmClipboardCopy** are not actually entered in the clipboard data structure until the call to **XmClipboardEndCopy**.

This function also frees up temporary storage that was allocated by **XmClipboardStartCopy**, which must be called before **XmClipboardEndCopy**. The latter function should not be called if **XmClipboardCancelCopy** has been called.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.

item_id Specifies the number assigned to this data item, which was returned by a previous call to **XmClipboardStartCopy**.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardFail
The function failed because **XmClipboardStartCopy** was not called.

Related Information

XmClipboardCancelCopy(3X), **XmClipboardCopy(3X)** and **XmClipboardStartCopy(3X)**.

XmClipboardEndRetrieve—A clipboard function that ends a copy from the clipboard

Synopsis

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

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

Description

XmClipboardEndRetrieve suspends copying data incrementally from the clipboard. It tells the clipboard routines that the application is through copying an item from the clipboard. Until this function is called, data items can be retrieved incrementally from the clipboard with **XmClipboardRetrieve**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained with **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardRetrieve(3X), **XmClipboardStartCopy(3X)**, and **XmClipboardStartRetrieve(3X)**.

XmClipboardInquireCount—A clipboard function that returns the number of data item formats

Synopsis

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

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

Description

XmClipboardInquireCount returns the number of data item formats available for the data item in the clipboard. This function also returns the maximum name-length for all formats in which the data item is stored.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

count Returns the number of data item formats available for the data item in the clipboard. If no formats are available, this argument equals 0 (zero). The count includes the formats that were passed by name.

max_format_name_length
Specifies the maximum length of all format names for the data item in the clipboard.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardNoData
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardStartCopy(3X).

XmClipboardInquireFormat—A clipboard function that returns a specified format name

Synopsis

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

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

Description

XmClipboardInquireFormat returns a specified format name for the data item in the clipboard. If the name must be truncated, the function returns a warning status.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

index Specifies which of the ordered format names to obtain. If this index is greater than the number of formats for the data item, this function returns a 0 (zero) in the *copied_len* argument.

format_name_buf
Specifies the buffer that receives the format name.

buffer_len Specifies the number of bytes in the format name buffer.

copied_len Specifies the number of bytes in the string copied to the buffer. If this argument equals 0 (zero), there is no *n*th format for the data item.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardTruncate
The data returned is truncated because the user did not provide a buffer large enough to hold the data.

ClipboardNoData
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardStartCopy(3X).

XmClipboardInquireLength—A clipboard function that returns the length of the stored data

Synopsis

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

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

Description

XmClipboardInquireLength returns the length of the data stored under a specified format name for the clipboard data item. If no data is found for the specified format, or if there is no item on the clipboard, this function returns a value of 0 (zero).

Any format passed by name is assumed to have *length* passed in a call to **XmClipboardCopy**, even though the data has not yet been transferred to the clipboard in that format.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name Specifies the name of the format for the data item.

length Specifies the length of the next data item in the specified format. This argument equals 0 (zero) if no data is found for the specified format, or if there is no item on the clipboard.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardNoData
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardCopy(3X) and **XmClipboardStartCopy(3X)**.

XmClipboardInquirePendingItems—A clipboard function that returns a list of data ID/private ID pairs

Synopsis

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

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

Description

XmClipboardInquirePendingItems returns a list of data ID/private ID pairs for the specified format name. A data item is considered pending if the application originally passed it by name, the application has not yet copied the data, and the item has not been deleted from the clipboard. The application is responsible for freeing the memory provided by this function to store the list. To free the memory, call **XtFree**.

This function is used by an application when exiting, to determine if the data that is passed by name should be sent to the clipboard.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name Specifies a string that contains the name of the format for which the list of data ID/private ID pairs is to be obtained.

item_list Specifies the address of the array of data ID/private ID pairs for the specified format name. This argument is a type **XmClipboardPendingList**. The application is responsible for freeing the memory provided by this function for storing the list.

count Specifies the number of items returned in the list. If there is no data for the specified format name, or if there is no item on the clipboard, this argument equals 0 (zero).

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardStartCopy(3X).

XmClipboardLock—A clipboard function that locks the clipboard

Synopsis

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

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

Description

XmClipboardLock locks the clipboard from access by another application until **XmClipboardUnlock** is called. All clipboard functions lock and unlock the clipboard to prevent simultaneous access. This function allows the application to keep the clipboard data from changing between calls to **Inquire** and other clipboard functions. The application does not need to lock the clipboard between calls to **XmClipboardStartCopy** and **XmClipboardEndCopy** or to **XmClipboardStartRetrieve** and **XmClipboardEndRetrieve**.

If the clipboard is already locked by another application, **XmClipboardLock** returns an error status. Multiple calls to this function by the same application increases the lock level.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardEndCopy(3X), **XmClipboardEndRetrieve(3X)**, **XmClipboardStartCopy(3X)**, **XmClipboardStartRetrieve(3X)**, and **XmClipboardUnlock(3X)**.

XmClipboardRegisterFormat—A clipboard function that registers a new format

Synopsis

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

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

Description

XmClipboardRegisterFormat registers a new format. Each format stored on the clipboard should have a length associated with it; this length must be known to the clipboard routines. Formats are known as targets in the *Inter-Client Communication Conventions Manual* (ICCCM). All of the formats specified by the ICCCM conventions are preregistered. Any other format that the application wants to use must either be 8-bit data or be registered through this routine. Failure to register the length of the data results in incompatible applications across platforms having different byte-swapping orders.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

format_name Specifies the string name for the new format (target).

format_length Specifies the format length in bits (8, 16, or 32).

Return Value

ClipboardBadFormat
The *format_name* must not be NULL, and the *format_length* must be 8, 16, or 32.

ClipboardSuccess
The function was successful.

ClipboardLocked

The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardFail

The function failed because the format was already registered with this length.

Related Information

XmClipboardStartCopy(3X).

XmClipboardRetrieve—A clipboard function that retrieves a data item from the clipboard

Synopsis

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

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

Description

XmClipboardRetrieve retrieves the current data item from clipboard storage. It returns a warning if the clipboard is locked, if there is no data on the clipboard, or if the data needs to be truncated because the buffer length is too short.

Between a call to **XmClipboardStartRetrieve** and a call to **XmClipboardEndRetrieve**, multiple calls to **XmClipboardRetrieve** with the same format name result in data being incrementally copied from the clipboard until the data in that format has all been copied.

The return value **ClipboardTruncate** from calls to **XmClipboardRetrieve** indicates that more data remains to be copied in the given format. It is recommended that any calls to the **Inquire** functions that the application needs to make to effect the copy from the clipboard be made between the call to **XmClipboardStartRetrieve** and the first call to **XmClipboardRetrieve**. This way, the application does not need to call **XmClipboardLock** and **XmClipboardUnlock**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name Specifies the name of a format in which the data is stored on the clipboard.

buffer Specifies the buffer to which the application wants the clipboard to copy the data.

length Specifies the length of the application buffer.

num_bytes Specifies the number of bytes of data copied into the application buffer.

private_id Specifies the private data stored with the data item by the application that placed the data item on the clipboard. If the application did not store private data with the data item, this argument returns 0 (zero).

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

ClipboardTruncate
The data returned is truncated because the user did not provide a buffer large enough to hold the data.

ClipboardNoData
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard but not in the requested format; or the data in the requested format was passed by name and is no longer available.

Related Information

XmClipboardEndRetrieve(3X), **XmClipboardLock(3X)**, **XmClipboardStartCopy(3X)**, **XmClipboardStartRetrieve(3X)**, and **XmClipboardUnlock(3X)**.

XmClipboardStartCopy—A clipboard function that sets up a storage and data structure

Synopsis

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

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

Description

XmClipboardStartCopy sets up storage and data structures to receive clipboard data. An application calls this function during a cut or copy operation. The data item that these structures receive then becomes the next data item in the clipboard.

Copying a large piece of data to the clipboard can take a long time. It is possible that, once copied, no application will ever request that data. The Motif Toolkit provides a mechanism so that an application does not need to actually pass data to the clipboard until the data has been requested by some application.

Instead, the application passes format and length information in **XmClipboardCopy** to the clipboard functions, along with a widget ID and a callback function address that is passed in **XmClipboardStartCopy**. The widget ID is necessary for communications between the clipboard functions in the application that owns the data and the clipboard functions in the application that requests the data.

The callback functions are responsible for copying the actual data to the clipboard through **XmClipboardCopyByName**. The callback function is also called if the data item is removed from the clipboard and the actual data is no longer needed.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

clip_label Specifies the label to be associated with the data item. This argument is used to identify the data item, for example, in a clipboard viewer. An example of a label is the name of the application that places the data in the clipboard.

timestamp Specifies the time of the event that triggered the copy. A valid timestamp must be supplied; it is not sufficient to use **CurrentTime**.

widget Specifies the ID of the widget that receives messages requesting data previously passed by name. This argument must be present in order to pass data by name. Any valid widget ID in your application can be used for this purpose and all the message handling is taken care of by the cut and paste functions.

callback Specifies the address of the callback function that is called when the clipboard needs data that was originally passed by name. This is also the callback to receive the **delete** message for items that were originally passed by name. This argument must be present in order to pass data by name.

item_id Specifies the number assigned to this data item. The application uses this number in calls to **XmClipboardCopy**, **XmClipboardEndCopy**, and **XmClipboardCancelCopy**.

For more information on passing data by name, see **XmClipboardCopy(3X)** and **XmClipboardCopyByName(3X)**.

The *widget* and *callback* arguments must be present in order to pass data by name. The callback format is as follows:

void (**callback*) (*widget, data_id, private, reason*)
Widget *widget*;
int **data_id*;
int **private*;
int **reason*;

widget Specifies the ID of the widget passed to this function.

data_id Specifies the identifying number returned by **XmClipboardCopy**, which identifies the pass-by-name data.

private Specifies the private information passed to **XmClipboardCopy**.

reason Specifies the reason. The reason can be either one of the following: **XmCR_CLIPBOARD_DATA_DELETE** or **XmCR_CLIPBOARD_DATA_REQUEST**.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardCancelCopy(3X), **XmClipboardCopy(3X)**, **XmClipboardCopyByName(3X)**, **XmClipboardEndCopy(3X)**, **XmClipboardEndRetrieve(3X)**, **XmClipboardInquireCount(3X)**, **XmClipboardInquireFormat(3X)**, **XmClipboardInquireLength(3X)**, **XmClipboardInquirePendingItems(3X)**, **XmClipboardLock(3X)**, **XmClipboardRegisterFormat(3X)**, **XmClipboardRetrieve(3X)**, **XmClipboardStartRetrieve(3X)**, **XmClipboardUndoCopy(3X)**, **XmClipboardUnlock(3X)**, and **XmClipboardWithdrawFormat(3X)**.

XmClipboardStartRetrieve—A clipboard function that starts a copy from the clipboard

Synopsis

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

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

Description

XmClipboardStartRetrieve tells the clipboard routines that the application is ready to start copying an item from the clipboard. The clipboard is locked by this routine and stays locked until **XmClipboardEndRetrieve** is called. Between a call to **XmClipboardStartRetrieve** and a call to **XmClipboardEndRetrieve**, multiple calls to **XmClipboardRetrieve** with the same format name result in data being incrementally copied from the clipboard until the data in that format has all been copied.

A return value of **ClipboardTruncate** from calls to **XmClipboardRetrieve** indicates that more data remains to be copied in the given format. It is recommended that any calls to the **Inquire** functions that the application needs to make to complete the copy from the clipboard be made between the call to **XmClipboardStartRetrieve** and the first call to **XmClipboardRetrieve**. This way, the application does not need to call **XmClipboardLock** and **XmClipboardUnlock**.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

timestamp Specifies the time of the event that triggered the copy. A valid timestamp must be supplied; it is not sufficient to use **CurrentTime**.

Return Value

ClipboardSuccess
The function is successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardEndRetrieve(3X), **XmClipboardInquireCount(3X)**, **XmClipboardInquireFormat(3X)**, **XmClipboardInquireLength(3X)**, **XmClipboardInquirePendingItems(3X)**, **XmClipboardLock(3X)**, **XmClipboardRetrieve(3X)**, **XmClipboardStartCopy(3X)**, and **XmClipboardUnlock(3X)**.

XmClipboardUndoCopy—A clipboard function that deletes the last item placed on the clipboard

Synopsis

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

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

Description

XmClipboardUndoCopy deletes the last item placed on the clipboard if the item was placed there by an application with the passed *display* and *window* arguments. Any data item deleted from the clipboard by the original call to **XmClipboardCopy** is restored. If the *display* or *window* IDs do not match the last copied item, no action is taken, and this function has no effect.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardLock(3X) and **XmClipboardStartCopy(3X)**.

XmClipboardUnlock—A clipboard function that unlocks the clipboard

Synopsis

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

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

Description

XmClipboardUnlock unlocks the clipboard, enabling it to be accessed by other applications.

If multiple calls to **XmClipboardLock** have occurred, the same number of calls to **XmClipboardUnlock** is necessary to unlock the clipboard, unless *remove_all_locks* is set to True.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

remove_all_locks
When True, indicates that all nested locks should be removed. When False, indicates that only one level of lock should be removed.

Return Value

ClipboardSuccess
The function was successful.

ClipboardFail
The function failed because the clipboard was not locked or was locked by another application.

Related Information

XmClipboardCancelCopy(3X), **XmClipboardCopy(3X)**, **XmClipboardEndCopy(3X)**, **XmClipboardEndRetrieve(3X)**, **XmClipboardInquireCount(3X)**, **XmClipboardInquireFormat(3X)**, **XmClipboardInquireLength(3X)**, **XmClipboardInquirePendingItems(3X)**, **XmClipboardLock(3X)**, **XmClipboardRegisterFormat(3X)**, **XmClipboardRetrieve(3X)**, **XmClipboardStartCopy(3X)**, **XmClipboardStartRetrieve(3X)**, **XmClipboardUndoCopy(3X)**, and **XmClipboardWithdrawFormat(3X)**.

XmClipboardWithdrawFormat—A clipboard function that indicates that the application no longer wants to supply a data item

Synopsis

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

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

Description

XmClipboardWithdrawFormat indicates that the application no longer supplies a data item to the clipboard that the application had previously passed by name.

display Specifies a pointer to the **Display** structure that was returned in a previous call to **XOpenDisplay** or **XtDisplay**.

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through **XtWindow**. The same application instance should pass the same window ID to each clipboard function it calls.

data_id Specifies an identifying number assigned to the data item, which uniquely identifies the data item and the format. This was assigned to the item when it was originally passed by **XmClipboardCopy**.

Return Value

ClipboardSuccess
The function was successful.

ClipboardLocked
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

Related Information

XmClipboardCopy(3X) and **XmClipboardStartCopy(3X)**.

XmCommand—The Command widget class

Synopsis

#include <Xm/Command.h>

Description

Command is a special-purpose composite widget for command entry that provides a built-in command-history mechanism. Command includes a command-line text-input field, a command-line prompt, and a command-history list region.

One additional **WorkArea** child may be added to the Command after creation.

Whenever a command is entered, it is automatically added to the end of the command-history list and made visible. This does not change the selected item in the list, if there is one.

Many of the new resources specified for Command are actually SelectionBox resources that have been renamed for clarity and ease of use.

Classes

Command inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, **XmBulletinBoard**, and **XmSelectionBox** classes.

The class pointer is **xmCommandWidgetClass**.

The class name is **XmCommand**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmCommand Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNcommand XmCTextString	"" XmString	CSG
XmNcommandChangedCallback XmCCallback	NULL XtCallbackList	C
XmNcommandEnteredCallback XmCCallback	NULL XtCallbackList	C
XmNhistoryItems XmCItems	NULL XmStringTable	CSG
XmNhistoryItemCount XmCItemCount	0 int	CSG
XmNhistoryMaxItems XmCMaxItems	100 int	CSG
XmNhistoryVisibleItemCount XmCVisibleItemCount	dynamic int	CSG
XmNpromptString XmCPromptString	dynamic XmString	CSG

XmNcommand

Contains the current command-line text. This is the **XmNtextString** resource in SelectionBox, renamed for Command. This resource can also be modified with **XmCommandSetValue** and **XmCommandAppendValue** functions. The command area is a Text widget.

XmNcommandChangedCallback

Specifies the list of callbacks that is called when the value of the command changes. The callback reason is **XmCR_COMMAND_CHANGED**. This is equivalent to the **XmNvalueChangedCallback** of the Text widget, except that a pointer to an **XmCommandCallbackStructure** is passed, and the structure's *value* member contains the **XmString**.

XmNcommandEnteredCallback
Specifies the list of callbacks that is called when a command is entered in the Command. The callback reason is **XmCR_COMMAND_ENTERED**. A pointer to an **XmCommandCallback** structure is passed.

XmNhistoryItems
Lists **XmString** items that make up the contents of the history list. This is the **XmNlistItems** resource in SelectionBox, renamed for Command. **XtGetValues** for this resource returns the list items themselves, not a copy of the list items. The application must not free the returned items.

XmNhistoryItemCount
Specifies the number of **XmStrings** in **XmNhistoryItems**. This is the **XmNlistItemCount** resource in SelectionBox, renamed for Command. The value must not be negative.

XmNhistoryMaxItems
Specifies the maximum number of items allowed in the history list. Once this number is reached, an existing list item must be removed before a new item can be added to the list. For each command entered, the first list item is removed from the list, so the new command can be added to the list. The value must be greater than 0 (zero).

XmNhistoryVisibleItemCount
Specifies the number of items in the history list that should be visible at one time. In effect, it sets the height (in lines) of the history list window. This is the **XmNlistVisibleItemCount** resource in SelectionBox, renamed for Command. The value must be greater than 0 (zero). The default is dynamic based on the height of the list.

XmNpromptString
Specifies a prompt for the command line. This is the **XmNselectionLabelString** resource in SelectionBox, renamed for Command. The default may vary depending on the value of the **XmNstringDirection** resource and the locale. In the C locale the default is > (right angle bracket).

Inherited Resources

Command inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmSelectionBox Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNapplyCallback XmCCallback	NULL XtCallbackList	N/A
XmNapplyLabelString XmCApplyLabelString	dynamic XmString	N/A
XmNcancelCallback XmCCallback	NULL XtCallbackList	N/A
XmNcancelLabelString XmCCancelLabelString	dynamic XmString	N/A
XmNchildPlacement XmCChildPlacement	XmPLACE_ABOVE_SELECTION unsigned char	CSG
XmNdialogType XmCDialogType	XmDIALOG_COMMAND unsigned char	G
XmNhelpLabelString XmCHelpLabelString	dynamic XmString	N/A
XmNlistItemCount XmCItemCount	0 int	CSG
XmNlistItems XmCItems	NULL XmStringTable	CSG
XmNlistLabelString XmCListLabelString	NULL XmString	N/A
XmNlistVisibleItemCount XmCVisibleItemCount	dynamic int	CSG
XmNminimizeButtons XmCMinimizeButtons	False Boolean	N/A

Name Class	Default Type	Access
XmNmustMatch XmCMustMatch	False Boolean	N/A
XmNnoMatchCallback XmCCallback	NULL XtCallbackList	N/A
XmNokCallback XmCCallback	NULL XtCallbackList	N/A
XmNokLabelString XmCOkLabelString	dynamic XmString	N/A
XmNselectionLabelString XmCSelectionLabelString	dynamic XmString	CSG
XmNtextAccelerators XmCTextAccelerators	default XtAccelerators	C
XmNtextColumns XmCColumns	dynamic short	CSG
XmNtextString XmCTextString	"" XmString	CSG

XmBulletinBoard Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowOverlap XmCAllowOverlap	True Boolean	CSG
XmNautoUnmanage XmCAutoUnmanage	False Boolean	N/A
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	N/A
XmNcancelButton XmCWidget	NULL Widget	N/A
XmNdefaultButton XmCWidget	NULL Widget	N/A
XmNdefaultPosition XmCDefaultPosition	False Boolean	CSG
XmNdialogStyle XmCDialogStyle	dynamic unsigned char	CSG
XmNdialogTitle XmCDialogTitle	NULL XmString	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	10 Dimension	CSG
XmNmarginWidth XmCMarginWidth	10 Dimension	CSG
XmNnoResize XmCNoResize	False Boolean	CSG
XmNresizePolicy XmCResizePolicy	XmRESIZE_NONE unsigned char	CSG

Name Class	Default Type	Access
XmNshadowType XmCShadowType	XmSHADOW_OUT unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNtextTranslations XmCTranslations	NULL XtTranslations	C
XmNunmapCallback XmCCallback	NULL XtCallbackList	C

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	dynamic Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	N/A
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    XmString    value;
    int         length;
} XmCommandCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

value Specifies the **XmString** in the CommandArea

length Specifies the size of the command in **XmString**

Translations

XmCommand inherits translations from **XmSelectionBox**.

Accelerators

The **XmNtextAccelerators** from **XmSelectionBox** are added to the Text descendant of **XmCommand**.

Action Routines

The **XmCommand** action routines are described below:

SelectionBoxUpOrDown(0|1|2|3):

When called with an argument of 0 (zero), selects the previous item in the history list and replaces the text with that item.

When called with an argument of 1, selects the next item in the history list and replaces the text with that item.

When called with an argument of 2, selects the first item in the history list and replaces the text with that item.

When called with an argument of 3, selects the last item in the history list and replaces the text with that item.

Calls the callbacks for **XmNcommandChangedCallback**.

Additional Behavior

The Command widget has the following additional behavior:

MAny KCancel:

If the parent of the Command is a manager, the event is passed to the parent.

KActivate in Text:

Calls the Text widget's **XmNactivateCallback** callbacks. If the text is empty, this action then returns. Otherwise, if the history list has **XmNhistoryMaxItems** items, it removes the first item in the list. It adds the text to the history list as the last item, clears the text, and calls the **XmNcommandEnteredCallback** callbacks.

<Key> in Text:

When any change is made to the text edit widget, this action calls the callbacks for **XmNcommandChangedCallback**.

<DoubleClick> or **<KActivate>** in List:

Calls the List widget's **XmNdefaultActionCallback** callbacks. If the history list has **XmNhistoryMaxItems** items, this action removes the first item in the list. It adds the selected List item to the history list as the last item, clears the text, and calls the **XmNcommandEnteredCallback** callbacks.

<FocusIn>: Calls the callbacks for **XmNfocusCallback**.

<MapWindow>:
When a Command that is the child of a DialogShell is mapped, this action calls the callbacks for **XmNmapCallback**.

<UnmapWindow>:
When a Command that is the child of a DialogShell is unmapped, this action calls the callbacks for **XmNunmapCallback**.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmBulletinBoard(3X)**, **XmCommandAppendValue(3X)**, **XmCommandError(3X)**, **XmCommandGetChild(3X)**, **XmCommandSetValue(3X)**, **XmCreateCommand(3X)**, **XmManager(3X)**, and **XmSelectionBox(3X)**.

XmCommandAppendValue—A Command function that appends the passed XmString to the end of the string displayed in the command area of the widget

Synopsis

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

void XmCommandAppendValue (widget, command)
        Widget      widget;
        XmString    command;
```

Description

XmCommandAppendValue appends the passed **XmString** to the end of the string displayed in the command area of the Command widget.

widget Specifies the Command widget ID

command Specifies the passed **XmString**

For a complete definition of Command and its associated resources, see **XmCommand(3X)**.

Related Information

XmCommand(3X).

XmCommandError—A Command function that displays an error message

Synopsis

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

void XmCommandError (widget, error)
        Widget      widget;
        XmString    error;
```

Description

XmCommandError displays an error message in the history area of the Command widget. The **XmString** error is displayed until the next command entered occurs.

widget Specifies the Command widget ID

error Specifies the passed **XmString**

For a complete definition of Command and its associated resources, see **XmCommand(3X)**.

Related Information

XmCommand(3X).

XmCommandGetChild—A Command function that is used to access a component

Synopsis

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

Widget XmCommandGetChild (widget, child)
        Widget          widget;
        unsigned char   child;
```

Description

XmCommandGetChild is used to access a component within a Command. The parameters given to the function are the Command widget and a value indicating which component to access.

widget Specifies the Command widget ID.

child Specifies a component within the Command. The following values are legal for this parameter:

- **XmDIALOG_COMMAND_TEXT**
- **XmDIALOG_PROMPT_LABEL**
- **XmDIALOG_HISTORY_LIST**
- **XmDIALOG_WORK_AREA**

For a complete definition of Command and its associated resources, see **XmCommand(3X)**.

Return Value

Returns the widget ID of the specified Command component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmCommand(3X).

XmCommandSetValue—A Command function that replaces a displayed string

Synopsis

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

void XmCommandSetValue (widget, command)
        Widget      widget;
        XmString    command;
```

Description

XmCommandSetValue replaces the string displayed in the command area of the Command widget with the passed **XmString**.

widget Specifies the Command widget ID

command Specifies the passed **XmString**

For a complete definition of Command and its associated resources, see **XmCommand(3X)**.

Related Information

XmCommand(3X).

XmConvertUnits—A function that converts a value in one unit type to another unit type

Synopsis

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

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

Description

XmConvertUnits converts the value and returns it as the return value from the function.

widget Specifies the widget for which the data is to be converted.

orientation Specifies whether the converter uses the horizontal or vertical screen resolution when performing the conversions. The *orientation* parameter can have values of **XmHORIZONTAL** or **XmVERTICAL**.

from_unit_type
Specifies the current unit type of the supplied value.

from_value Specifies the value to be converted.

to_unit_type Converts the value to the unit type specified.

The parameters *from_unit_type* and *to_unit_type* can have the following values:

XmPIXELS All values provided to the widget are treated as normal pixel values. This is the default for the resource.

Xm100TH_MILLIMETERS
All values provided to the widget are treated as 1/100 of a millimeter.

Xm1000TH_INCHES
All values provided to the widget are treated as 1/1000 of an inch.

Xm100TH_POINTS
All values provided to the widget are treated as 1/100 of a point. A point is a unit typically used in text processing applications and is defined as 1/72 of an inch.

Xm100TH_FONT_UNITS
All values provided to the widget are treated as 1/100 of a font unit. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

Return Value

Returns the converted value. If a NULL widget, incorrect *orientation*, or incorrect *unit_type* is supplied as parameter data, 0 (zero) is returned.

Related Information

XmSetFontUnits(3X) and **XmScreen(3X)**.

XmCreateArrowButton—The ArrowButton widget creation function

Synopsis

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

Widget XmCreateArrowButton (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateArrowButton creates an instance of an ArrowButton widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ArrowButton and its associated resources, see **XmArrowButton(3X)**.

Return Value

Returns the ArrowButton widget ID.

Related Information

XmArrowButton(3X).

XmCreateArrowButtonGadget—The ArrowButtonGadget creation function

Synopsis

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

Widget XmCreateArrowButtonGadget (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateArrowButtonGadget creates an instance of an ArrowButtonGadget widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ArrowButtonGadget and its associated resources, see **XmArrowButtonGadget(3X)**.

Return Value

Returns the ArrowButtonGadget widget ID.

Related Information

XmArrowButtonGadget(3X).

XmCreateBulletinBoard—The BulletinBoard widget creation function

Synopsis

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

Widget XmCreateBulletinBoard (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateBulletinBoard creates an instance of a BulletinBoard widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of BulletinBoard and its associated resources, see **XmBulletinBoard(3X)**.

Return Value

Returns the BulletinBoard widget ID.

Related Information

XmBulletinBoard(3X).

XmCreateBulletinBoardDialog—The BulletinBoard BulletinBoardDialog convenience creation function

Synopsis

#include <Xm/BulletinB.h>

Widget XmCreateBulletinBoardDialog (*parent, name, arglist, argcount*)

Widget	*parent*;
String	*name*;
ArgList	*arglist*;
Cardinal	*argcount*;

Description

XmCreateBulletinBoardDialog is a convenience creation function that creates a DialogShell and an unmanaged BulletinBoard child of the DialogShell. A BulletinBoardDialog is used for interactions not supported by the standard dialog set. This function does not automatically create any labels, buttons, or other dialog components. Such components should be added by the application after the BulletinBoardDialog is created.

Use **XtManageChild** to pop up the BulletinBoardDialog (passing the BulletinBoard as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of BulletinBoard and its associated resources, see **XmBulletinBoard(3X)**.

Return Value

Returns the BulletinBoard widget ID.

Related Information

XmBulletinBoard(3X).

XmCreateCascadeButton—The CascadeButton widget creation function

Synopsis

#include <Xm/CascadeB.h>

Widget XmCreateCascadeButton (*parent, name, arglist, argcount*)

Widget	*parent*;
String	*name*;
ArgList	*arglist*;
Cardinal	*argcount*;

Description

XmCreateCascadeButton creates an instance of a CascadeButton widget and returns the associated widget ID.

parent Specifies the parent widget ID. The parent must be a RowColumn widget.

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of CascadeButton and its associated resources, see **XmCascadeButton(3X)**.

Return Value

Returns the CascadeButton widget ID.

Related Information

XmCascadeButton(3X).

XmCreateCascadeButtonGadget—The CascadeButtonGadget creation function

Synopsis

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

Widget XmCreateCascadeButtonGadget (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateCascadeButtonGadget creates an instance of a CascadeButtonGadget and returns the associated widget ID.

parent Specifies the parent widget ID. The parent must be a RowColumn widget.

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of CascadeButtonGadget and its associated resources, see **XmCascadeButtonGadget(3X)**.

Return Value

Returns the CascadeButtonGadget widget ID.

Related Information

XmCascadeButtonGadget(3X).

XmCreateCommand—The Command widget creation function

Synopsis

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

Widget XmCreateCommand (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateCommand creates an instance of a Command widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Command and its associated resources, see **XmCommand(3X)**.

Return Value

Returns the Command widget ID.

Related Information

XmCommand(3X).

XmCreateDialogShell—The DialogShell widget creation function

Synopsis

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

Widget XmCreateDialogShell (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateDialogShell creates an instance of a DialogShell widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DialogShell and its associated resources, see **XmDialogShell(3X)**.

Return Value

Returns the DialogShell widget ID.

Related Information

XmDialogShell(3X).

XmCreateDragIcon—A Drag and Drop function that creates a DragIcon widget

Synopsis

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

Widget XmCreateDragIcon (widget, name, arglist, argcount)
        Widget      widget;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateDragIcon creates a DragIcon and returns the associated widget ID.

widget Specifies the ID of the widget that the function uses to access default values for visual attributes of the DragIcon. This widget may be different than the actual parent of the DragIcon.

name Specifies the name of the DragIcon widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of DragIcon and its associated resources, see **XmDragIcon(3X)**.

Return Value

The function creates a DragIcon and returns the associated widget ID.

Related Information

XmDragContext(3X), **XmDragIcon(3X)**, and **XmScreen(3X)**.

XmCreateDrawingArea—The DrawingArea widget creation function

Synopsis

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

Widget XmCreateDrawingArea (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateDrawingArea creates an instance of a DrawingArea widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DrawingArea and its associated resources, see **XmDrawingArea(3X)**.

Return Value

Returns the DrawingArea widget ID.

Related Information

XmDrawingArea(3X).

XmCreateDrawnButton—The DrawnButton widget creation function

Synopsis

#include <Xm/DrawnB.h>

Widget XmCreateDrawnButton (*parent, name, arglist, argcount*)

Widget	*parent*;
String	*name*;
ArgList	*arglist*;
Cardinal	*argcount*;

Description

XmCreateDrawnButton creates an instance of a DrawnButton widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DrawnButton and its associated resources, see **XmDrawnButton(3X)**.

Return Value

Returns the DrawnButton widget ID.

Related Information

XmDrawnButton(3X).

XmCreateErrorDialog—The MessageBox ErrorDialog convenience creation function

Synopsis

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

Widget XmCreateErrorDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateErrorDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. An ErrorDialog warns the user of an invalid or potentially dangerous condition. It includes a symbol, a message, and three buttons. The default symbol is an octagon with a diagonal slash. The default button labels are **OK**, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the ErrorDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCreateFileSelectionBox—The FileSelectionBox widget creation function

Synopsis

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

Widget XmCreateFileSelectionBox (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateFileSelectionBox creates an unmanaged FileSelectionBox. A FileSelectionBox is used to select a file and includes the following:

- An editable text field for the directory mask
- A scrolling list of filenames
- An editable text field for the selected file
- Labels for the list and text fields
- Four buttons

The default button labels are **OK**, **Filter**, **Cancel**, and **Help**. Additional work area children may be added to the FileSelectionBox after creation. FileSelectionBox inherits the layout functionality provided by SelectionBox for any additional work area children.

If the parent of the FileSelectionBox is a DialogShell, use **XtManageChild** to pop up the FileSelectionDialog (passing the FileSelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox(3X)**.

Return Value

Returns the FileSelectionBox widget ID.

Related Information

XmFileSelectionBox(3X).

XmCreateFileSelectionDialog—The FileSelectionBox FileSelectionDialog convenience creation function

Synopsis

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

Widget XmCreateFileSelectionDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateFileSelectionDialog is a convenience creation function that creates a DialogShell and an unmanaged FileSelectionBox child of the DialogShell. A FileSelectionDialog selects a file. It includes the following:

- An editable text field for the directory mask
- A scrolling list of filenames
- An editable text field for the selected file
- Labels for the list and text fields
- Four buttons

The default button labels are **OK**, **Filter**, **Cancel**, and **Help**. One additional **WorkArea** child may be added to the FileSelectionBox after creation.

Use **XtManageChild** to pop up the FileSelectionDialog (passing the FileSelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox(3X)**.

Return Value

Returns the FileSelectionBox widget ID.

Related Information

XmFileSelectionBox(3X).

XmCreateForm—The Form widget creation function

Synopsis

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

Widget XmCreateForm (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateForm creates an instance of a Form widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Form and its associated resources, see **XmForm(3X)**.

Return Value

Returns the Form widget ID.

Related Information

XmForm(3X).

XmCreateFormDialog—A Form FormDialog convenience creation function

Synopsis

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

Widget XmCreateFormDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateFormDialog is a convenience creation function that creates a DialogShell and an unmanaged Form child of the DialogShell. A FormDialog is used for interactions not supported by the standard dialog set. This function does not automatically create any labels, buttons, or other dialog components. Such components should be added by the application after the FormDialog is created.

Use **XtManageChild** to pop up the FormDialog (passing the Form as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Form and its associated resources, see **XmForm(3X)**.

Return Value

Returns the Form widget ID.

Related Information

XmForm(3X).

XmCreateFrame—The Frame widget creation function

Synopsis

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

Widget XmCreateFrame (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateFrame creates an instance of a Frame widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Frame and its associated resources, see **XmFrame(3X)**.

Return Value

Returns the Frame widget ID.

Related Information

XmFrame(3X).

XmCreateInformationDialog—The MessageBox InformationDialog convenience creation function

Synopsis

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

Widget XmCreateInformationDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateInformationDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. An InformationDialog gives the user information, such as the status of an action. It includes a symbol, a message, and three buttons. The default symbol is **i**. The default button labels are **OK**, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the InformationDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCreateLabel—The Label widget creation function

Synopsis

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

Widget XmCreateLabel (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateLabel creates an instance of a Label widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Label and its associated resources, see **XmLabel(3X)**.

Return Value

Returns the Label widget ID.

Related Information

XmLabel(3X).

XmCreateLabelGadget—The LabelGadget creation function

Synopsis

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

Widget XmCreateLabelGadget (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateLabelGadget creates an instance of a LabelGadget widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of LabelGadget and its associated resources, see **XmLabelGadget(3X)**.

Return Value

Returns the LabelGadget widget ID.

Related Information

XmLabelGadget(3X).

XmCreateList—The List widget creation function

Synopsis

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

Widget XmCreateList (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateList creates an instance of a List widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns the List widget ID.

Related Information

XmList(3X).

XmCreateMainWindow—The MainWindow widget creation function

Synopsis

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

Widget XmCreateMainWindow (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateMainWindow creates an instance of a MainWindow widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MainWindow and its associated resources, see **XmMainWindow(3X)**.

Return Value

Returns the MainWindow widget ID.

Related Information

XmMainWindow(3X).

XmCreateMenuBar—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateMenuBar (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateMenuBar creates an instance of a RowColumn widget of type **XmMENU_BAR** and returns the associated widget ID. It is provided as a convenience function for creating RowColumn widgets configured to operate as a MenuBar and is not implemented as a separate widget class.

The MenuBar widget is generally used for building a Pulldown menu system. Typically, a MenuBar is created and placed along the top of the application window, and several CascadeButtons are inserted as the children. Each of the CascadeButtons has a Pulldown MenuPane associated with it. These Pulldown MenuPanes must have been created as children of the MenuBar. The user interacts with the MenuBar by using either the mouse or the keyboard.

The MenuBar displays a 3-D shadow along its border. The application controls the shadow attributes using the visual-related resources supported by **XmManager**.

The MenuBar widget is homogeneous in that it accepts only children that are a subclass of **XmCascadeButton** or **XmCascadeButtonGadget**. Attempting to insert a child of a different class results in a warning message.

If the MenuBar does not have enough room to fit all of its subwidgets on a single line, the MenuBar attempts to wrap the remaining entries onto additional lines if allowed by the geometry manager of the parent widget.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCascadeButton(3X), **XmCascadeButtonGadget(3X)**, **XmCreatePulldownMenu(3X)**, **XmCreateSimpleMenuBar(3X)**, **XmManager(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimpleMenuBar(3X)**.

XmCreateMenuShell—The MenuShell widget creation function

Synopsis

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

Widget XmCreateMenuShell (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateMenuShell creates an instance of a MenuShell widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MenuShell and its associated resources, see **XmMenuShell(3X)**.

Return Value

Returns the MenuShell widget ID.

Related Information

XmMenuShell(3X).

XmCreateMessageBox—The MessageBox widget creation function

Synopsis

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

Widget XmCreateMessageBox (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateMessageBox creates an unmanaged MessageBox. A MessageBox is used for common interaction tasks, which include giving information, asking questions, and reporting errors. It includes an optional symbol, a message, and three buttons.

By default, there is no symbol. The default button labels are **OK**, **Cancel**, and **Help**.

If the parent of the MessageBox is a DialogShell, use **XtManageChild** to pop up the MessageBox (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCreateMessageDialog—The MessageBox MessageDialog convenience creation function

Synopsis

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

Widget XmCreateMessageDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateMessageDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A MessageDialog is used for common interaction tasks, which include giving information, asking questions, and reporting errors. It includes a symbol, a message, and three buttons. By default, there is no symbol. The default button labels are **OK**, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the MessageDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCreateOptionMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateOptionMenu (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateOptionMenu creates an instance of a RowColumn widget of type **XmMENU_OPTION** and returns the associated widget ID.

It is provided as a convenience function for creating a RowColumn widget configured to operate as an OptionMenu and is not implemented as a separate widget class.

The OptionMenu widget is a specialized RowColumn manager composed of a label, a selection area, and a single Pulldown MenuPane. When an application creates an OptionMenu widget, it supplies the label string and the Pulldown MenuPane. In order for the operation to be successful, there must be a valid **XmNsubMenuId** resource set when this function is called. When the OptionMenu is created, the Pulldown MenuPane must have been created as a child of the OptionMenu's parent and must be specified. The LabelGadget and the selection area (a CascadeButtonGadget) are created by the OptionMenu.

The OptionMenu's Pulldown MenuPane must not contain any ToggleButtons or ToggleButtonGadgets. The results of including CascadeButtons or CascadeButtonGadgets in the OptionMenu's Pulldown MenuPane are undefined.

An OptionMenu is laid out with the label displayed on one side of the widget and the selection area on the other side when **XmNorientation** is **XmHORIZONTAL.** If the value is **XmVERTICAL**, the label is above the selection area. The selection area has a dual purpose; it displays the label of the last item selected from the associated Pulldown MenuPane, and it provides the means for posting the Pulldown MenuPane.

The OptionMenu typically does not display any 3-D visuals around itself or the internal LabelGadget. By default, the internal CascadeButtonGadget has a visible 3-D shadow. The application may change this by getting the CascadeButtonGadget ID using **XmOptionButtonGadget**, and then calling **XtSetValues** using the standard visual-related resources.

The Pulldown MenuPane is posted when the mouse pointer is moved over the selection area and a mouse button that is defined by OptionMenu's RowColumn parent is pressed. The Pulldown MenuPane is posted and positioned so that the last

selected item is directly over the selection area. The mouse is then used to arm the desired menu item. When the mouse button is released, the armed menu item is selected and the label within the selection area is changed to match that of the selected item. By default, **BSelect** is used to interact with an OptionMenu. The default can be changed with the RowColumn resource **XmNmenuPost**.

The OptionMenu also operates with the keyboard interface mechanism. If the application has established a mnemonic with the OptionMenu, pressing **<Alt>** with the mnemonic causes the Pulldown MenuPane to be posted with traversal enabled. The standard traversal keys can then be used to move within the MenuPane. Pressing **<Return>** or typing a mnemonic or accelerator for one of the menu items selects that item.

An application may use the **XmNmenuHistory** resource to indicate which item in the Pulldown MenuPane should be treated as the current choice and have its label displayed in the selection area. By default, the first item in the Pulldown MenuPane is used.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. These widgets (or gadgets) and the associated OptionMenu areas are

Option Menu Label Gadget	**OptionLabel**
Option Menu Cascade Button	**OptionButton**

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCascadeButtonGadget(3X), **XmCreatePulldownMenu(3X)**, **XmCreateSimpleOptionMenu(3X)**, **XmLabelGadget(3X)**, **XmOptionButtonGadget(3X)**, **XmOptionLabelGadget(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimpleOptionMenu(3X)**.

XmCreatePanedWindow—The PanedWindow widget creation function

Synopsis

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

Widget XmCreatePanedWindow (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreatePanedWindow creates an instance of a PanedWindow widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of PanedWindow and its associated resources, see **XmPanedWindow(3X)**.

Return Value

Returns the PanedWindow widget ID.

Related Information

XmPanedWindow(3X).

XmCreatePopupMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreatePopupMenu (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreatePopupMenu creates an instance of a RowColumn widget of type **XmMENU_POPUP** and returns the associated widget ID. When this function is used to create the Popup MenuPane, a MenuShell widget is automatically created as the parent of the MenuPane. The parent of the MenuShell widget is the widget indicated by the *parent* parameter.

XmCreatePopupMenu is provided as a convenience function for creating RowColumn widgets configured to operate as Popup MenuPanes and is not implemented as a separate widget class.

The PopupMenu is used as the first MenuPane within a PopupMenu system; all other MenuPanes are of the Pulldown type. A Popup MenuPane displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the MenuPane.

The Popup MenuPane must be created as the child of a MenuShell widget in order to function properly when it is incorporated into a menu. If the application uses this convenience function for creating a Popup MenuPane, the MenuShell is automatically created as the real parent of the MenuPane. If the application does not use this convenience function to create the RowColumn to function as a Popup MenuPane, it is the application's responsibility to create the MenuShell widget.

To access the PopupMenu, the application must first position the widget using the **XmMenuPosition** function and then manage it using **XtManageChild**.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

Popup MenuPanes support tear-off capabilities for tear-off menus through **XmRowColumn** resources. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateSimplePopupMenu(3X), **XmMenuPosition(3X)**, **XmMenuShell(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimplePopupMenu(3X)**.

XmCreatePromptDialog—The SelectionBox PromptDialog convenience creation function

Synopsis

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

Widget XmCreatePromptDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreatePromptDialog is a convenience creation function that creates a DialogShell and an unmanaged SelectionBox child of the DialogShell. A PromptDialog prompts the user for text input. It includes a message, a text input region, and three managed buttons. The default button labels are **OK**, **Cancel**, and **Help**. An additional button, with **Apply** as the default label, is created unmanaged; it may be explicitly managed if needed. One additional **WorkArea** child may be added to the SelectionBox after creation.

XmCreatePromptDialog forces the value of the SelectionBox resource **XmNdialogType** to **XmDIALOG_PROMPT**.

Use **XtManageChild** to pop up the PromptDialog (passing the SelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox(3X)**.

Return Value

Returns the SelectionBox widget ID.

Related Information

XmSelectionBox(3X).

XmCreatePulldownMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreatePulldownMenu (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreatePulldownMenu creates an instance of a RowColumn widget of type **XmMENU_PULLDOWN** and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

Specifies the number of attribute/value pairs in the argument list (*arglist*) When using this function to create the Pulldown MenuPane, a MenuShell widget is automatically created as the parent of the MenuPane. If the widget specified by the *parent* parameter is a Popup or a Pulldown MenuPane, the MenuShell widget is created as a child of the *parent*'s MenuShell; otherwise, it is created as a child of the specified *parent* widget.

XmCreatePulldownMenu is provided as a convenience function for creating RowColumn widgets configured to operate as Pulldown MenuPanes and is not implemented as a separate widget class.

A Pulldown MenuPane displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the MenuPane.

A Pulldown MenuPane is used with submenus that are to be attached to a CascadeButton or a CascadeButtonGadget. This is the case for all MenuPanes that are part of a PulldownMenu system (a MenuBar), the MenuPane associated with an OptionMenu, and any MenuPanes that cascade from a Popup MenuPane. Pulldown MenuPanes that are to be associated with an OptionMenu must be created before the OptionMenu is created.

The Pulldown MenuPane must be attached to a CascadeButton or CascadeButtonGadget that resides in a MenuBar, a Popup MenuPane, a Pulldown MenuPane, or an OptionMenu. It is attached with the button resource **XmNsubMenuId**.

A MenuShell widget is required between the Pulldown MenuPane and its parent. If the application uses this convenience function for creating a Pulldown MenuPane, the MenuShell is automatically created as the real parent of the MenuPane; otherwise, it is the application's responsibility to create the MenuShell widget.

To function correctly when incorporated into a menu, the Pulldown MenuPane's hierarchy must be considered. This hierarchy depends on the type of menu system that is being built, as follows:

- If the Pulldown MenuPane is to be pulled down from a MenuBar, its *parent* must be the MenuBar.
- If the Pulldown MenuPane is to be pulled down from a Popup or another Pulldown MenuPane, its *parent* must be that Popup or Pulldown MenuPane.
- If the Pulldown MenuPane is to be pulled down from an OptionMenu, its *parent* must be the same as the OptionMenu parent.

PullDown MenuPanes support tear-off capabilities for tear-off menus through **XmRowColumn** resources. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCascadeButton(3X), **XmCascadeButtonGadget(3X)**, **XmCreateOptionMenu(3X)**, **XmCreatePopupMenu(3X)**, **XmCreateSimplePulldownMenu(3X)**, **XmMenuShell(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimplePulldownMenu(3X)**.

XmCreatePushButton—The PushButton widget creation function

Synopsis

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

Widget XmCreatePushButton (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreatePushButton creates an instance of a PushButton widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of PushButton and its associated resources, see **XmPushButton(3X)**.

Return Value

Returns the PushButton widget ID.

Related Information

XmPushButton(3X).

XmCreatePushButtonGadget—The PushButtonGadget creation function

Synopsis

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

Widget XmCreatePushButtonGadget (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreatePushButtonGadget creates an instance of a PushButtonGadget widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of PushButtonGadget and its associated resources, see **XmPushButtonGadget(3X)**.

Return Value

Returns the PushButtonGadget widget ID.

Related Information

XmPushButtonGadget(3X).

XmCreateQuestionDialog—The MessageBox QuestionDialog convenience creation function

Synopsis

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

Widget XmCreateQuestionDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateQuestionDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A QuestionDialog is used to get the answer to a question from the user. It includes a symbol, a message, and three buttons. The default symbol is a question mark. The default button labels are **OK**, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the QuestionDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCreateRadioBox—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateRadioBox (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateRadioBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID. Typically, this is a composite widget that contains multiple ToggleButtonGadgets. The RadioBox arbitrates and ensures that at most one ToggleButtonGadget is on at any time.

Unless the application supplies other values in the *arglist*, this function provides initial values for several RowColumn resources. It initializes **XmNpacking** to **XmPACK_COLUMN**, **XmNradioBehavior** to True, **XmNisHomogeneous** to True, and **XmNentryClass** to **XmToggleButtonGadgetClass**.

In a RadioBox, the ToggleButton or ToggleButtonGadget resource **XmNindicatorType** defaults to **XmONE_OF_MANY**, and the ToggleButton or ToggleButtonGadget resource**XmNvisibleWhenOff** defaults to True.

This routine is provided as a convenience function for creating RowColumn widgets.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateRowColumn(3X), **XmCreateSimpleCheckBox(3X)**, **XmCreateSimpleRadioBox(3X)**, **XmCreateWorkArea(3X)**, **XmRowColumn(3X)**, **XmVaCreateSimpleCheckBox(3X)**, and **XmVaCreateSimpleRadioBox(3X)**.

XmCreateRowColumn—The RowColumn widget creation function

Synopsis

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

Widget XmCreateRowColumn (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateRowColumn creates an instance of a RowColumn widget and returns the associated widget ID. If **XmNrowColumnType** is not specified, then it is created with **XmWORK_AREA**, which is the default.

If this function is used to create a Popup Menu of type **XmMENU_POPUP** or a Pulldown Menu of type **XmMENU_PULLDOWN**, a MenuShell widget is not automatically created as the parent of the MenuPane. The application must first create the MenuShell by using either **XmCreateMenuShell** or the standard toolkit create function.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateMenuBar(3X), **XmCreateMenuShell(3X)**, **XmCreateOptionMenu(3X)**, **XmCreatePopupMenu(3X)**, **XmCreatePulldownMenu(3X)**, **XmCreateRadioBox(3X)**, **XmCreateSimpleCheckBox(3X)**, **XmCreateSimpleMenuBar(3X)**, **XmCreateSimpleOptionMenu(3X)**, **XmCreateSimplePopupMenu(3X)**, **XmCreateSimplePulldownMenu(3X)**, **XmCreateSimpleRadioBox(3X)**, **XmCreateWorkArea(3X)**, **XmRowColumn(3X)**, **XmVaCreateSimpleCheckBox(3X)**, **XmVaCreateSimpleMenuBar(3X)**, **XmVaCreateSimpleOptionMenu(3X)**, **XmVaCreateSimplePopupMenu(3X)**, **XmVaCreateSimplePulldownMenu(3X)**, and **XmVaCreateSimpleRadioBox(3X)**.

XmCreateScale—The Scale widget creation function

Synopsis

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

Widget XmCreateScale (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateScale creates an instance of a Scale widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Scale and its associated resources, see **XmScale(3X)**.

Return Value

Returns the Scale widget ID.

Related Information

XmScale(3X).

XmCreateScrollBar—The ScrollBar widget creation function

Synopsis

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

Widget XmCreateScrollBar (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateScrollBar creates an instance of a ScrollBar widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ScrollBar and its associated resources, see **XmScrollBar(3X)**.

Return Value

Returns the ScrollBar widget ID.

Related Information

XmScrollBar(3X).

XmCreateScrolledList—The List ScrolledList convenience creation function

Synopsis

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

Widget XmCreateScrolledList (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateScrolledList creates an instance of a List widget that is contained within a ScrolledWindow. All ScrolledWindow subarea widgets are automatically created by this function. The ID returned by this function is that of the List widget. Use this ID for all normal List operations, as well as those that are relevant for the ScrolledList widget.

All arguments to either the List or the ScrolledWindow widget can be specified at creation time using this function. Changes to initial position and size are sent only to the ScrolledWindow widget. Other resources are sent to the List or the ScrolledWindow widget as appropriate.

This function forces the following initial values for ScrolledWindow resources:

- **XmNscrollingPolicy** is set to **XmAPPLICATION_DEFINED**.
- **XmNvisualPolicy** is set to **XmVARIABLE**.
- **XmNscrollBarDisplayPolicy** is set to **XmSTATIC**. (No initial value is forced for the List's **XmNscrollBarDisplayPolicy**.)
- **XmNshadowThickness** is set to 0 (zero).

To obtain the ID of the ScrolledWindow widget associated with the ScrolledList, use the Xt Intrinsics **XtParent** function. The name of the ScrolledWindow created by this function is formed by concatenating **SW** onto the end of the *name* specified in the parameter list.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns the List widget ID.

Related Information

XmList(3X) and **XmScrolledWindow(3X)**.

XmCreateScrolledText—The Text ScrolledText convenience creation function

Synopsis

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

Widget XmCreateScrolledText (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateScrolledText creates an instance of a Text widget that is contained within a ScrolledWindow. All ScrolledWindow subarea widgets are automatically created by this function. The ID returned by this function is that of the Text widget. Use this ID for all normal Text operations, as well as those that are relevant for the ScrolledText widget.

The Text widget defaults to single-line text edit; therefore, no ScrollBars are displayed. The Text resource **XmNeditMode** must be set to **XmMULTI_LINE_EDIT** to display the ScrollBars. The results of placing a Text widget inside a ScrolledWindow when the Text's **XmNeditMode** is **XmSINGLE_LINE_EDIT** are undefined.

All arguments to either the Text or the ScrolledWindow widget can be specified at creation time with this function. Changes to initial position and size are sent only to the ScrolledWindow widget. Other resources are sent to the Text or the ScrolledWindow widget as appropriate.

This function forces the following initial values for ScrolledWindow resources:

- **XmNscrollingPolicy** is set to **XmAPPLICATION_DEFINED**.
- **XmNvisualPolicy** is set to **XmVARIABLE**.
- **XmNscrollBarDisplayPolicy** is set to **XmSTATIC**.
- **XmNshadowThickness** is set to 0 (zero).

To obtain the ID of the ScrolledWindow widget associated with the ScrolledText, use the Xt Intrinsics **XtParent** function. The name of the ScrolledWindow created by this function is formed by concatenating the letters **SW** onto the end of the *name* specified in the parameter list.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns the Text widget ID.

Related Information

XmScrolledWindow(3X) and **XmText(3X)**.

XmCreateScrolledWindow—The ScrolledWindow widget creation function

Synopsis

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

Widget XmCreateScrolledWindow (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateScrolledWindow creates an instance of a ScrolledWindow widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ScrolledWindow and its associated resources, see **XmScrolledWindow(3X)**.

Return Value

Returns the ScrolledWindow widget ID.

Related Information

XmScrolledWindow(3X).

XmCreateSelectionBox—The SelectionBox widget creation function

Synopsis

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

Widget XmCreateSelectionBox (parent, name, arglist, argcount)
    Widget      parent;
    String      name;
    ArgList     arglist;
    Cardinal    argcount;
```

Description

XmCreateSelectionBox creates an unmanaged SelectionBox. A SelectionBox is used to get a selection from a list of alternatives from the user and includes the following:

- A scrolling list of alternatives
- An editable text field for the selected alternative
- Labels for the list and text field
- Three or four buttons

The default button labels are **OK**, **Cancel**, and **Help**. By default, an **Apply** button is also created. If the parent of the SelectionBox is a DialogShell, it is managed; otherwise it is unmanaged. Additional work area children may be added to the SelectionBox after creation.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox(3X)**.

Return Value

Returns the SelectionBox widget ID.

Related Information

XmSelectionBox(3X).

XmCreateSelectionDialog—The SelectionBox SelectionDialog convenience creation function

Synopsis

#include <Xm/SelectioB.h>

Widget XmCreateSelectionDialog (*parent, name, arglist, argcount*)

Widget	*parent*;
String	*name*;
ArgList	*arglist*;
Cardinal	*argcount*;

Description

XmCreateSelectionDialog is a convenience creation function that creates a DialogShell and an unmanaged SelectionBox child of the DialogShell. A SelectionDialog offers the user a choice from a list of alternatives and gets a selection. It includes the following:

- A scrolling list of alternatives
- An editable text field for the selected alternative
- Labels for the text field
- Four buttons

The default button labels are **OK**, **Cancel**, **Apply**, and **Help**. One additional **WorkArea** child may be added to the SelectionBox after creation.

XmCreateSelectionDialog forces the value of the SelectionBox resource **XmNdialogType** to **XmDIALOG_SELECTION**.

Use **XtManageChild** to pop up the SelectionDialog (passing the SelectionBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox(3X)**.

Return Value

Returns the SelectionBox widget ID.

Related Information

XmSelectionBox(3X).

XmCreateSeparator—The Separator widget creation function

Synopsis

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

Widget XmCreateSeparator (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSeparator creates an instance of a Separator widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Separator and its associated resources, see **XmSeparator(3X)**.

Return Value

Returns the Separator widget ID.

Related Information

XmSeparator(3X).

XmCreateSeparatorGadget—The SeparatorGadget creation function

Synopsis

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

Widget XmCreateSeparatorGadget (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSeparatorGadget creates an instance of a SeparatorGadget widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of SeparatorGadget and its associated resources, see **XmSeparatorGadget(3X)**.

Return Value

Returns the SeparatorGadget widget ID.

Related Information

XmSeparatorGadget(3X).

XmCreateSimpleCheckBox—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateSimpleCheckBox (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSimpleCheckBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID.

This routine creates a CheckBox and its ToggleButtonGadget children. A CheckBox is similar to a RadioBox, except that more than one button can be selected at a time. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. Buttons are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button type allowed in the **XmNbuttonType** resource is **XmCHECKBUTTON**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3X), **XmCreateRowColumn(3X)**, **XmCreateSimpleRadioBox(3X)**, **XmRowColumn(3X)**, **XmVaCreateSimpleCheckBox(3X)**, and **XmVaCreateSimpleRadioBox(3X)**.

XmCreateSimpleMenuBar—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateSimpleMenuBar (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSimpleMenuBar creates an instance of a RowColumn widget of type **XmMENU_BAR** and returns the associated widget ID.

This routine creates a MenuBar and its CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. Buttons are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button type allowed in the **XmNbuttonType** resource is **XmCASCADEBUTTON**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateMenuBar(3X), **XmCreateRowColumn(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimpleMenuBar(3X)**.

XmCreateSimpleOptionMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateSimpleOptionMenu (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSimpleOptionMenu creates an instance of a RowColumn widget of type **XmMENU_OPTION** and returns the associated widget ID.

This routine creates an OptionMenu and its submenu containing PushButtonGadget or CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of separators in the menu. Buttons and separators are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. These widgets (or gadgets) and the associated OptionMenu areas are

Option Menu Label Gadget	**OptionLabel**
Option Menu Cascade Button	**OptionButton**

A number of resources exist specifically for use with this and other simple menu creation routines. The only button types allowed in the **XmNbuttonType** resource are **XmPUSHBUTTON**, **XmCASCADEBUTTON**, **XmSEPARATOR**, and **XmDOUBLE_SEPARATOR**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateOptionMenu(3X), **XmCreateRowColumn(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimpleOptionMenu(3X)**.

XmCreateSimplePopupMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateSimplePopupMenu (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSimplePopupMenu creates an instance of a RowColumn widget of type **XmMENU_POPUP** and returns the associated widget ID.

This routine creates a Popup MenuPane and its button children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of separators in the menu. The name of each title is **label_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of titles in the menu. Buttons, separators, and titles are named and created in the order in which they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the widget ID of the parent of the MenuShell
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button types allowed in the **XmNbuttonType** resource are **XmCASCADEBUTTON**, **XmPUSHBUTTON**, **XmRADIOBUTTON**, **XmCHECKBUTTON**, **XmTITLE**, **XmSEPARATOR**, and **XmDOUBLE_SEPARATOR**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreatePopupMenu(3X), **XmCreateRowColumn(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimplePopupMenu(3X)**.

XmCreateSimplePulldownMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateSimplePulldownMenu (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSimplePulldownMenu creates an instance of a RowColumn widget of type **XmMENU_PULLDOWN** and returns the associated widget ID.

This routine creates a Pulldown MenuPane and its button children. The name of each button is **button_*n***, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. The name of each separator is **separator_*n***, where *n* is an integer from 0 (zero) to 1 minus the number of separators in the menu. The name of each title is **label_*n***, where *n* is an integer from 0 (zero) to 1 minus the number of titles in the menu. Buttons, separators, and titles are named and created in the order they are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent	Specifies the widget ID of the parent of the MenuShell
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button types allowed in the **XmNbuttonType** resource are **XmCASCADEBUTTON**, **XmPUSHBUTTON**, **XmRADIOBUTTON**, **XmCHECKBUTTON**, **XmTITLE**, **XmSEPARATOR**, and **XmDOUBLE_SEPARATOR**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreatePulldownMenu(3X), **XmCreateRowColumn(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimplePulldownMenu(3X)**.

XmCreateSimpleRadioBox—A RowColumn widget convenience creation function

Synopsis

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

Widget XmCreateSimpleRadioBox (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateSimpleRadioBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID.

This routine creates a RadioBox and its ToggleButtonGadget children. The name of each button is **button_*n***, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. Buttons are named and created in the orderthey are specified in the RowColumn simple menu creation resources supplied in the argument list.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

A number of resources exist specifically for use with this and other simple menu creation routines. The only button type allowed in the **XmNbuttonType** resource is **XmRADIOBUTTON**. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3X), **XmCreateRowColumn(3X)**, **XmCreateSimpleCheckBox(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimpleRadioBox(3X)**.

XmCreateTemplateDialog—A MessageBox TemplateDialog convenience creation function

Synopsis

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

Widget XmCreateTemplateDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateTemplateDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. The MessageBox widget's **XmNdialogType** resource is set to **XmDIALOG_TEMPLATE**. By default, the TemplateDialog widget contains only the separator child. You can build a customized dialog by adding children to the TemplateDialog.

You can create the standard MessageBox pushbuttons, **Cancel**, **Help**, and **OK**, by specifying the associated callback and label string resources. Setting **XmNsymbolPixmap** or **XmNmessageString** creates a symbol or message label.

Use **XtManageChild** to pop up the TemplateDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCreateText—The Text widget creation function

Synopsis

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

Widget XmCreateText (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateText creates an instance of a Text widget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns the Text widget ID.

Related Information

XmText(3X).

XmCreateTextField—The TextField widget creation function

Synopsis

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

Widget XmCreateTextField (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateTextField creates an instance of a TextField widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns the TextField widget ID.

Related Information

XmTextField(3X).

XmCreateToggleButton—The ToggleButton widget creation function

Synopsis

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

Widget XmCreateToggleButton (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateToggleButton creates an instance of a ToggleButton widget and returns the associated widget ID.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ToggleButton and its associated resources, see **XmToggleButton(3X)**.

Return Value

Returns the ToggleButton widget ID.

Related Information

XmToggleButton(3X).

XmCreateToggleButtonGadget—The ToggleButtonGadget creation function

Synopsis

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

Widget XmCreateToggleButtonGadget (parent, name, arglist, argcount)
    Widget      parent;
    String      name;
    ArgList     arglist;
    Cardinal    argcount;
```

Description

XmCreateToggleButtonGadget creates an instance of a ToggleButtonGadget and returns the associated widget ID.

parent	Specifies the parent widget ID
name	Specifies the name of the created widget
arglist	Specifies the argument list
argcount	Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of ToggleButtonGadget and its associated resources, see **XmToggleButtonGadget(3X)**.

Return Value

Returns the ToggleButtonGadget widget ID.

Related Information

XmToggleButtonGadget(3X).

XmCreateWarningDialog—The MessageBox WarningDialog convenience creation function

Synopsis

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

Widget XmCreateWarningDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateWarningDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A WarningDialog warns users of action consequences and gives them a choice of resolutions. It includes a symbol, a message, and three buttons. The default symbol is an exclamation point. The default button labels are **OK**, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the WarningDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCreateWorkArea—A function that creates a RowColumn work area

Synopsis

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

Widget XmCreateWorkArea (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateWorkArea creates an instance of a RowColumn widget and returns the associated widget ID. The widget is created with **XmNrowColumnType** set to **XmWORK_AREA**.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3X), **XmCreateSimpleCheckBox(3X)**, **XmCreateSimpleRadioBox(3X)**, **XmRowColumn(3X)**, **XmVaCreateSimpleCheckBox(3X)**, and **XmVaCreateSimpleRadioBox(3X)**.

XmCreateWorkingDialog—The MessageBox WorkingDialog convenience creation function

Synopsis

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

Widget XmCreateWorkingDialog (parent, name, arglist, argcount)
        Widget      parent;
        String      name;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmCreateWorkingDialog is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A WorkingDialog informs users that there is a time-consuming operation in progress and allows them to cancel the operation. It includes a symbol, a message, and three buttons. The default symbol is an hourglass. The default button labels are **OK**, **Cancel**, and **Help**.

Use **XtManageChild** to pop up the WorkingDialog (passing the MessageBox as the widget parameter); use **XtUnmanageChild** to pop it down.

parent Specifies the parent widget ID

name Specifies the name of the created widget

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the MessageBox widget ID.

Related Information

XmMessageBox(3X).

XmCvtCTToXmString—A compound string function that converts compound text to a compound string

Synopsis

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

XmString XmCvtCTToXmString (text)
        char            * text;
```

Description

XmCvtCTToXmString converts a (**char** *) string in compound text format to a compound string. The application must call **XtAppInitialize** before calling this function. Conversion of compound text to compound strings is implementation dependent.

text Specifies a string in compound text format to be converted to a compound string.

Return Value

Returns a compound string derived from the compound text. The compound text is assumed to be NULL-terminated; NULLs within the compound text are handled correctly. The handling of HORIZONTAL TABULATION (HT) control characters within the compound text is undefined. The compound text format is described in the X Consortium Standard *Compound Text Encoding*.

Related Information

XmCvtXmStringToCT(3X).

XmCvtStringToUnitType—A function that converts a string to a unit-type value

Synopsis

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

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

Description

XmCvtStringToUnitType converts a string to a unit type. Refer to the reference pages for **XmGadget**, **XmManager**, or **XmPrimitive** for a description of the valid unit types. Use of this function as a resource converter is obsolete. It has been replaced by a new resource converter that uses the RepType facility.

args Specifies a list of additional **XrmValue** arguments to the converter if additional context is needed to perform the conversion. For example, the string-to-font converter needs the widget's screen and the string-to-pixel converter needs the widget's screen and color map. This argument is often NULL.

num_args Specifies the number of additional **XrmValue** arguments. This argument is often zero.

from_val Specifies the value to convert

to_val Specifies the descriptor to use to return the converted value

Related Information

XmGadget(3X), **XmManager(3X)**, and **XmPrimitive(3X)**.

XmCvtXmStringToCT—A compound string function that converts a compound string to compound text

Synopsis

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

char * XmCvtXmStringToCT (string)
        XmString        string;
```

Description

XmCvtXmStringToCT converts a compound string to a (**char** *) string in compound text format. The application must call **XtAppInitialize** before calling this function. The converter uses the font list tag associated with a given compound string segment to select a compound text format for that segment. A registry defines a mapping between font list tags and compound text encoding formats. The converter uses the following algorithm for each compound string segment:

1. If the compound string segment tag is mapped to **XmFONTLIST_DEFAULT_TAG** in the registry, the converter passes the text of the compound string segment to **XmbTextListToTextProperty** with an encoding style of **XCompoundTextStyle** and uses the resulting compound text for that segment.
2. If the compound string segment tag is mapped to an MIT registered charset in the registry, the converter creates the compound text for that segment using the charset (from the registry) and the text of the compound string segment as defined in the X Consortium Standard *Compound Text Encoding*.
3. If the compound string segment tag is mapped to a charset in the registry that is neither **XmFONTLIST_DEFAULT_TAG** nor an MIT registered charset, the converter creates the compound text for that segment using the charset (from the registry) and the text of the compound string segment as an "extended segment" with a variable number of octets per character.
4. If the compound string segment tag is not mapped in the registry, the result is implementation dependent.

string Specifies a compound string to be converted to compound text.

Return Value

Returns a (**char** *) string in compound text format. This format is described in the X Consortium Standard *Compound Text Encoding*.

Related Information

XmCvtCTToXmString(3X), **XmFontList(3X)**, **XmMapSegmentEncoding(3X)**, **XmRegisterSegmentEncoding(3X)**, and **XmString**.

XmDeactivateProtocol—A VendorShell function that deactivates a protocol without removing it

Synopsis

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

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

void XmDeactivateWMProtocol (shell, protocol)
        Widget      shell;
        Atom        protocol;
```

Description

XmDeactivateProtocol deactivates a protocol without removing it. It updates the handlers and the *property* if the *shell* is realized. It is sometimes useful to allow a protocol's state information (callback lists, and so on) to persist, even though the client may choose to temporarily resign from the interaction. The main use of this capability is to gray/ungray **f.send_msg** entries in the MWM system menu. To support this capability, *protocol* is allowd to be in one of two states: active or inactive. If *protocol* is active and *shell* is realized, *property* contains the *protocol* **Atom**. If *protocol* is inactive, **Atom** is not present in the *property*.

XmDeactivateWMProtocol is a convenience interface. It calls **XmDeactivateProtocol** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

property Specifies the protocol property

protocol Specifies the protocol atom (or an int type cast to **Atom**)

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

mwm(1X), **VendorShell(3X)**, **XmDeactivateWMProtocol(3X)**, and **XmInternAtom(3X)**.

XmDeactivateWMProtocol—A VendorShell convenience interface that deactivates a protocol without removing it

Synopsis

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

void XmDeactivateWMProtocol (shell, protocol)
        Widget      shell;
        Atom        protocol;
```

Description

XmDeactivateWMProtocol is a convenience interface. It calls **XmDeactivateProtocol** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

protocol Specifies the protocol atom (or an int type cast to **Atom**)

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmDeactivateProtocol(3X)**, and **XmInternAtom(3X)**.

XmDestroyPixmap—A pixmap caching function that removes a pixmap from the pixmap cache

Synopsis

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

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

Description

XmDestroyPixmap removes pixmaps that are no longer used. Pixmaps are completely freed only when there is no further reference to them.

screen Specifies the display screen for which the pixmap was requested

pixmap Specifies the pixmap to be destroyed

Return Value

Returns True when successful; returns False if there is no matching screen and pixmap in the pixmap cache.

Related Information

XmInstallImage(3X), **XmUninstallImage(3X)**, and **XmGetPixmap(3X)**.

XmDialogShell—The DialogShell widget class

Synopsis

#include <Xm/DialogS.h>

Description

Modal and modeless dialogs use DialogShell as the Shell parent. DialogShell widgets cannot be iconified. Instead, all secondary DialogShell widgets associated with an ApplicationShell widget are iconified and de-iconified as a group with the primary widget.

The client indirectly manipulates DialogShell through the convenience interfaces during creation, and it can directly manipulate its BulletinBoard-derived child. Much of the functionality of DialogShell assumes that its child is a BulletinBoard subclass, although it can potentially stand alone.

Setting **XmNheight**, **XmNwidth**, or **XmNborderWidth** for either a DialogShell or its managed child usually sets that resource to the same value in both the parent and the child. When an off-the-spot input method exists, the height and width of the shell may be greater than those of the managed child in order to accommodate the input method. In this case setting **XmNheight** or **XmNwidth** for the shell does not necessarily set that resource to the same value in the managed child, and setting **XmNheight** or **XmNwidth** for the child does not necessarily set that resource to the same value in the shell.

For the managed child of a DialogShell, regardless of the value of the shell's **XmNallowShellResize** resource, setting **XmNx** or **XmNy** sets the corresponding resource of the parent but does not change the child's position relative to the parent. The **XtGetValues** resource for the child's **XmNx** or **XmNy** yields the value of the corresponding resource in the parent. The x and y-coordinates of the child's upper left outside corner relative to the parent's upper left inside corner are both 0 (zero) minus the value of **XmNborderWidth**.

Note that the *Inter-Client Communication Conventions Manual* (ICCCM) allows a window manager to change or control the border width of a reparented top-level window.

Classes

DialogShell inherits behavior and resources from the **Core**, **Composite**, **Shell**, **WMShell**, **VendorShell**, and **TransientShell** classes.

The class pointer is **xmDialogShellWidgetClass**.

The class name is **XmDialogShell**.

New Resources

DialogShell defines no new resources but overrides the **XmNdeleteResponse** resource in the **VendorShell** class.

Inherited Resources

DialogShell inherits behavior and resources from the superclasses described in the following tables, which define sets of widget resources used by the programmer to specify data.

For a complete description of each resource, refer to the reference page for that superclass. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

TransientShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNtransientFor XmCTransientFor	NULL Widget	CSG

VendorShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaudibleWarning XmCAudibleWarning	XmBELL unsigned char	CSG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNdefaultFontList XmCDefaultFontList	dynamic XmFontList	CG
XmNdeleteResponse XmCDeleteResponse	XmUNMAP unsigned char	CSG
XmNinputMethod XmCInputMethod	NULL String	CSG
XmNkeyboardFocusPolicy XmCKeyboardFocusPolicy	XmEXPLICIT unsigned char	CSG
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmwmDecorations XmCMwmDecorations	-1 int	CSG
XmNmwmFunctions XmCMwmFunctions	-1 int	CSG
XmNmwmInputMode XmCMwmInputMode	-1 int	CSG
XmNmwmMenu XmCMwmMenu	NULL String	CSG
XmNpreeditType XmCPreeditType	dynamic String	CSG
XmNshellUnitType XmCShellUnitType	XmPIXELS unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNuseAsyncGeometry XmCUseAsyncGeometry	False Boolean	CSG

WMShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbaseHeight XmCBaseHeight	XtUnspecifiedShellInt int	CSG
XmNbaseWidth XmCBaseWidth	XtUnspecifiedShellInt int	CSG
XmNheightInc XmCHeightInc	XtUnspecifiedShellInt int	CSG
XmNiconMask XmCIconMask	NULL Pixmap	CSG
XmNiconPixmap XmCIconPixmap	NULL Pixmap	CSG
XmNiconWindow XmCIconWindow	NULL Window	CSG
XmNiconX XmCIconX	-1 int	CSG
XmNiconY XmCIconY	-1 int	CSG
XmNinitialState XmCInitialState	NormalState int	CSG
XmNinput XmCInput	True Boolean	CSG
XmNmaxAspectX XmCMaxAspectX	XtUnspecifiedShellInt int	CSG
XmNmaxAspectY XmCMaxAspectY	XtUnspecifiedShellInt int	CSG
XmNmaxHeight XmCMaxHeight	XtUnspecifiedShellInt int	CSG
XmNmaxWidth XmCMaxWidth	XtUnspecifiedShellInt int	CSG
XmNminAspectX XmCMinAspectX	XtUnspecifiedShellInt int	CSG

Name Class	Default Type	Access
XmNminAspectY XmCMinAspectY	XtUnspecifiedShellInt int	CSG
XmNminHeight XmCMinHeight	XtUnspecifiedShellInt int	CSG
XmNminWidth XmCMinWidth	XtUnspecifiedShellInt int	CSG
XmNtitle XmCTitle	dynamic String	CSG
XmNtitleEncoding XmCTitleEncoding	dynamic Atom	CSG
XmNtransient XmCTransient	True Boolean	CSG
XmNwaitForWm XmCWaitForWm	True Boolean	CSG
XmNwidthInc XmCWidthInc	XtUnspecifiedShellInt int	CSG
XmNwindowGroup XmCWindowGroup	dynamic Window	CSG
XmNwinGravity XmCWinGravity	dynamic int	CSG
XmNwmTimeout XmCWmTimeout	5000 ms int	CSG

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	False Boolean	CG
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	False Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	True Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for XmDialogShell.

Related Information

Composite(3X), **Core(3X)**, **Shell(3X)**, **TransientShell(3X)**, **WMShell(3X)**, **VendorShell(3X)**, and **XmCreateDialogShell(3X)**.

XmDisplay—The Display widget class

Synopsis

#include <Xm/Display.h>

Description

The XmDisplay object is used by the Motif widgets to store information that is specific to a display. It also allows the toolkit to access certain information on widget hierarchies that would otherwise be unavailable. Each client has one XmDisplay object for each display it accesses.

An XmDisplay object is automatically created when the application creates the first shell on a display (usually accomplished by a call to **XtAppInitialize** or **XtAppCreateShell**). It is not necessary to create an XmDisplay object by any other means. An application can use the function **XmGetXmDisplay** to obtain the widget ID of the XmDisplay object for a given display.

An application cannot supply initial values for XmDisplay resources as arguments to a call to any function that creates widgets. The application or user can supply initial values in a resource file. After creating the first shell on the display, the application can use **XmGetXmDisplay** to obtain the widget ID of the XmDisplay object and then call **XtSetValues** to set the XmDisplay resources.

XmDisplay resources specify the drag protocol style for a client participating in drag and drop transactions. The two basic protocol types are preregister and dynamic. When a preregister protocol is used, the toolkit handles any communication between the initiator and receiver clients and displays the appropriate drag-over and drag-under visual effects. A client registers its drop sites in advance and this information is stored in a property for each top-level window. When the drag pointer enters a top-level window, the drop site information is read by the initiator. A dynamic protocol allows the source and destination clients to dynamically communicate drag and drop state information between each other, and to update their respective visuals accordingly. The toolkit provides drop site information as the pointer passes over any given drop site. In this mode, a receiver can supply a procedure to generate its own drag-under effects.

Classes

Display inherits behavior and resources from **Core**, **Composite**, **Shell**, **WMShell**, **VendorShell**, **TopLevelShell**, and **ApplicationShell** classes.

The class pointer is **xmDisplayClass**.

The class name is **XmDisplay**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using XtSetValues (S), retrieved by using XtGetValues (G), or is not applicable (N/A).

XmDisplay Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdefaultVirtualBindings DefaultVirtualBindings	dynamic String	CG
XmNdragInitiatorProtocolStyle XmCDragInitiatorProtocolStyle	XmDRAG_PREFER_RECEIVER unsigned char	CG
XmNdragReceiverProtocolStyle XmCDragReceiverProtocolStyle	XmDRAG_PREFER_PREREGISTER unsigned char	CG

XmNdefaultVirtualBindings

Specifies the default virtual bindings for the display. Following is an example of a specification for the **defaultVirtualBindings** resource in a resource file:

```
*defaultVirtualBindings: \
        osfBackSpace        :       <Key>BackSpace \n\
        osfInsert           :       <Key>InsertChar\n\
...
        osfDelete           :       <Key>DeleteChar
```

XmNdragInitiatorProtocolStyle

Specifies the drag and drop protocol requirements or preference when the client is an initiator. The possible values are

XmDRAG_PREREGISTER

As an initiator, this client does not use the dynamic protocol and can only arrange visual effects with receivers who provide preregistered information.

XmDRAG_DYNAMIC

As an initiator, this client does not make use of any preregistered drop site information made available by other clients, and can only arrange visual effects with receivers who use the dynamic protocol.

XmDRAG_NONE

Specifies that drag and drop is disabled for this client.

XmDRAG_DROP_ONLY

As an initiator, this client does not use either the preregistered drop site information or the dynamic protocol. It supports dragging, and any time the cursor is over a client that supports drag and drop, valid feedback is provided. There are no other visual effects.

XmDRAG_PREFER_DYNAMIC

As an initiator, this client can support both the preregister and dynamic protocols, but prefers to use dynamic protocols whenever possible in order to provide high-quality drag-under feedback.

XmDRAG_PREFER_PREREGISTER

As an initiator, this client can support both the preregister and dynamic protocols, but prefers to use the preregister protocol whenever possible in order to accommodate performance needs or to provide consistent drag-over feedback.

XmDRAG_PREFER_RECEIVER

Indicates that this client can support both preregister and dynamic protocols, but will defer to the preference of the receiver client. This value is valid only for the **XmNdragInitiatorProtocolStyle** resource, and is its default value.

XmNdragReceiverProtocolStyle

Specifies the drag and drop protocol requirements or preference when this client is a receiver. The values are

XmDRAG_PREREGISTER

As a receiver, this client preregisters drop site information and does not use the dynamic protocol. It can only arrange visual effects with initiators who make use of the preregistered information.

XmDRAG_DYNAMIC

As a receiver, this client uses the dynamic protocol and does not preregister drop site information. It can only arrange visual effects with initiators who use the dynamic protocol.

XmDRAG_NONE

Specifies that drag and drop is disabled for this client.

XmDRAG_DROP_ONLY

As a receiver, this client neither uses the dynamic protocol nor preregisters drop site information. It supports dropping, and when dragging over this client, valid feedback is always provided, but there are no other visual effects.

XmDRAG_PREFER_DYNAMIC

As a receiver, this client can support both the preregister and dynamic protocols, but prefers to use the dynamic protocol whenever possible in order to provide high-quality drag-under feedback.

XmDRAG_PREFER_PREREGISTER

As a receiver, this client can support both the preregister and dynamic protocols, but prefers to use the preregister protocol whenever possible in order to accommodate performance needs.

The actual protocol used between an initiator and a receiver is based on the protocol style of the receiver and initiator. The decision matrix is described in the following table.

Drag Initiator Protocol Style	Drag Receiver Protocol Style			
	Preregister	**Prefer Preregister**	**Prefer Dynamic**	**Dynamic**
Preregister	Preregister	Preregister	Preregister	Drop Only
Prefer Preregister	Preregister	Preregister	Preregister	Dynamic
Prefer Receiver	Preregister	Preregister	Dynamic	Dynamic
Prefer Dynamic	Preregister	Dynamic	Dynamic	Dynamic
Dynamic	Drop Only	Dynamic	Dynamic	Dynamic

The value **XmDRAG_NONE** does not appear in the matrix. When specified for either the initiator or receiver side, **XmDRAG_NONE** implies that drag and drop transactions are not supported. A value of **XmDRAG_DROP_ONLY** (Drop Only) results when an initiator and receiver cannot compromise protocol styles, that is, one client requires dynamic mode while the other can only support preregister mode, or if either explicitly has specified **XmDRAG_DROP_ONLY**.

Inherited Resources

All of the superclass resources inherited by XmDisplay are designated N/A (not applicable).

Related Information

ApplicationShell(3X), **Composite(3X)**, **Core(3X)**, **TopLevelShell(3X)**, **VendorShell(3X)**, **WMShell(3X)**, **XmGetXmDisplay(3X)**, and **XmScreen(3X)**.

XmDragCancel—A Drag and Drop function that terminates a drag transaction

Synopsis

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

void XmDragCancel (dragcontext)
        Widget          dragcontext;
```

Description

XmDragCancel terminates a drag operation and cancels any pending actions of the specified DragContext. This routine can only be called by the initiator client.

dragcontext Specifies the ID of the DragContext widget associated with the drag and drop transaction to be terminated

For a complete definition of DragContext and its associated resources, see **XmDragContext(3X)**.

Related Information

XmDragContext(3X) and **XmDragStart(3X)**.

XmDragContext—The DragContext widget class

Synopsis

#include <Xm/DragDrop.h>

Description

DragContexts are special widgets used in drag and drop transactions. A DragContext is implemented as a widget, but a client does not explicitly create a DragContext widget. Instead, a client initiates a drag and drop transaction by calling **XmDragStart**, and this routine initializes and returns a DragContext widget. There is a unique DragContext for each drag operation. The toolkit frees a DragContext when a transaction is complete; therefore, an application programmer should not explicitly destroy a DragContext.

Initiator and receiver clients both use DragContexts to track the state of a transaction. When the initiator and receiver of a transaction are in the same client, they share the same DragContext instance. If they are in different clients, there are two separate DragContexts. In this case, the initiator calls **XmDragStart** and the toolkit provides a DragContext for the receiver client. The only resources pertinent to the receiver are **XmNexportTargets** and **XmNnumExportTargets**. These can both be passed as arguments to the **XmDropSiteRetrieve** function to obtain information about the current drop site.

In general, in order to receive data, a drop site must share at least one target type and operation in common with a drag source. The DragContext resource, **XmNexportTargets**, identifies the selection targets for the drag source. These export targets are compared with the **XmNimportTargets** resource list specified by a drop site. The DragContext resource, **XmNdragOperations**, identifies the valid operations that can be applied to the source data by the initiator. The drop site counterpart resource is **XmNdropSiteOperations**, which indicates a drop site's supported operations.

A client uses DragIcon widgets to define the drag-over animation effects associated with a given drag and drop transaction. An initiator specifies a set of drag icons, selects a blending model, and sets foreground and background cursor colors with DragContext resources.

The type of drag-over visual used to represent a drag operation depends on the drag protocol style. In preregister mode, the server is grabbed, and either a cursor or a pixmap may be used as a drag-over visual. In dynamic mode, drag-over visuals

must be implemented with the X cursor. If the resulting drag protocol style is Drop Only or None and the **XmNdragInitiatorProtocolStyle** is **XmDRAG_DYNAMIC** or **XmDRAG_PREFER_DYNAMIC**, then a dynamic visual style (cursor) is used. Otherwise, a preregister visual style is used.

Classes

DragContext inherits behavior and resources from **Core**.

The class pointer is **xmDragContextClass**.

The class name is **XmDragContext**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using XtSetValues (S), retrieved by using XtGetValues (G), or is not applicable (N/A).

XmDragContext Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNblendModel XmCBlendModel	XmBLEND_ALL unsigned char	CG
XmNclientData XmCClientData	NULL XtPointer	CSG
XmNconvertProc XmCConvertProc	NULL XtConvertSelectionIncrProc	CSG
XmNcursorBackground XmCCursorBackground	dynamic Pixel	CSG
XmNcursorForeground XmCCursorForeground	dynamic Pixel	CSG
XmNdragDropFinishCallback XmCCallback	NULL XtCallbackList	CSG
XmNdragMotionCallback XmCCallback	NULL XtCallbackList	C
XmNdragOperations XmCDragOperations	XmDROP_COPY | XmDROP_MOVE unsigned char	C
XmNdropFinishCallback XmCCallback	NULL XtCallbackList	C
XmNdropSiteEnterCallback XmCCallback	NULL XtCallbackList	C
XmNdropSiteLeaveCallback XmCCallback	NULL XtCallbackList	C
XmNdropStartCallback XmCCallback	NULL XtCallbackList	C
XmNexportTargets XmCExportTargets	NULL Atom *	CSG
XmNincremental XmCIncremental	False Boolean	CSG
XmNinvalidCursorForeground XmCCursorForeground	dynamic Pixel	CSG

Name Class	Default Type	Access
XmNnoneCursorForeground XmCCursorForeground	dynamic Pixel	CSG
XmNnumExportTargets XmCNumExportTargets	0 Cardinal	CSG
XmNoperationChangedCallback XmCCallback	NULL XtCallbackList	C
XmNoperationCursorIcon XmCOperationCursorIcon	dynamic Widget	CSG
XmNsourceCursorIcon XmCSourceCursorIcon	dynamic Widget	CSG
XmNsourcePixmapIcon XmCSourcePixmapIcon	dynamic Widget	CSG
XmNstateCursorIcon XmCStateCursorIcon	dynamic Widget	CSG
XmNtopLevelEnterCallback XmCCallback	NULL XtCallbackList	C
XmNtopLevelLeaveCallback XmCCallback	NULL XtCallbackList	C
XmNvalidCursorForeground XmCCursorForeground	dynamic Pixel	CSG

XmNblendModel

Specifies which combination of DragIcons are blended to produce a drag-over visual.

XmBLEND_ALL

Blends all three DragIcons: the source, state and operation icons. The icons are layered from top to bottom with the operation icon on top and the source icon on the bottom. The hotspot is derived from the state icon.

XmBLEND_STATE_SOURCE

Blends the state and source icons only. The hotspot is derived from the state icon.

XmBLEND_JUST_SOURCE
Specifies that only the source icon is used, which the initiator updates as required.

XmBLEND_NONE
Specifies that no drag-over visual is generated. The client tracks the drop site status through callback routines and updates the drag-over visuals as necessary.

XmNclientData
Specifies the client data to be passed to **XmNconvertProc** when it is invoked.

XmNconvertProc
Specifies a procedure of type **XtConvertSelectionIncrProc** that converts the source data to the format(s) requested by the receiver client. The *widget* argument passed to this procedure is the DragContext widget. The selection atom passed is _MOTIF_DROP. If **XmNincremental** is False, the procedure should ignore the *max_length*, *client_data*, and *request_id* arguments and should handle the conversion atomically. Data returned by **XmNconvertProc** must be allocated using **XtMalloc**, and will be freed automatically by the toolkit after the transfer. For additional information on selection conversion procedures, see *X Toolkit Intrinsics—C Language Interface*.

XmNcursorBackground
Specifies the background pixel value of the cursor.

XmNcursorForeground
Specifies the foreground pixel value of the cursor when the state icon is not blended. This resource defaults to the foreground color of the widget passed to the **XmDragStart** function.

XmNdragDropFinishCallback
Specifies the list of callbacks that are called when the transaction is completed. The type of the structure whose address is passed to this callback is **XmDragDropFinishCallbackStruct**. The reason sent by the callback is **XmCR_DRAG_DROP_FINISH**.

XmNdragMotionCallback

Specifies the list of callbacks that are invoked when the pointer moves. The type of structure whose address is passed to this callback is **XmDragMotionCallbackStruct**. The reason sent by the callback is **XmCR_DRAG_MOTION**.

XmNdragOperations

Specifies the set of valid operations associated with an initiator client for a drag transaction. This resource is a bit mask that is formed by combining one or more of the following values using a bitwise operation such as inclusive OR (|): **XmDROP_COPY**, **XmDROP_LINK**, **XmDROP_MOVE**. The value **XmDROP_NOOP** for this resource indicates that no operations are valid. For Text and TextField widgets, this resource is set to **XmDROP_COPY** | **XmDROP_MOVE**; for List widgets, it is set to **XmDROP_COPY**.

XmNdropFinishCallback

Specifies the list of callbacks that are invoked when the drop is completed. The type of the structure whose address is passed to this callback is **XmDropFinishCallbackStruct**. The reason sent by the callback is **XmCR_DROP_FINISH**.

XmNdropSiteEnterCallback

Specifies the list of callbacks that are invoked when the pointer enters a drop site. The type of the structure whose address is passed to this callback is **XmDropSiteEnterCallbackStruct**. The reason sent by the callback is **XmCR_DROP_SITE_ENTER**.

XmNdropSiteLeaveCallback

Specifies the list of callbacks that are invoked when the pointer leaves a drop site. The type of the structure whose address is passed to this callback is **XmDropSiteLeaveCallbackStruct**. The reason sent by the callback is **XmCR_DROP_SITE_LEAVE**.

XmNdropStartCallback

Specifies the list of callbacks that are invoked when a drop is initiated. The type of the structure whose address is passed to this callback is **XmDropStartCallbackStruct**. The reason sent by the callback is **XmCR_DROP_START**.

XmNexportTargets

Specifies the list of target atoms associated with this source. This resource identifies the selection targets this source can be converted to.

XmNincremental

Specifies a Boolean value that indicates whether the transfer on the initiator side uses the Xt incremental selection transfer mechanism described in *X Toolkit Intrinsics—C Language Interface*. If the value is True, the initiator uses incremental transfer; if the value is False, the initiator uses atomic transfer.

XmNinvalidCursorForeground

Specifies the foreground pixel value of the cursor when the state is invalid. This resource defaults to the value of the **XmNcursorForeground** resource.

XmNnoneCursorForeground

Specifies the foreground pixel value of the cursor when the state is none. This resource defaults to the value of the **XmNcursorForeground** resource.

XmNnumExportTargets

Specifies the number of entries in the list of export targets.

XmNoperationChangedCallback

Specifies the list of callbacks that are invoked when the drag is started and when the user requests that a different operation be applied to the drop. The type of the structure whose address is passed to this callback is **XmOperationChangedCallbackStruct**. The reason sent by the callback is **XmCR_OPERATION_CHANGED**.

XmNoperationCursorIcon

Specifies the cursor icon used to designate the type of operation performed by the drag transaction. If NULL, **XmScreen** resources provide default icons for copy, link, and move operations.

XmNsourceCursorIcon

Specifies the cursor icon used to represent the source when a dynamic visual style is used. If NULL, the **XmNdefaultSourceCursorIcon** resource of **XmScreen** provides a default cursor icon.

XmNsourcePixmapIcon

Specifies the pixmap icon used to represent the source when a preregister visual style is used. The icon is used in conjunction with the colormap of the widget passed to **XmDragStart**. If NULL, **XmNsourceCursorIcon** is used.

XmNstateCursorIcon

Specifies the cursor icon used to designate the state of a drop site. If NULL, **XmScreen** resources provide default icons for a valid, invalid, and no drop site condition.

XmNtopLevelEnterCallback

Specifies the list of callbacks that are called when the pointer enters a top-level window or root window (due to changing screens). The type of the structure whose address is passed to this callback is **XmTopLevelEnterCallbackStruct**. The reason sent by the callback is **XmCR_TOP_LEVEL_ENTER**.

XmNtopLevelLeaveCallback

Specifies the list of callbacks that are called when the pointer leaves a top level window or the root window (due to changing screens). The type of the structure whose address is passed to this callback is **XmTopLevelLeaveCallbackStruct**. The reason sent by the callback is **XmCR_TOP_LEVEL_LEAVE**.

XmNvalidCursorForeground

Specifies the foreground pixel value of the cursor designated as a valid cursor icon.

Inherited Resources

DragContext inherits behavior and resources from the superclass described in the following table. For a complete description of each resource, refer to the **Core** reference page.

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

Each of the DragContext callbacks has an associated callback structure.

A pointer to the following structure is passed to the **XmNdragDropFinishCallback** callback:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
}XmDragDropFinishCallbackStruct, *XmDragDropFinishCallback;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

timeStamp Specifies the time at which either the drag or the drop was completed

A pointer to the following structure is passed to callbacks for **XmNdragMotionCallback**:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
    Position        x;
    Position        y;
}XmDragMotionCallbackStruct, *XmDragMotionCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

operation Identifies an operation.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *operation* to the value of the *operation* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns.

If the toolkit has not called an **XmNdragProc** and the pointer is within an active drop site, the toolkit initializes *operation* by selecting an operation from the bitwise AND of the initial value of the *operations* member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

If the toolkit has not called an **XmNdragProc** and the pointer is not within an active drop site, the toolkit initializes *operation* by selecting an operation from the initial value of the *operations* member. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

operations Indicates the set of operations supported for the source data.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *operations* to the bitwise AND of the DropSite's **XmNdropOperations** and the value of the *operations* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

If the toolkit has not called an **XmNdragProc** and the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource.

If the toolkit has not called an **XmNdragProc** and the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

dropSiteStatus

Indicates whether or not a drop site is valid.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *dropSiteStatus* to the value of the *dropSiteStatus* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns.

If the toolkit has not called an **XmNdragProc**, it initializes *dropSiteStatus* as follows: the toolkit initializes *dropSiteStatus* to **XmNO_DROP_SITE** if the pointer is over an inactive drop site or is not over a drop site. The toolkit initializes *dropSiteStatus* to **XmDROP_SITE_VALID** if all the following conditions are met:

- The pointer is over an active drop site.
- The DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible.
- The initial value of the *operation* member is not **XmDROP_NOOP**.

Otherwise, the toolkit initializes *dropSiteStatus* to **XmDROP_SITE_INVALID**.

A pointer to the following structure is passed for the **XmNdropFinishCallback** callback:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
    unsigned char   dropAction;
    unsigned char   completionStatus;
}XmDropFinishCallbackStruct, *XmDropFinishCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the time at which the drop was completed.

operation Identifies an operation.

If the pointer is over an active drop site when the drop begins, the toolkit initializes *operation* to the value of the *operation* member of the **XmDropProcCallbackStruct** at the time the DropSite's **XmNdropProc** returns.

If the pointer is not over an active drop site when the drop begins, the toolkit initializes *operation* by selecting an operation from the initial value of the *operations* member. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If it finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

operations Indicates the set of operations supported for the source data.

If the pointer is over an active drop site when the drop begins, the toolkit initializes *operations* to the bitwise AND of the DropSite's **XmNdropOperations** and the value of the *operations* member of the **XmDropProcCallbackStruct** at the time the DropSite's **XmNdropProc** returns. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

If the pointer is not over an active drop site when the drop begins and if the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource.

If the pointer is not over an active drop site when the drop begins and if the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

dropSiteStatus

Indicates whether or not a drop site is valid.

If the pointer is over an active drop site when the drop begins, the toolkit initializes *dropSiteStatus* to the value of the *dropSiteStatus* member of the **XmDropProcCallbackStruct** at the time the DropSite's **XmNdropProc** returns.

If the pointer is not over an active drop site when the drop begins, the toolkit initializes *dropSiteStatus* to **XmNO_DROP_SITE**.

dropAction Identifies the drop action. The values are **XmDROP**, **XmDROP_CANCEL**, **XmDROP_HELP**, and **XmDROP_INTERRUPT**. The **XmDROP_INTERRUPT** value is currently unsupported; if specified, it will be interpreted as an **XmDROP_CANCEL**.

completionStatus

An IN/OUT member that indicates the status of the drop action. After the last callback procedure has returned, the final value of this member determines what visual transition effects will be applied. There are two values:

XmDROP_SUCCESS	The drop was successful.
XmDROP_FAILURE	The drop was unsuccessful.

A pointer to the following structure is passed to callbacks for **XmNdropSiteEnterCallback**:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
    Position        x;
    Position        y;
}XmDropSiteEnterCallbackStruct, *XmDropSiteEnterCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the time the crossing event occurred.

operation Identifies an operation.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *operation* to the value of the *operation* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns.

If the toolkit has not called an **XmNdragProc**, it initializes *operation* by selecting an operation from the bitwise AND of the initial value of the *operations* member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

operations Indicates the set of operations supported for the source data.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *operations* to the bitwise AND of the DropSite's **XmNdropOperations** and the value of the *operations* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

If the toolkit has not called an **XmNdragProc** and the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource.

If the toolkit has not called an **XmNdragProc** and the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

dropSiteStatus

Indicates whether or not a drop site is valid.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *dropSiteStatus* to the value of the *dropSiteStatus* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns.

If the toolkit has not called **XmNdragProc**, it initializes *dropSiteStatus* to **XmDROP_SITE_VALID** if the DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible and if the initial value of the *operation* member is not **XmDROP_NOOP**. Otherwise, the toolkit initializes *dropSiteStatus* to **XmDROP_SITE_INVALID**.

x Indicates the x-coordinate of the pointer in root window coordinates.

y Indicates the y-coordinate of the pointer in root window coordinates.

A pointer to the following structure is passed to callbacks for **XmNdropSiteLeaveCallback**:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
}XmDropSiteLeaveCallbackStruct, *XmDropSiteLeaveCallback;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

timeStamp Specifies the timestamp of the logical event

A pointer to the following structure is passed for the **XmNdropStartCallback** callback:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
    unsigned char   dropAction;
    Position        x;
    Position        y;
}XmDropStartCallbackStruct, *XmDropStartCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the time at which the drag was completed.

operation Identifies an operation.

If the pointer is over an active drop site when the drop begins, the toolkit initializes *operation* to the value of the *operation* member of the **XmDropProcCallbackStruct** at the time the DropSite's **XmNdropProc** returns.

If the pointer is not over an active drop site when the drop begins, the toolkit initializes *operation* by selecting an operation from the initial value of the *operations* member. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If it finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

operations Indicates the set of operations supported for the source data.

If the pointer is over an active drop site when the drop begins, the toolkit initializes *operations* to the bitwise AND of the DropSite's **XmNdropOperations** and the value of the *operations* member of the **XmDropProcCallbackStruct** at the time the DropSite's **XmNdropProc** returns. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

If the pointer is not over an active drop site when the drop begins and if the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource.

If the pointer is not over an active drop site when the drop begins and if the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

dropSiteStatus

Indicates whether or not a drop site is valid.

If the pointer is over an active drop site when the drop begins, the toolkit initializes *dropSiteStatus* to the value of the *dropSiteStatus* member of the **XmDropProcCallbackStruct** at the time the DropSite's **XmNdropProc** returns.

If the pointer is not over an active drop site when the drop begins, the toolkit initializes *dropSiteStatus* to **XmNO_DROP_SITE**.

dropAction An IN/OUT member that identifies the drop action. The values are **XmDROP**, **XmDROP_CANCEL**, **XmDROP_HELP**, and **XmDROP_INTERRUPT**. The value of *dropAction* can be modified to change the action actually initiated. The value **XmDROP_INTERRUPT** is currently unsupported; if specified, it will be interpreted as an **XmDROP_CANCEL**.

x Indicates the x-coordinate of the pointer in root window coordinates.

y Indicates the y-coordinate of the pointer in root window coordinates.

A pointer to the following structure is passed to the **XmNoperationChangedCallback** callback:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
}XmOperationChangedCallbackStruct, *XmOperationChangedCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the time at which the crossing event occurred.

operation Identifies an operation.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *operation* to the value of the *operation* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns.

If the toolkit has not called an **XmNdragProc**, and the pointer is within an active drop site, the toolkit initializes *operation* by selecting an operation from the bitwise AND of the initial value of the *operations* member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

If the toolkit has not called an **XmNdragProc**, and the pointer is not within an active drop site, the toolkit initializes *operation* by selecting an operation from the initial value of the *operations* member. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

operations Indicates the set of operations supported for the source data.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *operations* to the bitwise AND of the DropSite's **XmNdropOperations** and the value of the *operations* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

If the toolkit has not called an **XmNdragProc**, and the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource.

If the toolkit has not called an **XmNdragProc**, and the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

dropSiteStatus

Indicates whether or not a drop site is valid.

If the toolkit has just called a DropSite's **XmNdragProc**, the toolkit initializes *dropSiteStatus* to the value of the *dropSiteStatus* member of the **XmDragProcCallbackStruct** at the time the DropSite's **XmNdragProc** returns.

If the toolkit has not called an **XmNdragProc** it initializes *dropSiteStatus* to **XmNO_DROP_SITE** if the pointer is over an inactive drop site or is not over a drop site. The toolkit initializes *dropSiteStatus* to **XmDROP_SITE_VALID** if all the following conditions are met:

- The pointer is over an active drop site
- The DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible
- The initial value of the *operation* member is not **XmDROP_NOOP**

Otherwise, the toolkit initializes *dropSiteStatus* to **XmDROP_SITE_INVALID**.

A pointer to the following structure is passed to callbacks for **XmNtopLevelEnterCallback**:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    Screen          screen;
    Window          window;
    Position        x;
    Position        y;
    unsigned char   dragProtocolStyle;
}XmTopLevelEnterCallbackStruct, *XmTopLevelEnterCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

screen Specifies the screen associated with the top-level window or root window being entered.

window Specifies the ID of the top-level window or root window being entered.

x Indicates the x-coordinate of the pointer in root window coordinates.

y Indicates the y-coordinate of the pointer in root window coordinates.

dragProtocolStyle
Specifies the protocol style adopted by the initiator. The values are **XmDRAG_DROP_ONLY**, **XmDRAG_DYNAMIC**, **XmDRAG_NONE**, and **XmDRAG_PREREGISTER**.

A pointer to the following structure is passed to callbacks for **XmNtopLevelLeaveCallback**:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    Screen          screen;
    Window          window;
}XmTopLevelLeaveCallbackStruct, *XmTopLevelLeaveCallback;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

timeStamp Specifies the timestamp of the logical event

screen Specifies a screen associated with the top-level window or root window being left

window Specifies the ID of the top-level window or root window being left

Translations

The XmDragContext translations are described in the following list. These translations may not directly correspond to a translation table.

BTransfer Motion: **DragMotion()**

BTransfer Release: **FinishDrag()**

KCancel: **CancelDrag()**

KHelp: **HelpDrag()**

Action Routines

The XmDragContext action routines are

CancelDrag():
Cancels the drag operation and frees the associated DragContext.

DragMotion():
Drags the selected data as the pointer is moved.

FinishDrag():
Finishes the drag operation and starts the drop operation.

HelpDrag(): Initiates a conditional drop that enables the receiver to provide help information to the user. The user can cancel or continue the drop operation in response to this information.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmDisplay(3X)**, **XmDragCancel(3X)**, **XmDragIcon(3X)**, **XmDragStart(3X)**, **XmDropSite(3X)**, **XmDropTransfer(3X)**, and **XmScreen(3X)**.

XmDragIcon—The DragIcon widget class

Synopsis

#include <Xm/DragDrop.h>

Description

A DragIcon is a component of the visual used to represent the source data in a drag and drop transaction. During a drag operation, a real or simulated X cursor provides drag-over visuals consisting of a static portion that represents the object being dragged, and dynamic cues that provide visual feedback during the drag operation. The visual is attained by blending together various **XmDragIcons** specified in the **XmDragContext** associated with the drag operation.

The static portion of the drag-over visual is the graphic representation that identifies the drag source. For example, when a user drags several items within a list, a DragIcon depicting a list might be supplied as the visual. The **XmDragContext** resources, **XmNsourceCursorIcon** or **XmNsourcePixmapIcon**, specify a DragIcon to use for the static portion of the visual.

A drag-over visual incorporates dynamic cues in order to provide visual feedback in response to the user's actions. For instance, the drag-over visual might use different indicators to identify the type of operation (copy, link, or move) being performed. Dynamic cues could also alert the user that a drop site is valid or invalid as the pointer traverses the drop site. The **XmNoperationCursorIcon** and **XmNstateCursorIcon** resources of **XmDragContext** specify DragIcons for dynamic cues.

A drag-over visual typically consists of a source, operation and state DragIcon. The **XmNblendModel** resource of **XmDragContext** offers several options that determine which icons are blended to produce the drag-over visual. DragIcon resources control the relative position of the operation and state icons (if used). If a particular DragIcon is not specified, the toolkit uses the **XmScreen** default DragIcons.

An application initializes a DragIcon with the function **XmCreateDragIcon** or through entries in the resource database. If a pixmap and its mask (optional) are specified in the resource database, the toolkit converts the values in the X11 Bitmap file format and assigns values to the corresponding resources.

Classes

DragIcon inherits behavior and a resource from **Object**.

The class pointer is **xmDragIconObjectClass**.

The class name is **XmDragIcon**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmDragIcon Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNattachment XmCAttachment	XmATTACH_NORTH_WEST unsigned char	CSG
XmNdepth XmCDepth	1 int	CSG
XmNheight XmCHeight	0 Dimension	CSG
XmNhotX XmCHot	0 Position	CSG
XmNhotY XmCHot	0 Position	CSG
XmNmask XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNoffsetX XmCOffset	0 Position	CSG
XmNoffsetY XmCOffset	0 Position	CSG
XmNpixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNwidth XmCWidth	0 Dimension	CSG

XmNattachment

Specifies a relative location on the source icon for the attachment of the state or operation icon. The origin of the state and operation icons is aligned with the specified compass point on the source icon. The **XmNoffsetX** and **XmNoffsetY** resources can be used to further refine the icon positions. The possible values are

XmATTACH_NORTH_WEST

Attaches the origin of the state or operation icon to the northwest point on the source icon.

XmATTACH_NORTH

Attaches the origin of the state or operation icon to the north point on the source icon.

XmATTACH_NORTH_EAST
Attaches the origin of the state or operation icon to the northeast point on the source icon.

XmATTACH_EAST
Attaches the origin of the state or operation icon to the east point on the source icon.

XmATTACH_SOUTH_EAST
Attaches the origin of the state or operation icon to the southeast point on the source icon.

XmATTACH_SOUTH
Attaches the origin of the state or operation icon to the south point on the source icon.

XmATTACH_SOUTH_WEST
Attaches the origin of the state or operation icon to the southwest point on the source icon.

XmATTACH_WEST
Attaches the origin of the state or operation icon to the west point on the source icon.

XmATTACH_CENTER
Attaches the origin of the state or operation icon to the center of the source icon. The **XmNoffsetX** and **XmNoffsetY** resources may be used to center the attached icon.

XmATTACH_HOT
Attaches the hotspot coordinates of a state or operation DragIcon to an x,y position on the source icon. The x,y coordinate is taken from the event passed to the **XmDragStart** function, and made relative to the widget passed as an argument to the same function.

XmNdepth Specifies the depth of the pixmap.

XmNheight Specifies the height of the pixmap.

XmNhotX Specifies the x-coordinate of the hotspot of a cursor DragIcon in relation to the origin of the pixmap bounding box.

XmNhotY Specifies the y-coordinate of the hotspot of a cursor DragIcon in relation to the origin of the pixmap bounding box.

XmNmask Specifies a pixmap of depth 1 to use as the DragIcon mask pixmap.

XmNoffsetX Specifies a horizontal offset (in pixels) of the origin of the state or operation icon relative to the attachment point on the source icon. A positive offset value moves the origin to the right; a negative value moves the origin to the left.

XmNoffsetY Specifies a vertical offset (in pixels) of the origin of the state or operation icon relative to the attachment point on the source icon. A positive offset value moves the origin down; a negative value moves the origin up.

XmNpixmap Specifies a pixmap to use as the DragIcon pixmap.

XmNwidth Specifies the width of the pixmap.

Inherited Resources

DragIcon inherits behavior and a resource from **Object**. For a complete description of this resource, refer to the **Object** reference page.

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Related Information

Object(3X), **XmCreateDragIcon(3X)**, **XmDisplay(3X)**, **XmDragContext(3X)**, **XmDropSite(3X)**, **XmDropTransfer(3X)**, and **XmScreen(3X)**.

XmDragStart—A Drag and Drop function that initiates a drag and drop transaction

Synopsis

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

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

Description

XmDragStart initiates a drag operation. This routine returns the DragContext widget that it initializes for the associated drag transaction. The toolkit is responsible for freeing the DragContext when the drag and drop transaction is complete.

widget Specifies the ID of the smallest widget and/or gadget that encloses the source elements selected for a drag operation.

event Specifies the **XEvent** that triggered the drag operation. This event must be a ButtonPress event.

arglist Specifies the argument list. Any **XmDragContext** resources not specified in the argument list are obtained from the resource database or are set to their default values.

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DragContext and its associated resources, see **XmDragContext(3X)**.

Return Value

Returns the ID of the DragContext widget that controls this drag and drop transaction.

Related Information

XmDragCancel(3X) and **XmDragContext(3X)**.

XmDrawingArea—The DrawingArea widget class

Synopsis

#include <Xm/DrawingA.h>

Description

DrawingArea is an empty widget that is easily adaptable to a variety of purposes. It does no drawing and defines no behavior except for invoking callbacks. Callbacks notify the application when graphics need to be drawn (exposure events or widget resize) and when the widget receives input from the keyboard or mouse.

Applications are responsible for defining appearance and behavior as needed in response to DrawingArea callbacks.

DrawingArea is also a composite widget and subclass of **XmManager** that supports minimal geometry management for multiple widget or gadget children.

Classes

DrawingArea inherits behavior and resources from the **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is **xmDrawingAreaWidgetClass**.

The class name is **XmDrawingArea**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmDrawingArea Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNexposeCallback XmCCallback	NULL XtCallbackList	C
XmNinputCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	10 Dimension	CSG
XmNmarginWidth XmCMarginWidth	10 Dimension	CSG
XmNresizeCallback XmCCallback	NULL XtCallbackList	C
XmNresizePolicy XmCResizePolicy	XmRESIZE_ANY unsigned char	CSG

XmNexposeCallback

Specifies the list of callbacks that is called when DrawingArea receives an exposure event. The callback reason is **XmCR_EXPOSE**. The callback structure also includes the exposure event.

The default bit gravity for Manager windows is **NorthWestGravity**. This may cause the **XmNexposeCallback** procedures not to be invoked when the DrawingArea window is made smaller.

XmNinputCallback

Specifies the list of callbacks that is called when the DrawingArea receives a keyboard or mouse event (key or button, up or down). The callback reason is **XmCR_INPUT**. The callback structure also includes the input event.

XmNmarginHeight

Specifies the minimum spacing in pixels between the top or bottom edge of DrawingArea and any child widget.

XmNmarginWidth

Specifies the minimum spacing in pixels between the left or right edge of DrawingArea and any child widget.

XmNresizeCallback

Specifies the list of callbacks that is called when the DrawingArea is resized. The callback reason is **XmCR_RESIZE**.

XmNresizePolicy

Controls the policy for resizing DrawingArea widgets. Possible values include **XmRESIZE_NONE** (fixed size), **XmRESIZE_ANY** (shrink or grow as needed), and **XmRESIZE_GROW** (grow only).

Inherited Resources

DrawingArea inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the reference page for that superclass.

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	0 Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    Window      window;
} XmDrawingAreaCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback. This is NULL for the **XmNresizeCallback**.

window Is set to the widget window.

Translations

XmDrawingArea inherits translations from XmManager. Before calling the XmManager actions, all events in the inherited translations except **<BtnMotion>**, **<EnterWindow>**, **<LeaveWindow>**, **<FocusIn>**, and **<FocusOut>** also call the **DrawingAreaInput()** action.

XmDrawingArea has the following additional translations. These translations may not directly correspond to a translation table.

MAny BAny Press:	**DrawingAreaInput()**
MAny BAny Release:	**DrawingAreaInput()**
MAny KAny Press:	**DrawingAreaInput()** **ManagerGadgetKeyInput()**
MAny KAny Release:	**DrawingAreaInput()**

Action Routines

The XmDrawingArea action routines are described below:

DrawingAreaInput():
Unless the event takes place in a gadget, calls the callbacks for **XmNinputCallback**

ManagerGadgetKeyInput():
Causes the current gadget to process a keyboard event

Additional Behavior

The XmDrawingArea widget has the following additional behavior:

<Expose>: Calls the callbacks for **XmNexposeCallback**

<Widget Resize>:
Calls the callbacks for **XmNresizeCallback**

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreateDrawingArea(3X)**, and **XmManager(3X)**.

XmDrawnButton—The DrawnButton widget class

Synopsis

#include <Xm/DrawnB.h>

Description

The DrawnButton widget consists of an empty widget window surrounded by a shadow border. It provides the application developer with a graphics area that can have PushButton input semantics.

Callback types are defined for widget exposure and widget resize to allow the application to redraw or reposition its graphics. If the DrawnButton widget has a highlight and shadow thickness, the application should not draw in that area. To avoid drawing in the highlight and shadow area, create the graphics context with a clipping rectangle for drawing in the widget. The clipping rectangle should take into account the size of the widget's highlight thickness and shadow.

Classes

DrawnButton inherits behavior and resources from the **Core**, **XmPrimitive**, and **XmLabel** classes.

The class pointer is **xmDrawnButtonWidgetClass**.

The class name is **XmDrawnButton**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmDrawnButton Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNarmCallback XmCCallback	NULL XtCallbackList	C
XmNdisarmCallback XmCCallback	NULL XtCallbackList	C
XmNexposeCallback XmCCallback	NULL XtCallbackList	C
XmNmultiClick XmCMultiClick	dynamic unsigned char	CSG
XmNpushButtonEnabled XmCPushButtonEnabled	False Boolean	CSG
XmNresizeCallback XmCCallback	NULL XtCallbackList	C
XmNshadowType XmCShadowType	XmSHADOW_ETCHED_IN unsigned char	CSG

XmNactivateCallback

Specifies the list of callbacks that is called when the widget becomes selected. The reason sent by the callback is **XmCR_ACTIVATE**.

XmNarmCallback

Specifies the list of callbacks that is called when the widget becomes armed. The reason sent by the callback is **XmCR_ARM**.

XmNdisarmCallback

Specifies the list of callbacks that is called when the widget becomes disarmed. The reason sent by the callback is **XmCR_DISARM**.

XmNexposeCallback

Specifies the list of callbacks that is called when the widget receives an exposure event. The reason sent by the callback is **XmCR_EXPOSE**.

XmNmultiClick

If a button click is followed by another button click within the time span specified by the display's multiclick time, and this resource is set to **XmMULTICLICK_DISCARD**, the second click is not processed. If this resource is set to **XmMULTICLICK_KEEP**, the event is processed and *click_count* is incremented in the callback structure. When the button is not in a menu, the default value is **XmMULTICLICK_KEEP**.

XmNpushButtonEnabled

Enables or disables the 3-dimensional shadow drawing as in PushButton.

XmNresizeCallback

Specifies the list of callbacks that is called when the widget receives a resize event. The reason sent by the callback is **XmCR_RESIZE**. The event returned for this callback is NULL.

XmNshadowType

Describes the drawing style for the DrawnButton. This resource can have the following values:

XmSHADOW_IN

Draws the DrawnButton so that the shadow appears inset. This means that the bottom shadow visuals and top shadow visuals are reversed.

XmSHADOW_OUT

Draws the DrawnButton so that the shadow appears outset.

XmSHADOW_ETCHED_IN

Draws the DrawnButton using a double line. This gives the effect of a line etched into the window. The thickness of the double line is equal to the value of **XmNshadowThickness**.

XmSHADOW_ETCHED_OUT

Draws the DrawnButton using a double line. This gives the effect of a line coming out of the window. The thickness of the double line is equal to the value of **XmNshadowThickness**.

Inherited Resources

DrawnButton inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmLabel Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	N/A
XmNacceleratorText XmCAcceleratorText	NULL XmString	N/A
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	"\0" XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	0 Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	0 Dimension	CSG
XmNmarginRight XmCMarginRight	0 Dimension	CSG
XmNmarginTop XmCMarginTop	0 Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG
XmNmnemonic XmCMnemonic	NULL KeySym	N/A

Name Class	Default Type	Access
XmNmnemonicCharSet XmCMnemonicCharSet	XmFONTLIST_DEFAULT_TAG String	N/A
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    Window      window;
    int         click_count;
} XmDrawnButtonCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback. This is NULL for **XmNresizeCallback**.

window Is set to the window ID in which the event occurred.

click_count Contains the number of clicks in the last multiclick sequence if the **XmNmultiClick** resource is set to **XmMULTICLICK_KEEP**, otherwise it contains 1. The activate callback is invoked for each click if **XmNmultiClick** is set to **XmMULTICLICK_KEEP**.

Translations

XmDrawnButton includes translations from Primitive. Additional XmDrawnButton translations are described in the following list. These translations may not directly correspond to a translation table.

BSelect Press: **Arm()**

BSelect Click: **Activate()**
Disarm()

Select Release:	**Activate()** **Disarm()**
BSelect Press 2+:	**MultiArm()**
BSelect Release 2+:	**MultiActivate()**
KSelect:	**ArmAndActivate()**
KHelp:	**Help()**

Action Routines

The XmDrawnButton action routines are

Activate(): If **XmNpushButtonEnabled** is True, redraws the shadow in the unselected state; otherwise, redraws the shadow according to **XmNshadowType**. If the pointer is within the DrawnButton, calls the **XmNactivateCallback** callbacks.

Arm(): If **XmNpushButtonEnabled** is True, redraws the shadow in the selected state; otherwise, redraws the shadow according to **XmNshadowType**. Calls the callbacks for **XmNarmCallback**.

ArmAndActivate():

If **XmNpushButtonEnabled** is True, redraws the shadow in the selected state; otherwise, redraws the shadow according to **XmNshadowType**. Calls the callbacks for **XmNarmCallback**.

If **XmNpushButtonEnabled** is True, the shadow is redrawn in the unselected state; otherwise, the shadow is redrawn according to **XmNshadowType**. The callbacks for **XmNactivateCallback** and **XmNdisarmCallback** are called. These actions happen either immediately or at a later time:

Disarm(): Marks the DrawnButton as unselected and calls the callbacks for **XmNdisarmCallback**.

Help(): Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MultiActivate():

If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action increments *click_count* in the callback structure. If **XmNpushButtonEnabled** is True, this action redraws the shadow

in the unselected state; otherwise, it redraws the shadow according to **XmNshadowType**. If the pointer is within the DrawnButton, this action calls the **XmNactivateCallback** callbacks and calls the callbacks for **XmNdisarmCallback**.

MultiArm(): If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP** and if **XmNpushButtonEnabled** is True, this action redraws the shadow in the selected state; otherwise, it redraws the shadow according to **XmNshadowType** and Calls the callbacks for **XmNarmCallback**.

Additional Behavior

This widget has the following additional behavior:

<EnterWindow>:

Draws the shadow in its selected state if **XmNpushButtonEnabled** is True and if the cursor leaves and re-enters the window while **BSelect** is pressed.

<LeaveWindow>:

Draws the shadow in its unselected state if **XmNpushButtonEnabled** is True and if the cursor leaves the window while **BSelect** is pressed.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateDrawnButton**, **XmLabel(3X)**, **XmPrimitive(3X)**, **XmPushButton**, and **XmSeparator(3X)**.

XmDropSite—The DropSite Registry

Synopsis

#include <Xm/DragDrop.h>

Description

A client registers a widget or gadget as a drop site using the **XmDropSiteRegister** function. In addition, this routine defines the behavior and capabilities of a drop site by specifying appropriate resources. For example, the **XmNimportTargets** and **XmNnumImportTargets** resources identify respectively the selection target types and number of types supported by a drop site. The visual animation effects associated with a drop site are also described with DropSite resources.

Drop site animation effects that occur in response to the pointer entering a valid drop site are called drag-under effects. A receiver can select from several animation styles supplied by the toolkit or can provide customized animation effects. Drag-under effects supplied by the toolkit include border highlighting, shadow in/out drawing, and pixmap representation.

When a preregister drag protocol style is used, the toolkit generates drag-under visual effects based on the value of the **XmNanimationStyle** resource. In dynamic mode, if the drop site **XmNdragProc** resource is NULL, the toolkit also provides animation effects based on the **XmNanimationStyle** resource. Otherwise, if the **XmNdragProc** routine is specified, the receiver can either assume responsibility for animation effects (through the **XmNdragProc** routine) or rely on the toolkit to provide animation.

Drop sites may overlap. The initial stacking order corresponds to the order in which the drop sites were registered. When a drop site overlaps another drop site, the drag-under effects of the drop site underneath are clipped by the obscuring drop site(s).

The **XmDropSiteUpdate** routine sets resources for a widget that is registered as a drop site. **XmDropSiteRetrieve** gets drop site resource values previously specified for a registered widget. These routines are used instead of **XtSetValues** and **XtGetValues**.

Classes

XmDropSite does not inherit from any widget class.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of

the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XmDropSiteUpdate** (S), retrieved by using **XmDropSiteRetrieve** (G), or is not applicable (N/A).

XmDropSite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNanimationMask XmCAnimationMask	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNanimationPixmap XmCAnimationPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNanimationPixmapDepth XmCAnimationPixmapDepth	0 int	CSG
XmNanimationStyle XmCAnimationStyle	XmDRAG_UNDER_HIGHLIGHT unsigned char	CSG
XmNdragProc XmCDragProc	NULL XtCallbackProc	CSG
XmNdropProc XmCDropProc	NULL XtCallbackProc	CSG
XmNdropRectangles XmCDropRectangles	dynamic XRectangle *	CSG
XmNdropSiteActivity XmCDropSiteActivity	XmDROP_SITE_ACTIVE unsigned char	CSG
XmNdropSiteOperations XmCDropSiteOperations	XmDROP_MOVE I XmDROP_COPY unsigned char	CSG
XmNdropSiteType XmCDropSiteType	XmDROP_SITE_SIMPLE unsigned char	CG
XmNimportTargets XmCImportTargets	NULL Atom *	CSG
XmNnumDropRectangles XmCNumDropRectangles	1 Cardinal	CSG
XmNnumImportTargets XmCNumImportTargets	0 Cardinal	CSG

XmNanimationMask

Specifies a mask to use with the pixmap specified by **XmNanimationPixmap** when the animation style is **XmDRAG_UNDER_PIXMAP**.

XmNanimationPixmap

Specifies a pixmap for drag-under animation when the animation style is **XmDRAG_UNDER_PIXMAP**. The pixmap is drawn with its origin at the upper left corner of the bounding box of the drop site. If the drop site window is larger than the animation pixmap, the portion of the window not covered by the pixmap will be tiled with the window's background color.

XmNanimationPixmapDepth

Specifies the depth of the pixmap specified by the **XmNanimationPixmap** resource. When the depth is 1, the colors are taken from the foreground and background of the drop site widget. For any other value, drop site animation occurs only if the **XmNanimationPixmapDepth** matches the depth of the drop site window. Colors are derived from the current colormap.

XmNanimationStyle

Specifies the drag-under animation style used when a drag enters a valid drop site. The possible values are

XmDRAG_UNDER_HIGHLIGHT

The drop site uses highlighting effects.

XmDRAG_UNDER_SHADOW_OUT

The drop site uses an outset shadow.

XmDRAG_UNDER_SHADOW_IN

The drop site uses an inset shadow.

XmDRAG_UNDER_PIXMAP

The drop site uses the pixmap specified by **XmNanimationPixmap** to indicate that it can receive the drop.

XmDRAG_UNDER_NONE

The drop site does not use animation effects. A client using a dynamic protocol, may provide drag-under effects in its **XmNdragProc** routine.

XmNdragProc

Specifies the procedure that is invoked when the drop site receives a crossing, motion, or operation changed message. This procedure is called only when a dynamic protocol is used. The type of structure whose address is passed to this procedure is **XmDragProcCallbackStruct**. The reason sent to the procedure is one of the following:

- **XmCR_DROP_SITE_ENTER_MESSAGE**
- **XmCR_DROP_SITE_LEAVE_MESSAGE**
- **XmCR_DRAG_MOTION_MESSAGE**
- **XmCR_OPERATION_CHANGED_MESSAGE**

The drag procedure may change the values of some members of the **XmDragProcCallbackStruct** passed to it. After the drag procedure returns, the toolkit uses the final values in initializing some members of the callback structure passed to the appropriate callbacks of the initiator (the DragContext's **XmNdropSiteEnterCallback**, **XmNdropSiteLeaveCallback**, **XmNdragMotionCallback**, or **XmNoperationChangedCallback** callbacks).

XmNdropProc

Specifies the procedure that is invoked when a drop (excluding a cancel or interrupt action) occurs on a drop site regardless of the status of the drop site. The type of the structure whose address is passed to this procedure is **XmDropProcCallbackStruct**. The reason sent to the procedure is **XmCR_DROP_MESSAGE**.

The drop procedure may change the values of some members of the **XmDropProcCallbackStruct** passed to it. After the drop procedure returns, the toolkit uses the final values in initializing some members of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

XmNdropRectangles

Specifies a list of rectangles that describe the shape of a drop site. The locations of the rectangles are relative to the origin of the enclosing object. When **XmNdropRectangles** is NULL, the drop site is assumed to be the sensitive area of the enclosing widget. If **XmNdropSiteType** is **XmDROP_SITE_COMPOSITE**, this resource cannot be specified by the application.

XmNdropSiteActivity
Indicates whether a drop site is active or inactive. The values are **XmDROP_SITE_ACTIVE** and **XmDROP_SITE_INACTIVE**. An active drop site can receive a drop, whereas an inactive drop site is dormant. An inactive drop site is treated as if it was not a registered drop site and any drag-under visuals associated with entering or leaving the drop site do not occur. However, it is still used for clipping drag-under effects.

XmNdropSiteOperations
Specifies the set of valid operations associated with a drop site. This resource is a bit mask that is formed by combining one or more of the following values using a bitwise operation such as inclusive OR (|): **XmDROP_COPY**, **XmDROP_LINK**, and **XmDROP_MOVE**. The value **XmDROP_NOOP** for this resource indicates that no operations are valid.

XmNdropSiteType
Specifies the type of the drop site. The possible values are

XmDROP_SITE_SIMPLE
The widget does not have any additional children that are registered as drop sites.

XmDROP_SITE_COMPOSITE
The widget will have children that are registered as drop sites.

XmNimportTargets
Specifies the list of target atoms that this drop site accepts.

XmNnumDropRectangles
Specifies the number of rectangles in the **XmNdropRectangles** list. If the drop site type is **XmDROP_SITE_COMPOSITE**, this resource can not be specified by the application.

XmNnumImportTargets
Specifies the number of atoms in the target atom list.

Callback Information

A pointer to the following structure is passed to the **XmNdragProc** routine when the drop site receives crossing, motion, or operation changed messages:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    Widget          dragContext
    Position        x;
    Position        y;
    unsigned char   dropSiteStatus;
    unsigned char   operation;
    unsigned char   operations;
    Boolean         animate;
} XmDragProcCallbackStruct, *XmDragProcCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

dragContext Specifies the ID of the DragContext widget associated with the transaction.

x Indicates the x-coordinate of the pointer relative to the drop site.

y Indicates the y-coordinate of the pointer relative to the drop site.

dropSiteStatus

An IN/OUT member that indicates whether or not a drop site is valid.

When *reason* is **XmCR_DROP_SITE_ENTER_MESSAGE** or **XmCR_OPERATION_CHANGED_MESSAGE**, or *reason* is **XmCR_DRAG_MOTION_MESSAGE** or **XmCR_DROP_SITE_LEAVE_MESSAGE** and the pointer is not in the same drop site as on the previous invocation of the drag procedure, the toolkit initializes *dropSiteStatus* to **XmDROP_SITE_VALID** if the DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible and if the initial value of the *operation* member is not **XmDROP_NOOP**. Otherwise, the toolkit initializes *dropSiteStatus* to **XmDROP_SITE_INVALID**.

When the *reason* is **XmCR_DRAG_MOTION_MESSAGE** or **XmCR_DROP_SITE_LEAVE_MESSAGE** and the pointer is within the same drop site as on the previous invocation of the drag procedure, the toolkit initializes *dropSiteStatus* to the value of *dropSiteStatus* at the time the previous invocation of the drag procedure returns.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the *dropSiteStatus* member of the callback struct passed to the appropriate callbacks of the initiator.

operation An IN/OUT member that identifies an operation.

The toolkit initializes *operation* by selecting an operation from the bitwise AND of the initial value of the *operations* member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the *operation* member of the callback struct passed to the appropriate callbacks of the initiator.

operations An IN/OUT member that indicates the set of operations supported for the source data.

If the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource. If the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the *operations* member of the callback struct passed to the appropriate callbacks of the initiator.

animate An OUT member that indicates whether the toolkit or the receiver client provides drag-under effects for a valid drop site. If *animate* is set to True, the toolkit provides drop site animation per the **XmNanimationStyle** resource value; if it is set to False, the receiver generates drag-under animation effects.

A pointer to the following structure is passed to the **XmNdropProc** routine when the drop site receives a drop message:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Time            timeStamp;
    Widget          dragContext;
    Position        x;
    Position        y;
    unsigned char   dropSiteStatus;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropAction;
} XmDropProcCallbackStruct, *XmDropProcCallback;
```

reason Indicates why the callback was invoked.

event Specifies the **XEvent** that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

dragContext Specifies the ID of the DragContext widget associated with the transaction.

x Indicates the x-coordinate of the pointer relative to the drop site.

y Indicates the y-coordinate of the pointer relative to the drop site.

dropSiteStatus

An IN/OUT member that indicates whether or not a drop site is valid.

The toolkit initializes *dropSiteStatus* to **XmDROP_SITE_VALID** if the DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible and if the initial value of the *operation* member is not **XmDROP_NOOP**. Otherwise, the toolkit initializes *dropSiteStatus* to **XmDROP_SITE_INVALID**.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the *dropSiteStatus* member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

operation An IN/OUT member that identifies an operation.

The toolkit initializes *operation* by selecting an operation from the bitwise AND of the initial value of the *operations* member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for **XmDROP_MOVE**, then for **XmDROP_COPY**, then for **XmDROP_LINK**, and initializes *operation* to the first operation it finds in the set. If it finds none of these operations in the set, it initializes *operation* to **XmDROP_NOOP**.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the *operation* member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

operations An IN/OUT member that indicates the set of operations supported for the source data.

If the user does not select an operation (by pressing a modifier key), the toolkit initializes *operations* to the value of the DragContext's **XmNdragOperations** resource. If the user does select an operation, the toolkit initializes *operations* to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes *operations* to **XmDROP_NOOP**.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the *operations* member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

dropAction An IN/OUT member that identifies the action associated with the drop. The possible values are

XmDROP A drop was attempted. If the drop site is valid, drop transfer handling proceeds.

XmDROP_HELP
The user has requested help on the drop site.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the *dropAction* member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

Related Information

XmDragContext(3X), **XmDragIcon(3X)**, **XmDropSiteConfigureStackingOrder(3X)**, **XmDropSiteEndUpdate(3X)**, **XmDropSiteQueryStackingOrder(3)**, **XmDropSiteRegister(3X)**, **XmDropSiteStartUpdate(3X)**, **XmDropSiteUpdate(3X)**, **XmDropSiteUnregister(3X)**, **XmDropTransfer(3X)**, and **XmTargetsAreCompatible(3X)**.

XmDropSiteConfigureStackingOrder—A Drag and Drop function that reorders a stack of widgets that are registered drop sites

Synopsis

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

void XmDropSiteConfigureStackingOrder (widget, sibling, stack_mode)
        Widget      widget;
        Widget      sibling;
        Cardinal    stack_mode;
```

Description

XmDropSiteConfigureStackingOrder changes the stacking order of the drop site specified by *widget*. The stacking order controls the manner in which drag-under effects are clipped by overlapping siblings, regardless of whether they are active. The stack mode is relative either to the entire stack, or to another drop site within the stack. The stack order can be modified only if the drop sites are siblings in both the widget and drop site hierarchy, and the widget parent of the drop sites is registered as a composite drop site.

widget Specifies the drop site to be restacked.

sibling Specifies a sibling drop site for stacking operations. If specified, then *widget* is restacked relative to this drop site's stack position.

stack_mode Specifies the new stack position for the specified widget. The values are **XmABOVE** and **XmBELOW**. If a sibling is specified, then *widget* is restacked as follows:

XmABOVE The widget is placed just above the sibling.

XmBELOW The widget is placed just below the sibling.

If the *sibling* parameter is not specified, then *widget* is restacked as follows:

XmABOVE The widget is placed at the top of the stack.

XmBELOW The widget is placed at the bottom of the stack.

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Related Information

XmDropSite(3X), **XmDropSiteRetrieve(3X)**, and **XmDropSiteQueryStackingOrder(3X)**.

XmDropSiteEndUpdate—A Drag and Drop function that facilitates processing updates to multiple drop sites

Synopsis

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

void XmDropSiteEndUpdate (widget)
        Widget          widget;
```

Description

XmDropSiteEndUpdate is used in conjunction with **XmDropSiteStartUpdate** to process updates to multiple drop sites within the same hierarchy. **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** signal the beginning and the end respectively of a series of calls to **XmDropSiteUpdate**. Calls to **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** can be recursively stacked. Using these routines optimizes the processing of update information.

widget Specifies the ID of any widget within a given hierarchy. The function uses this widget to identify the shell that contains the drop sites.

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Related Information

XmDropSiteStartUpdate(3X) and **XmDropSiteUpdate(3X)**.

XmDropSiteQueryStackingOrder—A Drag and Drop function that returns the parent, a list of children, and the number of children for a specified widget

Synopsis

#include <Xm/DragDrop.h>

Status XmDropSiteQueryStackingOrder (*widget, parent_return, child_returns, num_child_returns*)

Widget	*widget*;
Widget	**parent_return*;
Widget	***child_returns*;
Cardinal	**num_child_returns*;

Description

XmDropSiteQueryStackingOrder obtains the parent, a list of children registered as drop sites, and the number of children registered as drop sites for a given widget. The children are listed in current stacking order, from bottom-most (first child) to the top-most (last child). This function allocates memory for the returned data that must be freed by calling **XtFree**.

widget Specifies the widget ID. For this widget, you obtain the list of its children, its parent, and the number of children.

parent_return
Returns the widget ID of the drop site parent of the specified widget.

child_returns
Returns a pointer to the list of drop site children associated with the specified widget.

num_child_returns
Returns the number of drop site children for the specified widget.

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Return Value

Returns 0 (zero) if the routine fails; returns a nonzero value if it succeeds.

Related Information

XmDropSite(3X) and **XmDropSiteConfigureStackingOrder(3X)**.

XmDropSiteRegister—A Drag and Drop function that identifies a drop site and assigns resources that specify its behavior

Synopsis

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

void XmDropSiteRegister (widget, arglist, argcount)
        Widget          widget;
        ArgList         arglist;
        Cardinal        argcount;
```

Description

XmDropSiteRegister identifies the specified widget or gadget as a drop site and sets resource values that define the drop site's behavior. The routine assigns default values to any resources that are not specified in the argument list. The toolkit generates a warning message if a drop site is registered with **XmNdropSiteActivity** set to **XmDROP_SITE_ACTIVE** and the **XmNdropProc** resource is NULL.

If the drop site is a descendant of a widget that is registered as a drop site, the **XmNdropSiteType** resource of the ancestor drop site must be specified as **XmDROP_SITE_COMPOSITE**. The ancestor must be registered before the descendant. The drop site is stacked above all other sibling drop sites already registered.

widget Specifies the ID of the widget to be registered.

arglist Specifies the argument list.

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Related Information

XmDisplay(3X), **XmDropSite(3X)**, **XmDropSiteEndUpdate(3X)**, **XmDropSiteStartUpdate(3X)**, **XmDropSiteUpdate(3X)**, **XmDropSiteUnregister(3X)**, and **XmScreen(3X)**.

XmDropSiteRetrieve—A Drag and Drop function that retrieves resource values set on a drop site

Synopsis

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

void XmDropSiteRetrieve (widget, arglist, argcount)
        Widget          widget;
        ArgList         arglist;
        Cardinal        argcount;
```

Description

XmDropSiteRetrieve extracts values for the given resources from the drop site specified by *widget*. An initiator can also obtain information about the current drop site by passing the associated DragContext widget as the *widget* parameter to this routine. The initiator can retrieve all of the drop site resources except **XmNdragProc** and **XmNdropProc** using this method.

widget Specifies the ID of the widget that encloses the drop site.

arglist Specifies the argument list.

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*).

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Related Information

XmDropSite(3X) and **XmDropSiteUpdate(3X)**.

XmDropSiteStartUpdate—A Drag and Drop function that facilitates processing updates to multiple drop sites

Synopsis

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

void XmDropSiteStartUpdate (widget)
        Widget          widget;
```

Description

XmDropSiteStartUpdate is used in conjunction with **XmDropSiteEndUpdate** to process updates to multiple drop sites within the same shell widget. **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** signal the beginning and the end respectively of a series of calls to **XmDropSiteUpdate**. Calls to **XmDropSiteStartUpdate** and **XmDropSiteEndUpdate** can be recursively stacked. Using these routines optimizes the processing of update information.

widget Specifies the ID of any widget within a given hierarchy. The function uses this widget to identify the shell that contains the drop sites.

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Related Information

XmDropSite(3X), **XmDropSiteEndUpdate(3X)**, and **XmDropSiteUpdate(3X)**.

XmDropSiteUnregister—A Drag and Drop function that frees drop site information

Synopsis

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

void XmDropSiteUnregister (widget)
        Widget          widget;
```

Description

XmDropSiteUnregister informs the toolkit that the specified widget is no longer a registered drop site. The function frees all associated drop site information.

widget Specifies the ID of the widget, registered as a drop site, that is to be unregistered

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Related Information

XmDropSite(3X) and **XmDropSiteRegister(3X)**.

XmDropSiteUpdate—A Drag and Drop function that sets resource values for a drop site

Synopsis

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

void XmDropSiteUpdate (widget, arglist, argcount)
        Widget          widget;
        ArgList         arglist;
        Cardinal        argcount;
```

Description

XmDropSiteUpdate modifies drop site resources associated with the specified widget. This routine updates the drop site resources specified in the *arglist.*

widget Specifies the ID of the widget registered as a drop site

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DropSite and its associated resources, see **XmDropSite(3X)**.

Related Information

XmDropSite(3X), **XmDropSiteEndUpdate(3X)**, **XmDropSiteRegister(3X)**, **XmDropSiteStartUpdate(3X)**, and **XmDropSiteUnregister(3X)**.

XmDropTransfer—The DropTransfer widget class

Synopsis

#include <Xm/DragDrop.h>

Description

DropTransfer provides a set of resources that identifies the procedures and associated information required by the toolkit in order to process and complete a drop transaction. Clients should not explicitly create a DropTransfer widget. Instead, a client initiates a transfer by calling **XmDropTransferStart**, which initializes and returns a DropTransfer widget. If this function is called within an **XmNdropProc** callback, the actual transfers are initiated after the callback returns. Even if no data needs to be transferred, **XmDropTransferStart** needs to be called (typically with no arguments, or just setting **XmNtransferStatus**) to finish the drag and drop transaction.

The **XmNdropTransfers** resource specifies a transfer list that describes the requested target types for the source data. A transfer list is an array of **XmDropTransferEntryRec** structures, each of which identifies a target type. The transfer list is analogous to the MULTIPLE selections capability defined in the *Inter-Client Communication Conventions Manual* (ICCCM).

The DropTransfer resource, **XmNtransferProc**, specifies a transfer procedure of type **XtSelectionCallbackProc** that delivers the requested selection data. This procedure operates in conjunction with the underlying Xt selection capabilities and is called for each target in the transfer list. Additional target types can be requested after a transfer is initiated by calling the **XmDropTransferAdd** function.

Structures

An **XmDropTransferEntry** is a pointer to the following structure of type **XmDropTransferEntryRec**, which identifies a selection target associated with a given drop transaction:

```
typedef struct
{
    XtPointer     client_data;
    Atom          target;
} XmDropTransferEntryRec, *XmDropTransferEntry;
```

client_data Specifies any additional information required by this selection target

target Specifies a selection target associated with the drop operation

Classes

DropTransfer inherits behavior and a resource from **Object**.

The class pointer is **xmDropTransferObjectClass**.

The class name is **XmDropTransfer**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmDropTransfer Resource Set

Name Class	Default Type	Access
XmNdropTransfers XmCDropTransfers	NULL XmDropTransferEntryRec *	CG
XmNincremental XmCIncremental	False Boolean	CSG
XmNnumDropTransfers XmCNumDropTransfers	0 Cardinal	CSG
XmNtransferProc XmCTransferProc	NULL XtSelectionCallbackProc	CSG
XmNtransferStatus XmCTransferStatus	XmTRANSFER_SUCCESS unsigned char	CSG

XmNdropTransfers

Specifies the address of an array of drop transfer entry records. The drop transfer is complete when all the entries in the list have been processed.

XmNincremental

Specifies a Boolean value that indicates whether the transfer on the receiver side uses the Xt incremental selection transfer mechanism described in *X Toolkit Intrinsics—C Language Interface*. If the value is True, the receiver uses incremental transfer; if the value is False, the receiver uses atomic transfer.

XmNnumDropTransfers

Specifies the number of entries in **XmNdropTransfers**. If this resource is set to 0 at any time, the transfer is considered complete. The value of **XmNtransferStatus** determines the completion handshaking process.

XmNtransferProc

Specifies a procedure of type **XtSelectionCallbackProc** that delivers the requested selection values. The *widget* argument passed to this procedure is the DropTransfer widget. The selection atom passed is _MOTIF_DROP. For additional information on selection callback procedures, see *X Toolkit Intrinsics—C Language Interface*.

XmNtransferStatus

Specifies the current status of the drop transfer. The client updates this value when the transfer ends and communicates the value to the initiator. The possible values are

XmTRANSFER_SUCCESS	The transfer succeeded.
XmTRANSFER_FAILURE	The transfer failed.

Inherited Resources

DropTransfer inherits behavior and a resource from **Object**. For a complete description of this resource, refer to the **Object** reference page.

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Related Information

Object(3X), **XmDisplay(3X)**, **XmDragContext(3X)**, **XmDragIcon(3X)**, **XmDropSite(3X)**, **XmDropTransferAdd(3X)**, and **XmDropTransferStart(3X)**.

XmDropTransferAdd—A Drag and Drop function that enables additional drop transfer entries to be processed after initiating a drop transfer

Synopsis

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

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

Description

XmDropTransferAdd identifies a list of additional drop transfer entries to be processed after a drop transfer is started.

drop_transfer
Specifies the ID of the DropTransfer widget returned by **XmDropTransferStart**

transfers Specifies the additional drop transfer entries that the receiver wants processed

num_transfers
Specifies the number of items in the *transfers* array

For a complete definition of DropTransfer and its associated resources, see **XmDropTransfer(3X)**.

Related Information

XmDragContext(3X), **XmDropTransfer(3X)**, and **XmDropTransferStart(3X)**.

XmDropTransferStart—A Drag and Drop function that initiates a drop transfer

Synopsis

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

Widget XmDropTransferStart (widget, arglist, argcount)
        Widget      widget;
        ArgList     arglist;
        Cardinal    argcount;
```

Description

XmDropTransferStart initiates a drop transfer and uses the specified argument list to initialize an **XmDropTransfer** object. The DropTransfer object can be manipulated with **XtSetValues** and **XtGetValues** until the last call to the **XmNtransferProc** procedure is made. After that point, the result of using the widget pointer is undefined. The DropTransfer object is freed by the toolkit when a transfer is complete.

widget Specifies the ID of the DragContext widget associated with the transaction

arglist Specifies the argument list

argcount Specifies the number of attribute/value pairs in the argument list (*arglist*)

For a complete definition of DropTransfer and its associated resources, see **XmDropTransfer(3X)**.

Return Value

Returns the ID of the DropTransfer widget.

Related Information

XmDragContext(3X), **XmDropTransfer(3X)**, and **XmDropTransferAdd(3X)**.

XmFileSelectionBox—The FileSelectionBox widget class

Synopsis

#include <Xm/FileSB.h>

Description

FileSelectionBox traverses through directories, views the files and subdirectories in them, and then selects files.

A FileSelectionBox has five main areas:

- A text input field for displaying and editing a directory mask used to select the files to be displayed
- A scrollable list of filenames
- A scrollable list of subdirectories
- A text input field for displaying and editing a filename
- A group of PushButtons, labeled **OK**, **Filter**, **Cancel**, and **Help**

Additional children may be added to the FileSelectionBox after creation. FileSelectionBox inherits the layout functionality provided by SelectionBox for any additional children. To remove the list of filenames, the list of subdirectories, or both from the FileSelectionBox after creation, unmanage the appropriate widgets and their labels. The list and label widgets are obtained through a call to the **XmFileSelectionBoxGetChild** function. To remove either the directory list or the file list, unmanage the parent of the appropriate list widget and unmanage the corresponding label.

The user can specify resources in a resource file for the automatically created widgets and gadgets of FileSelectionBox. The following list identifies the names of these widgets (or gadgets) and the associated FileSelectionBox areas:

Filter Text	**Text**
Directory List	**DirList**
Directory List Label	**Dir**

The directory mask is a string specifying the base directory to be examined and a search pattern. Ordinarily, the directory list displays the subdirectories of the base directory, as well as the base directory itself and its parent directory. The file list ordinarily displays all files and/or subdirectories in the base directory that match the search pattern.

A procedure specified by the **XmNqualifySearchDataProc** resource extracts the base directory and search pattern from the directory mask. If the directory specification is empty, the current working directory is used. If the search pattern is empty, a pattern that matches all files is used.

An application can supply its own **XmNqualifySearchDataProc** as well as its own procedures to search for subdirectories and files. The default **XmNqualifySearchDataProc** works as follows: The directory mask is a pathname that can contain zero or more *wildcard* characters in its directory portion, its file portion, or both. The directory components of the directory mask — up to, but not including, the first component with a wildcard character — specify the directory to be searched, relative to the current working directory. The remaining components specify the search pattern. If the directory mask is empty or if its first component contains a wildcard character, the current working directory is searched. If no component of the directory mask contains a wildcard character, the entire directory mask is the directory specification, and all files in that directory are matched.

The user can select a new directory to examine by scrolling through the list of directories and selecting the desired directory or by editing the directory mask. Selecting a new directory from the directory list does not change the search pattern. A user can select a new search pattern by editing the directory mask. Double clicking or pressing **KActivate** on a directory in the directory list initiates a search for files and subdirectories in the new directory, using the current search pattern.

The user can select a file by scrolling through the list of filenames and selecting the desired file or by entering the filename directly into the text edit area. Selecting a file from the list causes that filename to appear in the file selection text edit area.

The user may select a new file as many times as desired. The application is not notified until the user takes one of the following actions:

- Selects the **OK** PushButton
- Presses **KActivate** while the selection text edit area has the keyboard focus
- Double clicks or presses **KActivate** on an item in the file list

FileSelectionBox initiates a directory and file search when any of the following occurs:

- The FileSelectionBox is initialized
- The function **XtSetValues** is used to change **XmNdirMask**, **XmNdirectory**, **XmNpattern**, or **XmNfileTypeMask**

- The user activates the **Filter** PushButton
- The user double clicks or presses **KActivate** on an item in the directory list
- The application calls **XmFileSelectionDoSearch**
- The user presses **KActivate** while the directory mask text edit area has the keyboard focus

When a file search is initiated, the FileSelectionBox takes the following actions:

- Constructs an **XmFileSelectionBoxCallbackStruct** structure with values appropriate for the action that initiated the search
- Calls the **XmNqualifySearchDataProc** with the callback structure as the data input argument
- Sets **XmNdirectoryValid** and **XmNlistUpdated** to False
- Calls the **XmNdirSearchProc** with the qualified data returned by the **XmNqualifySearchDataProc**

If **XmNdirectoryValid** is True, the FileSelectionBox takes the following additional actions:

- Sets **XmNlistUpdated** to False
- Calls the **XmNfileSearchProc** with the qualified data returned by the **XmNqualifySearchDataProc** (and possibly modified by the **XmNdirSearchProc**)
- If **XmNlistUpdated** is True and the file list is empty, displays the **XmNnoMatchString** in the file list and clears the selection text and **XmNdirSpec**
- If **XmNlistUpdated** is True and the file list is not empty, sets the selection text and **XmNdirSpec** to the qualified *dir* returned by the **XmNqualifySearchDataProc** (and possibly modified by the **XmNdirSearchProc**)
- Sets the directory mask text and **XmNdirMask** to the qualified *mask* returned by the **XmNqualifySearchDataProc** (and possibly modified by the **XmNdirSearchProc**)

- Sets **XmNdirectory** to the qualified *dir* returned by the **XmNqualifySearchDataProc** (and possibly modified by the **XmNdirSearchProc**)
- Sets **XmNpattern** to the qualified *pattern* returned by the **XmNqualifySearchDataProc** (and possibly modified by the **XmNdirSearchProc**)

Classes

FileSelectionBox inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, **XmBulletinBoard**, and **XmSelectionBox**.

The class pointer is **xmFileSelectionBoxWidgetClass**.

The class name is **XmFileSelectionBox**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmFileSelectionBox Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdirectory XmCDirectory	dynamic XmString	CSG
XmNdirectoryValid XmCDirectoryValid	dynamic Boolean	SG
XmNdirListItems XmCDirListItems	dynamic XmStringTable	SG
XmNdirListItemCount XmCDirListItemCount	dynamic int	SG
XmNdirListLabelString XmCDirListLabelString	dynamic XmString	CSG
XmNdirMask XmCDirMask	dynamic XmString	CSG
XmNdirSearchProc XmCDirSearchProc	default procedure XmSearchProc	CSG
XmNdirSpec XmCDirSpec	dynamic XmString	CSG
XmNfileListItems XmCItems	dynamic XmStringTable	SG
XmNfileListItemCount XmCItemCount	dynamic int	SG
XmNfileListLabelString XmCFileListLabelString	dynamic XmString	CSG
XmNfileSearchProc XmCFileSearchProc	default procedure XmSearchProc	CSG
XmNfileTypeMask XmCFileTypeMask	XmFILE_REGULAR unsigned char	CSG
XmNfilterLabelString XmCFilterLabelString	dynamic XmString	CSG
XmNlistUpdated XmCListUpdated	dynamic Boolean	SG

Name Class	Default Type	Access
XmNnoMatchString XmCNoMatchString	" [ ] " XmString	CSG
XmNpattern XmCPattern	dynamic XmString	CSG
XmNqualifySearchDataProc XmCQualifySearchDataProc	default procedure XmQualifyProc	CSG

XmNdirectory

Specifies the base directory used in combination with **XmNpattern** in determining the files and directories to be displayed. The default value is determined by the **XmNqualifySearchDataProc** and depends on the initial values of **XmNdirMask**, **XmNdirectory**, and **XmNpattern**. If the default is NULL or empty, the current working directory is used.

XmNdirectoryValid

Specifies an attribute that is set only by the directory search procedure. The value is set to True if the directory passed to the directory search procedure can actually be searched. If this value is False the file search procedure is not called, and **XmNdirMask**, **XmNdirectory**, and **XmNpattern** are not changed.

XmNdirListItems

Specifies the items in the directory list. **XtGetValues** for this resource returns the list items themselves, not a copy of the list items. The application must not free the returned items.

XmNdirListItemCount

Specifies the number of items in the directory list. The value must not be negative.

XmNdirListLabelString

Specifies the label string of the directory list. The default for this resource depends on the locale. In the C locale the default is **Directories**.

XmNdirMask

Specifies the directory mask used in determining the files and directories to be displayed. The default value is determined by the **XmNqualifySearchDataProc** and depends on the initial values of **XmNdirMask**, **XmNdirectory**, and **XmNpattern**.

XmNdirSearchProc

Specifies a directory search procedure to replace the default directory search procedure. FileSelectionBox's default directory search procedure fulfills the needs of most applications. Because it is impossible to cover the requirements of all applications, you can replace the default search procedure.

The directory search procedure is called with two arguments: the FileSelectionBox widget and a pointer to an **XmFileSelectionBoxCallbackStruct** structure. The callback structure is generated by the **XmNqualifySearchDataProc** and contains all information required to conduct a directory search, including the directory mask and a qualified base directory and search pattern. Once called, it is up to the search routine to generate a new list of directories and update the FileSelectionBox widget by using **XtSetValues**.

The search procedure must set **XmNdirectoryValid** and **XmNlistUpdated**. If it generates a new list of directories, it must also set **XmNdirListItems** and **XmNdirListItemCount**.

If the search procedure cannot search the specified directory, it must warn the user and set **XmNdirectoryValid** and **XmNlistUpdated** to False, unless it prompts and subsequently obtains a valid directory. If the directory is valid but is the same as the current **XmNdirectory**, the search procedure must set **XmNdirectoryValid** to True, but it may elect not to generate a new list of directories. In this case, it must set **XmNlistUpdated** to False.

If the search procedure generates a new list of directories, it must set **XmNdirListItems** to the new list of directories and **XmNdirListItemCount** to the number of items in the list. If there are no directories, it sets **XmNdirListItems** to NULL and **XmNdirListItemCount** to 0 (zero). In either case, it must set **XmNdirectoryValid** and **XmNlistUpdated** to True.

The search procedure ordinarily should not change the callback structure. But if the original directory is not valid, the search procedure may obtain a new directory from the user. In this case it should set the *dir* member of the callback structure to the new

directory, call the **XmNqualifySearchDataProc** with the callback struct as the input argument, and copy the qualified data returned by the **XmNqualifySearchDataProc** into the callback struct.

XmNdirSpec Specifies the full file path specification. This is the **XmNtextString** resource in SelectionBox, renamed for FileSelectionBox. The default value is determined by the FileSelectionBox after conducting the initial directory and file search.

XmNfileListItems
Specifies the items in the file list. This is the **XmNlistItems** resource in SelectionBox, renamed for FileSelectionBox. **XtGetValues** for this resource returns the list items themselves, not a copy of the list items. The application must not free the returned items.

XmNfileListItemCount
Specifies the number of items in the file list. This is the **XmNlistItemCount** resource in SelectionBox, renamed for FileSelectionBox. The value must not be negative.

XmNfileListLabelString
Specifies the label string of the file list. This is the **XmNlistLabelString** resource in SelectionBox, renamed for FileSelectionBox. The default for this resource depends on the locale. In the C locale the default is **Files**.

XmNfileSearchProc
Specifies a file search procedure to replace the default file search procedure. FileSelectionBox's default file search procedure fulfills the needs of most applications. Because it is impossible to cover the requirements of all applications, you can replace the default search procedure.

The file search procedure is called with two arguments: the FileSelectionBox widget and a pointer to an **XmFileSelectionBoxCallbackStruct** structure. The callback structure is generated by the **XmNqualifySearchDataProc** (and possibly modified by the **XmNdirSearchProc**). It contains all information required to conduct a file search, including the directory mask and a qualified base directory and search pattern. Once this procedure is called, it is up to the search routine to generate a new list of files and update the FileSelectionBox widget by using **XtSetValues**.

The search procedure must set **XmNlistUpdated**. If it generates a new list of files, it must also set **XmNfileListItems** and **XmNfileListItemCount**.

It is recommended that the search procedure always generate a new list of files. If the *mask* member of the callback structure is the same as the *mask* member of the callback struct in the preceding call to the search procedure, the procedure may elect not to generate a new list of files. In this case it must set **XmNlistUpdated** to False.

If the search procedure generates a new list of files, it must set **XmNfileListItems** to the new list of files and **XmNfileListItemCount** to the number of items in the list. If there are no files, it sets **XmNfileListItems** to NULL and **XmNfileListItemCount** to 0 (zero). In either case it must set **XmNlistUpdated** to True.

In constructing the list of files, the search procedure should include only files of the types specified by the widget's **XmNfileTypeMask**.

Setting **XmNdirSpec** is optional, but recommended. Set this attribute to the full file specification of the directory searched. The directory specification is displayed below the directory and file lists.

XmNfileTypeMask

Specifies the type of files listed in the file list. Following are the possible values:

- **XmFILE_REGULAR** restricts the file list to contain only regular files.
- **XmFILE_DIRECTORY** restricts the file list to contain only directories.
- **XmFILE_ANY_TYPE** allows the list to contain all file types including directories.

XmNfilterLabelString

Specifies the label string for the text entry field for the directory mask. The default for this resource depends on the locale. In the C locale the default is **Filter**.

XmNlistUpdated

Specifies an attribute that is set only by the directory and file search procedures. Set to True if the search procedure updated the directory or file list.

XmNnoMatchString
Specifies a string to be displayed in the file list if the list of files is empty.

XmNpattern Specifies the search pattern used in combination with **XmNdirectory** in determining the files and directories to be displayed. The default value is determined by **XmNqualifySearchDataProc** and depends on the initial values of **XmNdirMask**, **XmNdirectory**, and **XmNpattern**. If the default is NULL or empty, a pattern that matches all files is used.

XmNqualifySearchDataProc
Specifies a search data qualification procedure to replace the default data qualification procedure. FileSelectionBox's default data qualification procedure fulfills the needs of most applications. Because it is impossible to cover the requirements of all applications, you can replace the default procedure.

The data qualification procedure is called to generate a qualified directory mask, base directory, and search pattern for use by the directory and file search procedures. It is called with three arguments: the FileSelectionBox widget and pointers to two **XmFileSelectionBoxCallbackStruct** structures. The first callback struct contains the input data. The second callback struct contains the output data, to be filled in by the data qualification procedure.

If the input *dir* and *pattern* members are not NULL, the procedure must copy them to the corresponding members of the output callback struct.

If the input *dir* is NULL, the procedure constructs the output *dir* as follows: If the input *mask* member is NULL, the procedure uses the widget's **XmNdirectory** as the output *dir*; otherwise, it extracts the output *dir* from the input *mask*. If the resulting output *dir* is empty, the procedure uses the current working directory instead.

If the input *pattern* is NULL, the procedure constructs the output *pattern* as follows: If the input *mask* member is NULL, the procedure uses the widget's **XmNpattern** as the output *pattern*; otherwise, it extracts the output *pattern* from the input *mask*. If the resulting output *pattern* is empty, the procedure uses a pattern that matches all files instead.

The data qualification procedure constructs the output *mask* from the output *dir* and *pattern*. The procedure must ensure that the output *dir*, *pattern*, and *mask* are fully qualified.

If the input *value* member is not NULL, the procedure must copy it to the output *value* member; otherwise, the procedure must copy the widget's **XmNdirSpec** to the output *value*.

The data qualification procedure must calculate the lengths of the output *value*, *mask*, *dir*, and *pattern* members and must fill in the corresponding length members of the output callback struct.

The data qualification procedure must copy the input *reason* and *event* members to the corresponding output members.

The values of the **XmNdirSearchProc** and **XmNfileSearchProc** are procedure pointers of type **XmSearchProc**, defined as follows:

void (* XmSearchProc) (*w, search_data*)
Widget *w*;
XtPointer *search_data*;

w The FileSelectionBox widget

search_data Pointer to an **XmFileSelectionBoxCallbackStruct** containing information for conducting a search

The value of the **XmNqualifySearchDataProc** resource is a procedure pointer of type **XmQualifyProc**, defined as follows:

void (* XmQualifyProc) (*w, input_data, output_data*)
Widget *w*;
XtPointer *input_data*;
XtPointer *output_data*;

w The FileSelectionBox widget

input_data Pointer to an **XmFileSelectionBoxCallbackStruct** containing input data to be qualified

output_data Pointer to an **XmFileSelectionBoxCallbackStruct** containing output data to be filled in by the qualification procedure

Inherited Resources

FileSelectionBox inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmSelectionBox Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNapplyCallback XmCCallback	NULL XtCallbackList	C
XmNapplyLabelString XmCApplyLabelString	dynamic XmString	CSG
XmNcancelCallback XmCCallback	NULL XtCallbackList	C
XmNcancelLabelString XmCCancelLabelString	dynamic XmString	CSG
XmNchildPlacement XmCChildPlacement	XmPLACE_ABOVE_SELECTION unsigned char	CSG
XmNdialogType XmCDialogType	XmDIALOG_FILE_SELECTION unsigned char	G
XmNhelpLabelString XmCHelpLabelString	dynamic XmString	CSG
XmNlistItemCount XmCItemCount	dynamic int	CSG
XmNlistItems XmCItems	dynamic XmStringTable	CSG
XmNlistLabelString XmCListLabelString	dynamic XmString	CSG
XmNlistVisibleItemCount XmCVisibleItemCount	dynamic int	CSG
XmNminimizeButtons XmCMinimizeButtons	False Boolean	CSG
XmNmustMatch XmCMustMatch	False Boolean	CSG
XmNnoMatchCallback XmCCallback	NULL XtCallbackList	C
XmNokCallback XmCCallback	NULL XtCallbackList	C

Name Class	Default Type	Access
XmNokLabelString XmCOkLabelString	dynamic XmString	CSG
XmNselectionLabelString XmCSelectionLabelString	dynamic XmString	CSG
XmNtextAccelerators XmCTextAccelerators	default XtAccelerators	C
XmNtextColumns XmCColumns	dynamic short	CSG
XmNtextString XmCTextString	dynamic XmString	CSG

XmBulletinBoard Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowOverlap XmCAllowOverlap	True Boolean	CSG
XmNautoUnmanage XmCAutoUnmanage	False Boolean	CG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNcancelButton XmCWidget	Cancel button Widget	SG
XmNdefaultButton XmCWidget	OK button Widget	SG
XmNdefaultPosition XmCDefaultPosition	True Boolean	CSG
XmNdialogStyle XmCDialogStyle	dynamic unsigned char	CSG
XmNdialogTitle XmCDialogTitle	NULL XmString	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	10 Dimension	CSG
XmNmarginWidth XmCMarginWidth	10 Dimension	CSG
XmNnoResize XmCNoResize	False Boolean	CSG
XmNresizePolicy XmCResizePolicy	XmRESIZE_ANY unsigned char	CSG

Name Class	Default Type	Access
XmNshadowType XmCShadowType	XmSHADOW_OUT unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNtextTranslations XmCTranslations	NULL XtTranslations	C
XmNunmapCallback XmCCallback	NULL XtCallbackList	C

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	dynamic Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	N/A
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    XmString    value;
    int         length;
    XmString    mask;
    int         mask_length;
    XmString    dir;
    int         dir_length;
    XmString    pattern;
    int         pattern_length;
} XmFileSelectionBoxCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

value Specifies the current value of **XmNdirSpec**

length Specifies the number of bytes in *value*

mask Specifies the current value of **XmNdirMask**

mask_length Specifies the number of bytes in *mask*

dir Specifies the current base directory

dir_length Specifies the number of bytes in *dir*

pattern Specifies the current search pattern

pattern_length
Specifies the number of bytes in *pattern*

Translations

XmFileSelectionBox inherits translations from **XmSelectionBox**.

Accelerators

The **XmNtextAccelerators** from **XmSelectionBox** are added to the selection and directory mask (filter) Text descendants of **XmFileSelectionBox**.

Action Routines

The **XmFileSelectionBox** action routines are

SelectionBoxUpOrDown(0|1|2|3):

If neither the selection text nor the directory mask (filter) text has the focus, this action does nothing.

If the selection text has the focus, the term *list* in the following description refers to the file list, and the term *text* refers to the selection text. If the directory mask text has the focus, *list* refers to the directory list, and *text* refers to the directory mask text.

When called with an argument of 0 (zero), this action selects the previous item in the list and replaces the text with that item.

When called with an argument of 1, this action selects the next item in the list and replaces the text with that item.

When called with an argument of 2, this action selects the first item in the list and replaces the text with that item.

When called with an argument of 3, this action selects the last item in the list and replaces the text with that item.

SelectionBoxRestore():

If neither the selection text nor the directory mask (filter) text has the focus, this action does nothing.

If the selection text has the focus, this action replaces the selection text with the selected item in the file list. If no item in the file list is selected, it clears the selection text.

If the directory mask text has the focus, this action replaces the directory mask text with a new directory mask constructed from the **XmNdirectory** and **XmNpattern** resources.

Additional Behavior

The FileSelectionBox widget has the following additional behavior:

MAny KCancel:
Calls the activate callbacks for the cancel button if it is sensitive. If no cancel button exists and the parent of the FileSelectionBox is a manager, it passes the event to the parent.

<KActivate> in Selection Text:
Calls the selection text widget's **XmNactivateCallback** callbacks. If **XmNmustMatch** is True and the selection text does not match an item in the file list, it calls the **XmNnoMatchCallback** callbacks with reason **XmCR_NO_MATCH**. Otherwise, it calls the **XmNokCallback** callbacks with reason **XmCR_OK**.

<KActivate> in Directory Mask Text:
Calls the directory mask text widget's **XmNactivateCallback** callbacks, initiates a directory and file search, and calls the **XmNapplyCallback** callbacks with reason **XmCR_APPLY**.

<DoubleClick> or **<KActivate>** in Directory List:
Calls the directory list widget's **XmNdefaultActionCallback** callbacks, initiates a directory and file search, and calls the **XmNapplyCallback** callbacks with reason **XmCR_APPLY**.

<DoubleClick> or **<KActivate>** in File List:
Calls the file list widget's **XmNdefaultActionCallback** callbacks and calls the **XmNokCallback** callbacks with reason **XmCR_OK**.

<Single Select> or **<Browse Select>** in Directory List:
Generates a new directory mask, using the selected list item as the directory and the pattern extracted from the current directory mask text as the search pattern. If the search pattern is empty, it uses a pattern that matches all files in the directory. Replaces the directory mask text with the new directory mask.

<Single Select> or **<Browse Select>** in File List:
Replaces the selection text with the selected list item.

<BTransfer> in File List:
Drags the content of one or more selected list items using the drag and drop facility. If **BTransfer** is pressed on an unselected item, drags only that item, excluding any other selected items.

The **XmNexportTargets** resource of the associated DragContext is set to target types of **COMPOUND_TEXT** and **FILE_NAME.** The **XmNclientData** resource is set to the index of the item in the list.

<BTransfer> in Directory List:
Drags the content of one or more selected list items using the drag and drop facility. If **BTransfer** is pressed on an unselected item, it drags only that item, excluding any other selected items.

The **XmNexportTargets** resource of the associated DragContext is set to target types of **COMPOUND_TEXT** and **FILE_NAME.** The **XmNclientData** resource is set to the index of the item in the list.

<Apply Button Activated>:
Initiates a directory and file search. Calls the **XmNapplyCallback** callbacks with reason **XmCR_APPLY.**

<OK Button Activated>:
If **XmNmustMatch** is True and the selection text does not match an item in the file list, calls the **XmNnoMatchCallback** callbacks with reason **XmCR_NO_MATCH.** Otherwise, it calls the **XmNokCallback** callbacks with reason **XmCR_OK.**

<Cancel Button Activated>:
Calls the **XmNcancelCallback** callbacks with reason **XmCR_CANCEL.**

<Help Button Activated>:
Calls the **XmNhelpCallback** callbacks with reason **XmCR_HELP.**

<KActivate>:
If no button, list widget, or text widget has the keyboard focus, if **XmNmustMatch** is True and the selection text does not match an item in the file list, it calls the **XmNnoMatchCallback** callbacks with reason **XmCR_NO_MATCH.** Otherwise, it calls the **XmNokCallback** callbacks with reason **XmCR_OK.**

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X).**

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmBulletinBoard(3X)**, **XmCreateFileSelectionBox(3X)**, **XmCreateFileSelectionDialog(3X)**, **XmFileSelectionBoxGetChild(3X)**, **XmFileSelectionDoSearch(3X)**, **XmManager(3X)**, and **XmSelectionBox(3X)**.

XmFileSelectionBoxGetChild—A FileSelectionBox function used to access a component

Synopsis

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

Widget XmFileSelectionBoxGetChild (widget, child)
        Widget          widget;
        unsigned char   child;
```

Description

XmFileSelectionBoxGetChild is used to access a component within a FileSelectionBox. The parameters given to the function are the FileSelectionBox widget and a value indicating which component to access.

widget Specifies the FileSelectionBox widget ID.

child Specifies a component within the FileSelectionBox. The following are legal values for this parameter:

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

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox(3X)**.

Return Value

Returns the widget ID of the specified FileSelectionBox component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmFileSelectionBox(3X).

XmFileSelectionDoSearch—A FileSelectionBox function that initiates a directory search

Synopsis

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

void XmFileSelectionDoSearch (widget, dirmask)
        Widget      widget;
        XmString    dirmask;
```

Description

XmFileSelectionDoSearch initiates a directory and file search in a FileSelectionBox widget. For a description of the actions that the FileSelectionBox takes when doing a search, see **XmFileSelectionBox(3X)**.

widget Specifies the FileSelectionBox widget ID.

dirmask Specifies the directory mask used in determining the directories and files displayed in the FileSelectionBox lists. This value is used as the *mask* member of the input data **XmFileSelectionBoxCallbackStruct** structure passed to the FileSelectionBox's **XmNqualifySearchDataProc**. The *dir* and *pattern* members of that structure are NULL.

For a complete definition of FileSelectionBox and its associated resources, see **XmFileSelectionBox(3X)**.

Related Information

XmFileSelectionBox(3X).

XmFontList—Data type for a font list

Synopsis

#include <Xm/Xm.h>

Description

XmFontList is the data type for a font list. A font list consists of font list entries. Each entry contains a font or a font set (a group of fonts) and is identified with a tag, which is optional. If this tag is NULL, the tag is set to **XmFONTLIST_DEFAULT_TAG**. **XmFONTLIST_DEFAULT_TAG** has a value of **XmFONTLIST_DEFAULT_TAG_STRING**.

When a compound string is displayed, the font list element tag of the compound string segment is matched with a font list entry tag in the font list and the matching font list entry is used to display the compound string. A font list entry is chosen as follows:

- The first font list entry whose tag matches the tag of the compound string segment is used.
- If no match has been found and if the tag of the compound string segment is **XmFONTLIST_DEFAULT_TAG**, the first font list entry whose tag matches the tag that would result from creating an entry with **XmSTRING_DEFAULT_CHARSET** is used. For example, if creating an entry with **XmSTRING_DEFAULT_CHARSET** would result in the tag **ISO8859-1**, the compound string segment tag **XmFONTLIST_DEFAULT_TAG** matches the font list entry tag **ISO8859-1**.
- If no match has been found and if the tag of the compound string segment matches the tag that would result from creating a segment with **XmSTRING_DEFAULT_CHARSET**, the first font list entry whose tag is **XmFONTLIST_DEFAULT_TAG** is used.
- If no match has been found, the first entry in the font list is used.

The font list interface consists of the routines listed in **Related Information**.

Font lists are specified in resource files with the following syntax:

resource_spec: *font_entry* [,*font_entry*]+

The resource value string consists of one or more font list entries separated by commas. Each *font_entry* identifies a font or font set and an optional font list entry tag. A tag specified for a single font follows the font name and is separated by = (equals sign); otherwise, in a font set the tag is separated by a colon. The colon is required whether a tag is specified or not. A font entry uses the following syntax to specify a single font:

font_name ['=' *tag*]

For example, the following entry specifies a 10 point Times Italic font without a font list entry tag;

```
*fontList:  -Adobe-Times-Medium-I-Normal--10*
```

A font entry containing a font set is similar, except a semicolon separates multiple font names and the specification ends with a colon followed by an optional tag:

font_name [';' *font_name*]+ ':' [*tag*]

A *font_name* is an X Logical Font Description (XLFD) string and *tag* is any set of characters from ISO646IRV except space, comma, colon, equal sign and semicolon. Following is an example of a font set entry. It consists of three fonts (except for charsets), and an explicit font list entry tag.

```
*fontList : -Adobe-Courier-Bold-R-Normal--25-180-100-100-M-150;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-240;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-120:MY_TAG
```

Related Information

XmFontListAdd(3X), **XmFontListAppendEntry(3X)**, **XmFontListCopy(3X)**, **XmFontListCreate(3X)**, **XmFontListEntryCreate(3X)**, **XmFontListEntryFree(3X)**, **XmFontListEntryGetFont(3X)**, **XmFontListEntryGetTag(3X)**, **XmFontListEntryLoad(3X)**, **XmFontListFree(3X)**, **XmFontListFreeFontContext(3X)**, **XmFontListGetNextFont(3X)**, **XmFontListInitFontContext(3X)**, **XmFontListNextEntry(3X)**, **XmFontListRemoveEntry(3X)**, and **XmString(3X)**.

XmFontListAdd—A font list function that creates a new font list

Synopsis

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

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

Description

XmFontListAdd creates a new font list consisting of the contents of *oldlist* and the new font list element being added. This function deallocates *oldlist* after extracting the required information; therefore, do not reference *oldlist* thereafter.

NOTE: This function is obsolete and exists for compatibility with previous releases. It has been replaced by **XmFontListAppendEntry**.

oldlist Specifies a pointer to the font list to which an entry will be added.

font Specifies a pointer to a font structure for which the new font list is generated. This is the structure returned by the XLib **XLoadQueryFont** function.

charset Specifies the character set identifier for the font. This can be **XmSTRING_DEFAULT_CHARSET**, but this value does not comply with the AES, and it may be removed in future versions of Motif. If the value is **XmSTRING_DEFAULT_CHARSET**, the routine derives the character set from the current language environment.

Return Value

Returns NULL if *oldlist* is NULL; returns *oldlist* if *font* or *charset* is NULL; otherwise, returns a new font list.

Related Information

XmFontList(3X) and **XmFontListAppendEntry(3X)**.

XmFontListAppendEntry—A font list function that appends an entry to a font list

Synopsis

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

XmFontList XmFontListAppendEntry (oldlist, entry)
        XmFontList      oldlist;
        XmFontListEntry entry;
```

Description

XmFontListAppendEntry creates a new font list that contains the contents of *oldlist*. This function copies the contents of the font list entry being added into this new font list. If *oldlist* is NULL, **XmFontListAppendEntry** creates a new font list containing only the single entry specified.

This function deallocates the original font list after extracting the required information. The caller must free the font list entry by using **XmFontListEntryFree**.

oldlist	Specifies the font list to be added to
entry	Specifies the font list entry to be added

Return Value

If *entry* is NULL, returns *oldlist*; otherwise, returns a new font list.

Related Information

XmFontList(3X), **XmFontListEntryCreate(3X)**, **XmFontListEntryFree(3X)**, **XmFontListEntryLoad(3X)**, **XmFontListFree(3X)**, and **XmFontListRemoveEntry(3X).**

XmFontListCopy—A font list function that copies a font list

Synopsis

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

XmFontList XmFontListCopy (fontlist)
        XmFontList      fontlist;
```

Description

XmFontListCopy creates a new font list consisting of the contents of the *fontlist* argument.

fontlist Specifies a font list to be copied

Return Value

Returns NULL if *fontlist* is NULL; otherwise, returns a new font list.

Related Information

XmFontList(3X) and **XmFontListFree(3X).**

XmFontListCreate—A font list function that creates a font list

Synopsis

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

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

Description

XmFontListCreate creates a new font list with a single element specified by the provided font and character set. It also allocates the space for the font list.

NOTE: This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmFontListAppendEntry**.

font Specifies a pointer to a font structure for which the new font list is generated. This is the structure returned by the XLib **XLoadQueryFont** function.

charset Specifies the character set identifier for the font. This can be **XmSTRING_DEFAULT_CHARSET**, but this value does not comply with the AES, and it may be removed in future versions of Motif. If the value is **XmSTRING_DEFAULT_CHARSET**, the routine derives the character set from the current language environment.

Return Value

Returns NULL if *font* or *charset* is NULL; otherwise, returns a new font list.

Related Information

XmFontList(3X) and **XmFontListAppendEntry(3X)**.

XmFontListEntryCreate—A font list function that creates a font list entry

Synopsis

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

XmFontListEntry XmFontListEntryCreate (tag, type, font)
        char        *tag;
        XmFontType  type;
        XtPointer   font;
```

Description

XmFontListEntryCreate creates a font list entry that contains either a font or font set and is identified by a tag.

tag Specifies a NULL-terminated string for the tag of the font list entry. The tag may be specified as **XmFONTLIST_DEFAULT_TAG**, which is used to identify the default font list element in a font list.

type Specifies whether the *font* argument is a font structure or a font set. Valid values are **XmFONT_IS_FONT** and **XmFONT_IS_FONTSET**.

font Specifies either an **XFontSet** returned by **XCreateFontSet** or a pointer to an **XFontStruct** returned by **XLoadQueryFont**.

The toolkit does not copy the X Font structure specified by the *font* argument. Therefore, an application programmer must not free **XFontStruct** or **XFontSet** until all font lists and/or font entries that reference it have been freed.

Return Value

Returns a font list entry.

Related Information

XmFontList(3X), **XmFontListAppendEntry(3X)**, **XmFontListEntryFree(3X)**, **XmFontListEntryGetFont(3X)**, **XmFontListEntryGetTag(3X)**, **XmFontListEntryLoad(3X)**, and **XmFontListRemoveEntry(3X)**.

XmFontListEntryFree—A font list function that recovers memory used by a font list entry

Synopsis

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

void XmFontListEntryFree (entry)
        XmFontListEntry *entry;
```

Description

XmFontListEntryFree recovers memory used by a font list entry. This routine does not free the **XFontSet** or **XFontStruct** associated with the font list entry.

entry Specifies the font list entry to be freed

Related Information

XmFontList(3X), **XmFontListAppendEntry(3X)**, **XmFontListEntryCreate(3X)**, **XmFontListEntryLoad(3X)**, **XmFontListNextEntry(3X)**, and **XmFontListRemoveEntry(3X)**.

XmFontListEntryGetFont—A font list function that retrieves font information from a font list entry

Synopsis

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

XtPointer XmFontListEntryGetFont (entry, type_return)
        XmFontListEntry entry;
        XmFontType      *type_return;
```

Description

XmFontListEntryGetFont retrieves font information for a specified font list entry. If the font list entry contains a font, *type_return* returns **XmFONT_IS_FONT** and the function returns a pointer to an **XFontStruct**. If the font list entry contains a font set, *type_return* returns **XmFONT_IS_FONTSET** and the function returns the **XFontSet**.

entry Specifies the font list entry.

type_return Specifies a pointer to the type of the font element for the current entry. Valid values are **XmFONT_IS_FONT** and **XmFONT_IS_FONTSET**.

The returned **XFontSet** or **XFontStruct** is not a copy of the toolkit data and must not be freed.

Return Value

Returns an **XFontSet** or a pointer to an **XFontStruct** structure.

Related Information

XmFontList(3X), **XmFontListEntryCreate(3X)**, **XmFontListEntryGetTag(3X)** **XmFontListEntryLoad(3X)**, and **XmFontListNextEntry(3X)**.

XmFontListEntryGetTag—A font list function that retrieves the tag of a font list entry

Synopsis

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

char* XmFontListEntryGetTag (entry)
        XmFontListEntry entry;
```

Description

XmFontListEntryGetTag retrieves a copy of the tag of the specified font list entry. This routine allocates memory for the tag string that must be freed by the application.

entry Specifies the font list entry

Return Value

Returns the tag for the font list entry.

Related Information

XmFontList(3X), **XmFontListEntryCreate(3X)**, **XmFontListEntryGetFont(3X)**, **XmFontListEntryLoad(3X)**, and **XmFontListNextEntry(3X)**.

XmFontListEntryLoad—A font list function that loads a font or creates a font set and creates an accompanying font list entry

Synopsis

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

XmFontListEntry XmFontListEntryLoad (display, font_name, type, tag)
        Display     *display;
        char        *font_name;
        XmFontType  type;
        char        *tag;
```

Description

XmFontListEntryLoad loads a font or creates a font set based on the value of the *type* argument. It creates and returns a font list entry that contains the font or font set and the specified tag.

If the value of *type* is **XmFONT_IS_FONT**, the function uses the **XtCvtStringToFontStruct** routine to convert the value of *font_name* to a font struct. If the value of *type* is **XmFONT_IS_FONTSET**, the function uses the **XtCvtStringToFontSet** converter to create a font set in the current locale. **XmFontListEntryLoad** creates a font list entry that contains the font or font set derived from the converter. For more information about **XtCvtStringToFontStruct** and **XtCvtStringToFontSet**, see *X Toolkit Intrinsics—C Language Interface*.

display Specifies the display where the font list will be used.

font_name

Specifies an X Logical Font Description (XLFD) string, which is interpreted either as a font name or as a base font name list. A base font name list is a comma-separated and NULL-terminated string.

type Specifies whether the *font_name* argument refers to a font name or to a base font name list. Valid values are **XmFONT_IS_FONT** and **XmFONT_IS_FONTSET**.

tag Specifies the tag of the font list entry to be created. The tag may be specified as **XmFONTLIST_DEFAULT_TAG**, which is used to identify the default font list element in a font list when specified as part of a resource.

Return Value

If the specified font is not found, or the specified font set cannot be created, returns NULL; otherwise, returns a font list entry.

Related Information

XmFontList(3X), **XmFontListAppendEntry(3X)**, **XmFontListEntryCreate(3X)**, **XmFontListEntryFree(3X)**, **XmFontListEntryGetFont(3X)**, **XmFontListEntryGetTag(3X)**, and **XmFontListRemoveEntry(3X)**.

XmFontListFree—A font list function that recovers memory used by a font list

Synopsis

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

void XmFontListFree (list)
        XmFontList    list;
```

Description

XmFontListFree recovers memory used by a font list. This routine does not free the XFontSet or XFontStruct associated with the specified font list.

list Specifies the font list to be freed

Related Information

XmFontList(3X), **XmFontListAppendEntry(3X)**, **XmFontListCopy(3X)**, and **XmFontListRemoveEntry(3X)**.

XmFontListFreeFontContext—A font list function that instructs the toolkit that the font list context is no longer needed

Synopsis

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

void XmFontListFreeFontContext (context)
        XmFontContext    context;
```

Description

XmFontListFreeFontContext instructs the toolkit that the context is no longer needed and will not be used without reinitialization.

context Specifies the font list context structure that was allocated by the **XmFontListInitFontContext** function

Related Information

XmFontListInitFontContext(3X) and **XmFontListNextEntry(3X)**.

XmFontListGetNextFont—A font list function that allows applications to access the fonts and character sets in a font list

Synopsis

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

Boolean XmFontListGetNextFont (context, charset, font)
        XmFontContext    context;
        XmStringCharSet *charset;
        XFontStruct     **font;
```

Description

XmFontListGetNextFont accesses the character set and font for the next entry of the font list. The application first uses the **XmFontListInitFontContext** routine to create a font list context. The application then calls **XmFontListGetNextFont** repeatedly with the same context. Each succeeding call accesses the next element of the font list. When finished, the application calls **XmFontListFreeFontContext** to free the allocated font list context.

This routine allocates memory for the character set string that must be freed by the application.

This function is obsolete and exists for compatibility with previous releases. It is replaced by **XmFontListNextEntry**. If **XmFontListGetNextFont** is passed a context that contains a font set entry, it will return the first font of the font set. The next call to the function will move to the next entry in the font list.

context Specifies the font list context

charset Specifies a pointer to a character set string; the routine returns the character set for the current font list element

font Specifies a pointer to a pointer to a font structure; the routine returns the font for the current font list element

Return Value

Returns True if the returned values are valid; otherwise, returns False.

Related Information

XmFontList(3X) and **XmFontListNextEntry(3X)**.

XmFontListInitFontContext—A font list function that allows applications to access the entries in a font list

Synopsis

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

Boolean XmFontListInitFontContext (context, fontlist)
        XmFontContext   *context;
        XmFontList      fontlist;
```

Description

XmFontListInitFontContext establishes a context to allow applications to access the contents of a font list. This context is used when reading the font list entry tag, font, or font set associated with each entry in the font list. A Boolean status is returned to indicate whether or not the font list is valid.

context Specifies a pointer to the allocated context

fontlist Specifies the font list

Return Value

Returns True if the context was allocated; otherwise, returns False.

Related Information

XmFontList(3X), **XmFontListFreeFontContext(3X)**, and **XmFontListNextEntry(3X)**.

XmFontListNextEntry—A font list function that returns the next entry in a font list

Synopsis

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

XmFontListEntry XmFontListNextEntry (context)
        XmFontContext   context;
```

Description

XmFontListNextEntry returns the next entry in the font list. The application uses the **XmFontListInitFontContext** routine to create a font list context. The first call to **XmFontListNextEntry** sets the context to the first entry in the font list. The application then calls **XmFontListNextEntry** repeatedly with the same context. Each succeeding call accesses the next entry of the font list. When finished, the application calls **XmFontListFreeFontContext** to free the allocated font list context.

context Specifies the font list context

Return Value

Returns NULL if the context does not refer to a valid entry or if it is at the end of the font list; otherwise, it returns a font list entry.

Related Information

XmFontList(3X), **XmFontListEntryFree(3X)**, **XmFontListEntryGetFont(3X)**, **XmFontListEntryGetTag(3X)**, **XmFontListFreeFontContext(3X)**, and **XmFontListInitFontContext(3X)**.

XmFontListRemoveEntry—A font list function that removes a font list entry from a font list

Synopsis

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

XmFontList XmFontListRemoveEntry (oldlist, entry)
        XmFontList      oldlist;
        XmFontListEntry entry;
```

Description

XmFontListRemoveEntry creates a new font list that contains the contents of *oldlist* minus those entries specified in *entry*. The routine removes any entries from *oldlist* that match the components (tag, type font/font set) of the specified entry. The function deallocates the original font list after extracting the required information. The caller uses **XmFontListEntryFree** to recover memory allocated for the specified entry. This routine does not free the **XFontSet** or **XFontStruct** associated with the font list entry that is removed.

oldlist	Specifies the font list
entry	Specifies the font list entry to be removed

Return Value

If *oldlist* is NULL, the function returns NULL. If *entry* is NULL or no entries are removed, the function returns *oldlist*. Otherwise, it returns a new font list.

Related Information

XmFontList(3X), **XmFontListAppendEntry(3X)**, **XmFontListEntryCreate(3X)**, **XmFontListEntryFree(3X)**, **XmFontListEntryLoad(3X)**, and **XmFontListFree(3X)**.

XmForm—The Form widget class

Synopsis

#include <Xm/Form.h>

Description

Form is a container widget with no input semantics of its own. Constraints are placed on children of the Form to define attachments for each of the child's four sides. These attachments can be to the Form, to another child widget or gadget, to a relative position within the Form, or to the initial position of the child. The attachments determine the layout behavior of the Form when resizing occurs.

The default value for **XmNinitialFocus** is the value of **XmNdefaultButton**.

Following are some important considerations in using a Form:

- Every child must have an attachment on either the left or the right. If initialization or **XtSetValues** leaves a widget without such an attachment, the result depends upon the value of **XmNrubberPositioning**.

  If **XmNrubberPositioning** is False, the child is given an **XmNleftAttachment** of **XmATTACH_FORM** and an **XmNleftOffset** equal to its current *x* value.

  If **XmNrubberPositioning** is True, the child is given an **XmNleftAttachment** of **XmATTACH_POSITION** and an **XmNleftPosition** proportional to the current *x* value divided by the width of the Form.

  In either case, if the child has not been previously given an *x* value, its *x* value is taken to be 0 (zero), which places the child at the left side of the Form.

- If you want to create a child without any attachments, and then later (for example, after creating and managing it, but before realizing it) give it a right attachment through **XtSetValues**, you must set its **XmNleftAttachment** to **XmATTACH_NONE** at the same time.

- The **XmNresizable** resource controls only whether a geometry request by the child will be granted. It has no effect on whether the child's size can be changed because of changes in geometry of the Form or of other children.

- Every child has a preferred width, based on geometry requests it makes (whether they are granted or not).

- If a child has attachments on both the left and the right sides, its size is completely controlled by the Form. It can be shrunk below its preferred width or enlarged above it, if necessary, due to other constraints. In addition, the child's geometry requests to change its own width may be refused.
- If a child has attachments on only its left or right side, it will always be at its preferred width (if resizable, otherwise at is current width). This may cause it to be clipped by the Form or by other children.
- If a child's left (or right) attachment is set to **XmATTACH_SELF**, its corresponding left (or right) offset is forced to 0 (zero). The attachment is then changed to **XmATTACH_POSITION**, with a position that corresponds to the *x* value of the child's left (or right) edge. To fix the position of a side at a specific *x* value, use **XmATTACH_FORM** or **XmATTACH_OPPOSITE_FORM** with the *x* value as the left (or right) offset.
- Unmapping a child has no effect on the Form except that the child is not mapped.
- Unmanaging a child unmaps it. If no other child is attached to it, or if all children attached to it and all children recursively attached to them are also all unmanaged, all of those children are treated as if they did not exist in determining the size of the Form.
- When using **XtSetValues** to change the **XmNx** resource of a child, you must simultaneously set its left attachment to either **XmATTACH_SELF** or **XmATTACH_NONE**. Otherwise, the request is not granted. If **XmNresizable** is False, the request is granted only if the child's size can remain the same.
- A left (or right) attachment of **XmATTACH_WIDGET**, where **XmNleftWidget** (or **XmNrightWidget**) is NULL, acts like an attachment of **XmATTACH_FORM**.
- If an attachment is made to a widget that is not a child of the Form, but an ancestor of the widget is a child of the Form, the attachment is made to the ancestor.

All these considerations are true of top and bottom attachments as well, with top acting like left, bottom acting like right, *y* acting like *x*, and height acting like width.

Classes

Form inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XmBulletinBoard**.

The class pointer is **xmFormWidgetClass**.

The class name is **XmForm**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmForm Resource Set

Name Class	Default Type	Access
XmNfractionBase XmCMaxValue	100 int	CSG
XmNhorizontalSpacing XmCSpacing	0 Dimension	CSG
XmNrubberPositioning XmCRubberPositioning	False Boolean	CSG
XmNverticalSpacing XmCSpacing	0 Dimension	CSG

XmNfractionBase

Specifies the denominator used in calculating the relative position of a child widget using **XmATTACH_POSITION** constraints. The value must not be 0 (zero).

If the value of a child's **XmNleftAttachment** (or **XmNrightAttachment**) is **XmATTACH_POSITION**, the position of the left (or right) side of the child is relative to the left side of the Form and is a fraction of the width of the Form. This fraction is the value of the child's **XmNleftPosition** (or **XmNrightPosition**) resource divided by the value of the Form's **XmNfractionBase**.

If the value of a child's **XmNtopAttachment** (or

XmNbottomAttachment) is **XmATTACH_POSITION**, the position of the top (or bottom) side of the child is relative to the top side
of the Form and is a fraction of the height of the Form. This fraction is the value of the child's **XmNtopPosition** (or **XmNbottomPosition**) resource divided by the value of the Form's **XmNfractionBase**.

XmNhorizontalSpacing

Specifies the offset for right and left attachments. This resource is only used if no offset resource is specified (when attaching to a widget), or if no margin resource is specified (when attaching to the Form).

XmNrubberPositioning

Indicates the default near (left) and top attachments for a child of the Form. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.)

The default left attachment is applied whenever initialization or **XtSetValues** leaves the child without either a left or right attachment. The default top attachment is applied whenever initialization or **XtSetValues** leaves the child without either a top or bottom attachment.

If this Boolean resource is set to False, **XmNleftAttachment** and **XmNtopAttachment** default to **XmATTACH_FORM**, **XmNleftOffset** defaults to the current *x* value of the left side of the child, and **XmNtopOffset** defaults to the current *y* value of the child. The effect is to position the child according to its absolute distance from the left or top side of the Form.

If this resource is set to True, **XmNleftAttachment** and **XmNtopAttachment** default to **XmATTACH_POSITION**, **XmNleftPosition** defaults to a value proportional to the current *x* value of the left side of the child divided by the width of the Form, and **XmNtopPosition** defaults to a value proportional to the current *y* value of the child divided by the height of the Form. The effect is to position the child relative to the left or top side of the Form and in proportion to the width or height of the Form.

XmNverticalSpacing

Specifies the offset for top and bottom attachments. This resource is only used if no offset resource is specified (when attaching to a widget), or if no margin resource is specified (when attaching to the Form).

XmForm Constraint Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomAttachment XmCAttachment	XmATTACH_NONE unsigned char	CSG
XmNbottomOffset XmCOffset	0 int	CSG
XmNbottomPosition XmCAttachment	0 int	CSG
XmNbottomWidget XmCWidget	NULL Widget	CSG
XmNleftAttachment XmCAttachment	XmATTACH_NONE unsigned char	CSG
XmNleftOffset XmCOffset	0 int	CSG
XmNleftPosition XmCAttachment	0 int	CSG
XmNleftWidget XmCWidget	NULL Widget	CSG
XmNresizable XmCBoolean	True Boolean	CSG
XmNrightAttachment XmCAttachment	XmATTACH_NONE unsigned char	CSG
XmNrightOffset XmCOffset	0 int	CSG
XmNrightPosition XmCAttachment	0 int	CSG
XmNrightWidget XmCWidget	NULL Widget	CSG
XmNtopAttachment XmCAttachment	XmATTACH_NONE unsigned char	CSG

Name Class	Default Type	Access
XmNtopOffset XmCOffset	0 int	CSG
XmNtopPosition XmCAttachment	0 int	CSG
XmNtopWidget XmCWidget	NULL Widget	CSG

XmNbottomAttachment

Specifies attachment of the bottom side of the child. It can have the following values:

XmATTACH_NONE

Do not attach the bottom side of the child.

Attach the bottom side of the child to the bottom side of the Form.

XmATTACH_OPPOSITE_FORM

Attach the bottom side of the child to the top side of the Form. **XmNbottomOffset** can be used to determine the visibility of the child.

XmATTACH_WIDGET

Attach the bottom side of the child to the top side of the widget or gadget specified in the **XmNbottomWidget** resource. If **XmNbottomWidget** is NULL, **XmATTACH_WIDGET** is replaced by **XmATTACH_FORM**, and the child is attached to the bottom side of the Form.

XmATTACH_OPPOSITE_WIDGET

Attach the bottom side of the child to the bottom side of the widget or gadget specified in the **XmNbottomWidget** resource.

XmATTACH_POSITION

Attach the bottom side of the child to a position that is relative to the top side of the Form and in proportion to the height of the Form. This position is determined by the **XmNbottomPosition** and **XmNfractionBase** resources.

XmATTACH_SELF
Attach the bottom side of the child to a position that is proportional to the current *y* value of the bottom of the child divided by the height of the Form. This position is determined by the **XmNbottomPosition** and **XmNfractionBase** resources. **XmNbottomPosition** is set to a value proportional to the current *y* value of the bottom of the child divided by the height of the Form.

XmNbottomOffset
Specifies the constant offset between the bottom side of the child and the object to which it is attached. The relationship established remains, regardless of any resizing operations that occur. When this resource is explicitly set, the value of **XmNverticalSpacing** is ignored.

XmNbottomPosition
This resource is used to determine the position of the bottom side of the child when the child's **XmNbottomAttachment** is set to **XmATTACH_POSITION**. In this case the position of the bottom side of the child is relative to the top side of the Form and is a fraction of the height of the Form. This fraction is the value of the child's **XmNbottomPosition** resource divided by the value of the Form's **XmNfractionBase**. For example, if the child's **XmNbottomPosition** is 50, the Form's **XmNfractionBase** is 100, and the Form's height is 200, the position of the bottom side of the child is 100.

XmNbottomWidget
Specifies the widget or gadget to which the bottom side of the child is attached. This resource is used if the **XmNbottomAttachment** resource is set to either **XmATTACH_WIDGET** or **XmATTACH_OPPOSITE_WIDGET**.

A string-to-widget resource converter is automatically installed for use with this resource. With this converter, the widget that is to be the value of the resource must exist at the time the widget that has the resource is created.

XmNleftAttachment

Specifies attachment of the near (left) side of the child. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) It can have the following values:

XmATTACH_NONE

Do not attach the left side of the child. If **XmNrightAttachment** is also **XmATTACH_NONE**, this value is ignored and the child is given a default left attachment.

XmATTACH_FORM

Attach the left side of the child to the left side of the Form.

XmATTACH_OPPOSITE_FORM

Attach the left side of the child to the right side of the Form. **XmNleftOffset** can be used to determine the visibility of the child.

XmATTACH_WIDGET

Attach the left side of the child to the right side of the widget or gadget specified in the **XmNleftWidget** resource. If **XmNleftWidget** is NULL, **XmATTACH_WIDGET** is replaced by **XmATTACH_FORM**, and the child is attached to the left side of the Form.

XmATTACH_OPPOSITE_WIDGET

Attach the left side of the child to the left side of the widget or gadget specified in the **XmNleftWidget** resource.

XmATTACH_POSITION

Attach the left side of the child to a position that is relative to the left side of the Form and in proportion to the width of the Form. This position is determined by the **XmNleftPosition** and **XmNfractionBase** resources.

XmATTACH_SELF
Attach the left side of the child to a position that is proportional to the current *x* value of the left side of the child divided by the width of the Form. This position is determined by the **XmNleftPosition** and **XmNfractionBase** resources. **XmNleftPosition** is set to a value proportional to the current *x* value of the left side of the child divided by the width of the Form.

XmNleftOffset
Specifies the constant offset between the near (left) side of the child and the object to which it is attached. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The relationship established remains, regardless of any resizing operations that occur. When this resource is explicitly set, the value of **XmNhorizontalSpacing** is ignored.

XmNleftPosition
This resource is used to determine the position of the near (left) side of the child when the child's **XmNleftAttachment** is set to **XmATTACH_POSITION**. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.)

In this case, the position of the left side of the child is relative to the left side of the Form and is a fraction of the width of the Form. This fraction is the value of the child's **XmNleftPosition** resource divided by the value of the Form's **XmNfractionBase**. For example, if the child's **XmNleftPosition** is 50, the Form's **XmNfractionBase** is 100, and the Form's width is 200, the position of the left side of the child is 100.

XmNleftWidget
Specifies the widget or gadget to which the near (left) side of the child is attached. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The **XmNleftWidget** resource is used if the **XmNleftAttachment** resource is set to either **XmATTACH_WIDGET** or **XmATTACH_OPPOSITE_WIDGET**.

A string-to-widget resource converter is automatically installed for use with this resource. With this converter, the widget that is to be the value of the resource must exist at the time the widget that has the resource is created.

XmNresizable

This Boolean resource specifies whether or not a child's request for a new size is (conditionally) granted by the Form. If this resource is set to True the request is granted if possible. If this resource is set to False the request is always refused.

If a child has both left and right attachments, its width is completely controlled by the Form, regardless of the value of the child's **XmNresizable** resource. If a child has a left or right attachment but not both, the child's **XmNwidth** is used in setting its width if the value of the child's **XmNresizable** resource is True. These conditions are also true for top and bottom attachments, with height acting like width.

XmNrightAttachment

Specifies attachment of the far (right) side of the child. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) It can have the following values:

XmATTACH_NONE

Do not attach the right side of the child.

XmATTACH_FORM

Attach the right side of the child to the right side of the Form.

XmATTACH_OPPOSITE_FORM

Attach the right side of the child to the left side of the Form. **XmNrightOffset** can be used to determine the visibility of the child.

XmATTACH_WIDGET

Attach the right side of the child to the left side of the widget or gadget specified in the **XmNrightWidget** resource. If **XmNrightWidget** is NULL, **XmATTACH_WIDGET** is replaced by **XmATTACH_FORM**, and the child is attached to the right side of the Form.

XmATTACH_OPPOSITE_WIDGET
Attach the right side of the child to the right side of the widget or gadget specified in the **XmNrightWidget** resource.

XmATTACH_POSITION
Attach the right side of the child to a position that is relative to the left side of the Form and in proportion to the width of the Form. This position is determined by the **XmNrightPosition** and **XmNfractionBase** resources.

XmATTACH_SELF
Attach the right side of the child to a position that is proportional to the current *x* value of the right side of the child divided by the width of the Form. This position is determined by the **XmNrightPosition** and **XmNfractionBase** resources. **XmNrightPosition** is set to a value proportional to the current *x* value of the right side of the child divided by the width of the Form.

XmNrightOffset
Specifies the constant offset between the far (right) side of the child and the object to which it is attached. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The relationship established remains, regardless of any resizing operations that occur. When this resource is explicitly set, the value of **XmNhorizontalSpacing** is ignored.

XmNrightPosition
This resource is used to determine the position of the far (right) side of the child when the child's **XmNrightAttachment** is set to **XmATTACH_POSITION**. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.)

In this case the position of the right side of the child is relative to the left side of the Form and is a fraction of the width of the Form. This fraction is the value of the child's **XmNrightPosition** resource divided by

the value of the Form's **XmNfractionBase**. For example, if the child's **XmNrightPosition** is 50, the Form's **XmNfractionBase** is 100, and the Form's width is 200, the position of the right side of the child is 100.

XmNrightWidget

Specifies the widget or gadget to which the far (right) side of the child is attached. (**Note:** Whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The **XmNrightWidget** resource is used if the **XmNrightAttachment** resource is set to either **XmATTACH_WIDGET** or **XmATTACH_OPPOSITE_WIDGET**.

A string-to-widget resource converter is automatically installed for use with this resource. With this converter, the widget that is to be the value of the resource must exist at the time the widget that has the resource is created.

XmNtopAttachment

Specifies attachment of the top side of the child. It can have following values:

XmATTACH_NONE

Do not attach the top side of the child. If **XmNbottomAttachment** is also **XmATTACH_NONE**, this value is ignored and the child is given a default top attachment.

XmATTACH_FORM

Attach the top side of the child to the top side of the Form.

XmATTACH_OPPOSITE_FORM

Attach the top side of the child to the bottom side of the Form. **XmNtopOffset** can be used to determine the visibility of the child.

XmATTACH_WIDGET

Attach the top side of the child to the bottom side of the widget or gadget specified in the **XmNtopWidget** resource. If **XmNtopWidget** is NULL, **XmATTACH_WIDGET** is replaced by **XmATTACH_FORM** and the child is attached to the top side of the Form.

XmATTACH_OPPOSITE_WIDGET
Attach the top side of the child to the top side of the widget or gadget specified in the **XmNtopWidget** resource.

XmATTACH_POSITION
Attach the top side of the child to a position that is relative to the top side of the Form and in proportion to the height of the Form. This position is determined by the **XmNtopPosition** and **XmNfractionBase** resources.

XmATTACH_SELF
Attach the top side of the child to a position that is proportional to the current *y* value of the child divided by the height of the Form. This position is determined by the **XmNtopPosition** and **XmNfractionBase** resources. **XmNtopPosition** is set to a value proportional to the current *y* value of the child divided by the height of the Form.

XmNtopOffset
Specifies the constant offset between the top side of the child and the object to which it is attached. The relationship established remains, regardless of any resizing operations that occur. When this resource is explicitly set, the value of **XmNverticalSpacing** is ignored.

XmNtopPosition
This resource is used to determine the position of the top side of the child when the child's **XmNtopAttachment** is set to **XmATTACH_POSITION**. In this case, the position of the top side of the child is relative to the top side of the Form and is a fraction of the height of the Form. This fraction is the value of the child's **XmNtopPosition** resource divided by the value of the Form's **XmNfractionBase**. For example, if the child's **XmNtopPosition** is 50, the Form's **XmNfractionBase** is 100, and the Form's height is 200, the position of the top side of the child is 100.

XmNtopWidget

Specifies the widget or gadget to which the top side of the child is attached. This resource is used if **XmNtopAttachment** is set to either **XmATTACH_WIDGET** or **XmATTACH_OPPOSITE_WIDGET**.

A string-to-widget resource converter is automatically installed for use with this resource. With this converter, the widget that is to be the value of the resource must exist at the time the widget that has the resource is created.

Inherited Resources

Form inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmBulletinBoard Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowOverlap XmCAllowOverlap	True Boolean	CSG
XmNautoUnmanage XmCAutoUnmanage	True Boolean	CG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNcancelButton XmCWidget	NULL Widget	SG
XmNdefaultButton XmCWidget	NULL Widget	SG
XmNdefaultPosition XmCDefaultPosition	True Boolean	CSG
XmNdialogStyle XmCDialogStyle	dynamic unsigned char	CSG
XmNdialogTitle XmCDialogTitle	NULL XmString	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	0 Dimension	CSG
XmNmarginWidth XmCMarginWidth	0 Dimension	CSG
XmNnoResize XmCNoResize	False Boolean	CSG
XmNresizePolicy XmCResizePolicy	XmRESIZE_ANY unsigned char	CSG

Name Class	Default Type	Access
XmNshadowType XmCShadowType	XmSHADOW_OUT unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNtextTranslations XmCTranslations	NULL XtTranslations	C
XmNunmapCallback XmCCallback	NULL XtCallbackList	C

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	dynamic Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	N/A
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

XmForm inherits translations from **XmBulletinBoard**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmBulletinBoard(3X)**, **XmCreateForm**, **XmCreateFormDialog(3X)**, and **XmManager(3X)**.

XmFrame—The Frame widget class

Synopsis

#include <Xm/Frame.h>

Description

Frame is a very simple manager used to enclose a single work area child in a border drawn by Frame. It uses the Manager class resources for border drawing and performs geometry management so that its size always matches its child's outer size plus the Frame's margins and shadow thickness.

Frame is most often used to enclose other managers when the application developer desires the manager to have the same border appearance as the primitive widgets. Frame can also be used to enclose primitive widgets that do not support the same type of border drawing. This gives visual consistency when you develop applications using diverse widget sets. Constraint resources are used to designate a child as the Frame title, align its text, and control its vertical alignment in relation to Frame's top shadow. The title appears only at the top of the Frame.

If the Frame's parent is a Shell widget, the **XmNshadowType** resource defaults to **XmSHADOW_OUT**, and the Manager's **XmNshadowThickness** resource defaults to 1.

If the Frame's parent is not a Shell widget, the **XmNshadowType** resouce defaults to **XmSHADOW_ETCHED_IN**, and the Manager's **XmNshadowThickness** resource defaults to 2.

Classes

Frame inherits behavior and resources from the **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is **xmFrameWidgetClass**.

The class name is **XmFrame**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmFrame Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNmarginWidth XmCMarginWidth	0 Dimension	CSG
XmNmarginHeight XmCMarginHeight	0 Dimension	CSG
XmNshadowType XmCShadowType	dynamic unsigned char	CSG

XmNmarginWidth

Specifies the padding space on the left and right sides between Frame's child and Frame's shadow drawing.

XmNmarginHeight

Specifies the padding space on the top and bottom sides between Frame's child and Frame's shadow drawing. When a title is present, the top margin equals the value specified by this resource plus the distance (if any) that the title extends below the top shadow.

XmNshadowType

Describes the drawing style for Frame. This resource can have the following values:

XmSHADOW_IN

Draws Frame so that it appears inset. This means that the bottom shadow visuals and top shadow visuals are reversed.

XmSHADOW_OUT

Draws Frame so that it appears outset. This is the default if Frame's parent is a Shell widget.

XmSHADOW_ETCHED_IN
Draws Frame using a double line giving the effect of a line etched into the window. The thickness of the double line is equal to the value of **XmNshadowThickness**. This is the default when Frame's parent is not a Shell widget.

XmSHADOW_ETCHED_OUT
Draws Frame using a double line giving the effect of a line coming out of the window. The thickness of the double line is equal to the value of **XmNshadowThickness**.

XmFrame Constraint Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildType XmCChildType	XmFRAME_WORKAREA_CHILD unsigned char	CSG
XmNchildHorizontalAlignment XmCChildHorizontalAlignment	XmALIGNMENT_BEGINNING unsigned char	CSG
XmNchildHorizontalSpacing XmCChildHorizontalSpacing	dynamic Dimension	CSG
XmNchildVerticalAlignment XmCChildVerticalAlignment	XmALIGNMENT_CENTER unsigned char	CSG

XmNchildType
Specifies whether a child is a title or work area. Frame supports a single title and/or work area child. The possible values are

- **XmFRAME_TITLE_CHILD**
- **XmFRAME_WORKAREA_CHILD**
- **XmFRAME_GENERIC_CHILD**

The Frame geometry manager ignores any child of type **XmFRAME_GENERIC_CHILD**.

XmNchildHorizontalAlignment
Specifies the alignment of the title. This resource has the following values:

- **XmALIGNMENT_BEGINNING**
- **XmALIGNMENT_CENTER**
- **XmALIGNMENT_END**

See the description of **XmNalignment** in the **XmLabel** reference page for an explanation of these values.

XmNchildHorizontalSpacing
Specifies the minimum distance between either edge of the title text and the inner edge of the Frame shadow. Clipping of the title text occurs in order to maintain this spacing. The default value is the margin width of the Frame.

XmNchildVerticalAlignment
Specifies the vertical alignment of the title text, or the title area in relation to the top shadow of the Frame.

XmALIGNMENT_BASELINE_BOTTOM
Causes the baseline of the title to align vertically with the top shadow of the Frame. In the case of a multi-line title, the baseline of the last line of text aligns vertically with the top shadow of the Frame.

XmALIGNMENT_BASELINE_TOP
Causes the baseline of the first line of the title to align vertically with the top shadow of the Frame.

XmALIGNMENT_WIDGET_TOP
Causes the top edge of the title area to align vertically with the top shadow of the Frame.

XmALIGNMENT_CENTER
Causes the center of the title area to align vertically with the top shadow of the Frame.

XmALIGNMENT_WIDGET_BOTTOM
Causes the bottom edge of the title area to align vertically with the top shadow of the Frame.

Inherited Resources

Frame inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the man page for that superclass.

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

XmFrame inherits translations from **XmManager**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreateFrame(3X)**, and **XmManager(3X)**.

XmGadget—The Gadget widget class

Synopsis

#include <Xm/Xm.h>

Description

Gadget is a widget class used as a supporting superclass for other gadget classes. It handles shadow-border drawing and highlighting, traversal activation and deactivation, and various callback lists needed by gadgets.

The color and pixmap resources defined by **XmManager** are directly used by gadgets. If **XtSetValues** is used to change one of the resources for a manager widget, all of the gadget children within the manager also change.

Classes

Gadget inherits behavior and resources from **Object** and **RectObj** classes.

The class pointer is **xmGadgetClass**.

The class name is **XmGadget**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a .Xdefaults file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a .Xdefaults file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	G
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	G
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	G
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

XmNbottomShadowColor

Contains the color to use to draw the bottom and right sides of the border shadow.

XmNhelpCallback

Specifies the list of callbacks that is called when the help key sequence is pressed. The reason sent by the callback is **XmCR_HELP**.

XmNhighlightColor

Contains the color of the highlighting rectangle.

XmNhighlightOnEnter
Specifies if the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is **XmEXPLICIT**, this resource is ignored, and the widget is highlighted when it has the focus. If the shell's focus policy is **XmPOINTER** and if this resource is True, the highlighting rectangle is drawn when the the cursor moves into the widget. If the shell's focus policy is **XmPOINTER** and if this resource is False, the highlighting rectangle is not drawn when the the cursor moves into the widget. The default is False.

XmNhighlightThickness
Specifies the thickness of the highlighting rectangle.

XmNnavigationType
Determines whether the widget is a tab group.

XmNONE Indicates that the widget is not a tab group.

XmTAB_GROUP
Indicates that the widget is a tab group, unless the **XmNnavigationType** of another widget in the hierarchy is **XmEXCLUSIVE_TAB_GROUP**.

XmSTICKY_TAB_GROUP
Indicates that the widget is a tab group, even if the **XmNnavigationType** of another widget in the hierarchy is **XmEXCLUSIVE_TAB_GROUP**.

XmEXCLUSIVE_TAB_GROUP
Indicates that the widget is a tab group and that widgets in the hierarchy whose **XmNnavigationType** is **XmTAB_GROUP** are not tab groups.

When a parent widget has an **XmNnavigationType** of **XmEXCLUSIVE_TAB_GROUP**, traversal of non-tab-group widgets within the group is based on the order of those widgets in their parent's **XmNchildren** list.

When the **XmNnavigationType** of any widget in a hierarchy is **XmEXCLUSIVE_TAB_GROUP**, traversal of tab groups in the hierarchy proceeds to widgets in the order in which their

XmNnavigationType resources were specified as **XmEXCLUSIVE_TAB_GROUP** or **XmSTICKY_TAB_GROUP**, whether by creating the widgets with that value, by calling **XtSetValues**, or by calling **XmAddTabGroup**.

XmNshadowThickness
Specifies the size of the drawn border shadow.

XmNtopShadowColor
Contains the color to use to draw the top and left sides of the border shadow.

XmNtraversalOn
Specifies traversal activation for this gadget.

XmNunitType
Provides the basic support for resolution independence. It defines the type of units a widget uses with sizing and positioning resources. If the widget's parent is a subclass of **XmManager** and if the **XmNunitType** resource is not explicitly set, it defaults to the unit type of the parent widget. If the widget's parent is not a subclass of **XmManager**, the resource has a default unit type of **XmPIXELS**.

XmNunitType can have the following values:

XmPIXELS All values provided to the widget are treated as normal pixel values.

Xm100TH_MILLIMETERS
All values provided to the widget are treated as 1/100 of a millimeter.

Xm1000TH_INCHES
All values provided to the widget are treated as 1/1000 of an inch.

Xm100TH_POINTS
All values provided to the widget are treated as 1/100 of a point. A point is a unit used in text processing applications and is defined as 1/72 of an inch.

Xm100TH_FONT_UNITS
All values provided to the widget are treated as 1/100 of a font unit. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

XmNuserData
Allows the application to attach any necessary specific data to the gadget. This is an internally unused resource.

Inherited Resources

Gadget inherits the following resources from the superclass described in the following table. For a complete description of each resource, refer to the reference page for that superclass.

RectObj Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	0 Dimension	N/A
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked. For this callback, *reason* is set to **XmCR_HELP**.

event Points to the **XEvent** that triggered the callback.

Behavior

Gadgets cannot have translations associated with them. Because of this, a Gadget's behavior is determined by the Manager widget into which the Gadget is placed. If focus is on a Gadget, events are passed to the Gadget by its Manager.

Related Information

Object(3X), **RectObj(3X)**, **XmManager(3X)**, and **XmScreen(3X)**.

XmGetAtomName—A function that returns the string representation for an atom

Synopsis

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

String XmGetAtomName (display, atom)
        Display     * display;
        Atom        atom;
```

Description

XmGetAtomName returns the string representation for an atom. It mirrors the **Xlib** interfaces for atom management but provides client-side caching. When and where caching is provided in **Xlib**, the routines will become pseudonyms for the **Xlib** routines.

display	Specifies the connection to the X server
atom	Specifies the atom for the property name you want returned

Return Value

Returns a string.

XmGetColorCalculation—A function to get the procedure used for default color calculation

Synopsis

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

XmColorProc XmGetColorCalculation ()
```

Description

XmGetColorCalculation returns the procedure being used to calculate default colors.

For a description of **XmColorProc**, see **XmSetColorCalculation(3X)**.

Return Value

Returns the procedure used for default color calculation.

Related Information

XmChangeColor(3X), **XmGetColors(3X)**, and **XmSetColorCalculation(3X)**.

XmGetColors—A function that generates foreground, select, and shadow colors

Synopsis

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

void XmGetColors (screen, colormap, background, foreground, top_shadow,
bottom_shadow, select)
        Screen      * screen;
        Colormap    colormap;
        Pixel       background;
        Pixel       * foreground;
        Pixel       * top_shadow;
        Pixel       * bottom_shadow;
        Pixel       * select;
```

Description

XmGetColors takes a screen, a colormap, and a background pixel, and returns pixel values for foreground, select, and shadow colors.

screen Specifies the screen for which these colors should be allocated.

colormap Specifies the colormap from which these colors should be allocated.

background Specifies the background on which the colors should be based.

foreground Specifies a pointer to the returned foreground pixel value. If this argument is NULL no value is allocated or returned for this color.

top_shadow Specifies a pointer to the returned top shadow pixel value. If this argument is NULL, no value is allocated or returned for this color.

bottom_shadow
Specifies a pointer to the returned bottom shadow pixel value. If this argument is NULL, no value is allocated or returned for this color.

select Specifies a pointer to the returned select pixel value. If this argument is NULL, no value is allocated or returned for this color.

Related Information

XmChangeColor(3X), **XmGetColorCalculation(3X)**, and **XmSetColorCalculation(3X)**.

XmGetDestination—A function that returns the widget ID of the widget to be used as the current destination for quick paste and certain clipboard operations

Synopsis

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

Widget XmGetDestination (display)
        Display         *display;
```

Description

XmGetDestination returns the widget that is the current destination on the specified display. The destination is generally the last editable widget on which a select, edit, insert, or paste operation was performed and is the destination for quick paste and certain clipboard functions. The destination is NULL if the application makes this call before any of the specified operations have been performed on an editable widget.

display Specifies the display whose destination widget is to be queried

Return Value

Returns the widget ID for the current destination or NULL if there is no current destination.

XmGetDragContext—A Drag and Drop function that retrieves the DragContext widget ID associated with a timestamp

Synopsis

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

Widget XmGetDragContext (refwidget, timestamp)
        Widget          refwidget;
        Time            timestamp;
```

Description

XmGetDragContext returns the widget ID of the active DragContext associated with a given display and timestamp. A timestamp uniquely identifies which DragContext is active when more than one drag and drop transaction has been initiated on a display. If the specified timestamp matches a timestamp processed between the start and finish of a single drag and drop transaction, the function returns the corresponding DragContext ID.

refwidget Specifies the ID of the widget that the routine uses to identify the intended display. The function returns the ID of the DragContext associated with the display value passed by this widget.

timestamp Specifies a timestamp.

For a complete definition of DragContext and its associated resources, see **XmDragContext(3X)**.

Return Value

Returns the ID of the DragContext widget that is active for the specified timestamp. Otherwise, returns NULL if no active DragContext is found.

Related Information

XmDragContext(3X).

XmGetFocusWidget—Returns the ID of the widget that has keyboard focus

Synopsis

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

Widget XmGetFocusWidget (widget)
        Widget          widget;
```

Description

XmGetFocusWidget examines the hierarchy that contains the specified widget and returns the ID of the widget that has keyboard focus. The function extracts the widget ID from the associated Shell widget; therefore, the specified widget can be located anywhere in the hierarchy.

widget Specifies a widget ID within a given hierarchy

Return Value

Returns the ID of the widget with keyboard focus. If no child of the Shell has focus, the function returns NULL.

Related Information

XmProcessTraversal(3X).

XmGetMenuCursor—A function that returns the cursor ID for the current menu cursor

Synopsis

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

Cursor XmGetMenuCursor (display)
        Display         * display;
```

Description

XmGetMenuCursor queries the menu cursor currently being used by this client on the specified display and returns the cursor ID. This function returns the menu cursor for the default screen of the display.

NOTE: XmGetMenuCursor is obsolete and exists for compatibility with previous releases. Instead of using this function, call **XtGetValues** for the XmScreen resource **XmNmenuCursor**.

display Specifies the display whose menu cursor is to be queried

Return Value

Returns the cursor ID for the current menu cursor or the value None if a cursor is not yet defined. A cursor will not be defined if the application makes this call before the client has created any menus on the specified display.

Related Information

XmScreen(3X).

XmGetPixmap—A pixmap caching function that generates a pixmap, stores it in a pixmap cache, and returns the pixmap

Synopsis

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

Pixmap XmGetPixmap (screen, image_name, foreground, background)
        Screen      *screen;
        char        *image_name;
        Pixel       foreground;
        Pixel       background;
```

Description

XmGetPixmap uses the parameter data to perform a lookup in the pixmap cache to see if a pixmap has already been generated that matches the data. If one is found, a reference count is incremented and the pixmap is returned. Applications should use **XmDestroyPixmap** when the pixmap is no longer needed.

screen Specifies the display screen on which the pixmap is to be drawn. The depth of the pixmap is the default depth for this screen.

image_name Specifies the name of the image to be used to generate the pixmap

foreground Combines the image with the *foreground* color to create the pixmap if the image referenced is a bit-per-pixel image

background Combines the image with the *background* color to create the pixmap if the image referenced is a bit-per-pixel image

If a pixmap is not found, *image_name* is used to perform a lookup in the image cache. If an image is found, it is used to generate the pixmap, which is then cached and returned.

If an image is not found, the *image_name* is used as a filename, and a search is made for an **X10** or **X11** bitmap file. If it is found, the file is read, converted into an image, and cached in the image cache. The image is then used to generate a pixmap, which is cached and returned.

If *image_name* has a leading slash (*/*), it specifies a full pathname, and **XmGetPixmap** opens the file as specified. Otherwise, *image_name* specifies a filename. In this case, **XmGetPixmap** looks for the file along a search path specified by the **XBMLANGPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set.

The **XBMLANGPATH** environment variable specifies a search path for X bitmap files. It can contain the substitution field **%B**, where the *image_name* argument to **XmGetPixmap** is substituted for **%B**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to **bitmaps**, and **%S** is always mapped to NULL.

If **XBMLANGPATH** is not set but the environment variable **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$XAPPLRESDIR/%L/bitmaps/%N/%B**
- **$XAPPLRESDIR/%l/bitmaps/%N/%B**
- **$XAPPLRESDIR/bitmaps/%N/%B**
- **$XAPPLRESDIR/%L/bitmaps/%B**
- **$XAPPLRESDIR/%l/bitmaps/%B**
- **$XAPPLRESDIR/bitmaps/%B**
- **$HOME/bitmaps/%B**
- **$HOME/%B**
- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**
- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

If neither **XBMLANGPATH** nor **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$HOME/%L/bitmaps/%N/%B**
- **$HOME/%l/bitmaps/%N/%B**

- **$HOME/bitmaps/%N/%B**
- **$HOME/%L/bitmaps/%B**
- **$HOME/%l/bitmaps/%B**
- **$HOME/bitmaps/%B**
- **$HOME/%B**
- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**
- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

These paths are defaults that vendors may change. For example, a vendor may use different directories for **/usr/lib/X11** and **/usr/include/X11**.

The following substitutions are used in these paths:

%B	The image name, from the *image_name* argument
%N	The class name of the application
%L	The display's language string
%l	The language component of the display's language string

Return Value

Returns a pixmap when successful; returns **XmUNSPECIFIED_PIXMAP** if the image corresponding to *image_name* cannot be found.

Related Information

XmDestroyPixmap(3X), **XmGetPixmapByDepth(3X)**, **XmInstallImage(3X)**, and **XmUninstallImage(3X)**.

XmGetPixmapByDepth—A pixmap caching function that generates a pixmap, stores it in a pixmap cache, and returns the pixmap

Synopsis

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

Pixmap XmGetPixmapByDepth (screen, image_name,foreground, background, depth)
        Screen      *screen;
        char        *image_name;
        Pixel       foreground;
        Pixel       background;
        int         depth;
```

Description

XmGetPixmapByDepth uses the parameter data to perform a lookup in the pixmap cache to see if a pixmap has already been generated that matches the data. If one is found, a reference count is incremented and the pixmap is returned. Applications should use **XmDestroyPixmap** when the pixmap is no longer needed.

screen Specifies the display screen on which the pixmap is to be drawn

image_name Specifies the name of the image to be used to generate the pixmap

foreground Combines the image with the *foreground* color to create the pixmap if the image referenced is a bit-per-pixel image

background Combines the image with the *background* color to create the pixmap if the image referenced is a bit-per-pixel image

depth Specifies the depth of the pixmap

If a matching pixmap is not found, *image_name* is used to perform a lookup in the image cache. If an image is found, it is used to generate the pixmap, which is then cached and returned.

If an image is not found, *image_name* is used as a filename, and a search is made for an **X10** or **X11** bitmap file. If it is found, the file is read, converted into an image, and cached in the image cache. The image is then used to generate a pixmap, which is cached and returned.

If *image_name* has a leading / (slash), it specifies a full pathname, and **XmGetPixmapByDepth** opens the file as specified. Otherwise, *image_name* specifies a filename. In this case, **XmGetPixmapByDepth** looks for the file along a search path specified by the **XBMLANGPATH** environment variable or by a default search path, which varies depending on whether or not the **XAPPLRESDIR** environment variable is set.

The **XBMLANGPATH** environment variable specifies a search path for X bitmap files. It can contain the substitution field **%B**, where the *image_name* argument to **XmGetPixmapByDepth** is substituted for **%B**. It can also contain the substitution fields accepted by **XtResolvePathname**. The substitution field **%T** is always mapped to **bitmaps**, and **%S** is always mapped to NULL.

If **XBMLANGPATH** is not set, but the environment variable **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$XAPPLRESDIR/%L/bitmaps/%N/%B**
- **$XAPPLRESDIR/%l/bitmaps/%N/%B**
- **$XAPPLRESDIR/bitmaps/%N/%B**
- **$XAPPLRESDIR/%L/bitmaps/%B**
- **$XAPPLRESDIR/%l/bitmaps/%B**
- **$XAPPLRESDIR/bitmaps/%B**
- **$HOME/bitmaps/%B**
- **$HOME/%B**
- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**
- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

If neither **XBMLANGPATH** nor **XAPPLRESDIR** is set, the following pathnames are searched:

- **%B**
- **$HOME/%L/bitmaps/%N/%B**
- **$HOME/%l/bitmaps/%N/%B**

- **$HOME/bitmaps/%N/%B**
- **$HOME/%L/bitmaps/%B**
- **$HOME/%l/bitmaps/%B**
- **$HOME/bitmaps/%B**
- **$HOME/%B**
- **/usr/lib/X11/%L/bitmaps/%N/%B**
- **/usr/lib/X11/%l/bitmaps/%N/%B**
- **/usr/lib/X11/bitmaps/%N/%B**
- **/usr/lib/X11/%L/bitmaps/%B**
- **/usr/lib/X11/%l/bitmaps/%B**
- **/usr/lib/X11/bitmaps/%B**
- **/usr/include/X11/bitmaps/%B**

These paths are defaults that vendors may change. For example, a vendor may use different directories for **/usr/lib/X11** and **/usr/include/X11**.

The following substitutions are used in these paths:

%B	The image name, from the *image_name* argument
%N	The class name of the application
%L	The display's language string
%l	The language component of the display's language string

Return Value

Returns a pixmap when successful; returns **XmUNSPECIFIED_PIXMAP** if the image corresponding to *image_name* cannot be found.

Related Information

XmDestroyPixmap(3X), **XmInstallImage(3X)**, and **XmUninstallImage(3X)**.

XmGetPostedFromWidget—A RowColumn function that returns the widget from which a menu was posted

Synopsis

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

Widget XmGetPostedFromWidget (menu)
        Widget          menu;
```

Description

XmGetPostedFromWidget returns the widget from which a menu was posted. For torn-off menus, this function returns the widget from which the menu was originally torn. An application can use this routine during the activate callback to determine the context in which the menu callback should be interpreted.

menu Specifies the widget ID of the menu

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the widget ID of the widget from which the menu was posted. If the menu is a Popup Menu, the returned widget is the widget from which the menu was popped up. If the menu is a Pulldown Menu, the returned widget is the MenuBar or OptionMenu from which the widget was pulled down.

Related Information

XmRowColumn(3X).

XmGetSecondaryResourceData—A function that provides access to secondary widget resource data

Synopsis

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

Cardinal XmGetSecondaryResourceData (widget_class, secondary_data_return)
        WidgetClass                 widget_class;
        XmSecondaryResourceData     **secondary_data_return;
```

Description

Some Motif widget classes (such as Gadget, Text, and VendorShell) have resources that are not accessible through the functions **XtGetResourceList** and **XtGetConstraintResourceList**. In order to retrieve the descriptions of these resources, an application must use **XmGetSecondaryResourceData**.

When a widget class has such resources, this function provides descriptions of the resources in one or more data structures. **XmGetSecondaryResourceData** takes a widget class argument and returns the number of these data structures associated with the widget class. If the return value is greater than 0 (zero), the function allocates and fills an array of pointers to the corresponding data structures. It returns this array at the address that is the value of the *secondary_data_return* argument.

The type **XmSecondaryResourceData** is a pointer to a structure with two members that are useful to an application: *resources*, of type **XtResourceList**, and *num_resources*, of type **Cardinal**. The *resources* member is a list of the widget resources that are not accessible using Xt functions. The *num_resources* member is the length of the *resources* list.

If the return value is greater than 0 (zero), **XmGetSecondaryResourceData** allocates memory that the application must free. Use **XtFree** to free the resource list in each structure (the value of the *resources* member), the structures themselves, and the array of pointers to the structures (the array whose address is *secondary_data_return*).

widget_class Specifies the widget class for which secondary resource data is to be retrieved.

secondary_data_return
Specifies a pointer to an array of **XmSecondaryResourceData** pointers to be returned by this function. If the widget class has no secondary resource data, for example, if the value returned by the function is 0 (zero), the function returns no meaningful value for this argument.

Return Value

Returns the number of secondary resource data structures associated with this widget class.

Example

The following example uses **XmGetSecondaryResourceData** to print the names of the secondary resources of the Motif Text widget and then frees the data allocated by the function:

```
XmSecondaryResourceData * block_array ;
Cardinal num_blocks, i, j ;
if (num_blocks = XmGetSecondaryResourceData (xmTextWidgetClass,
                                             &block_array)) {
  for (i = 0; i < num_blocks; i++) {
    for (j = 0 ; j < block_array[i]->num_resources; j++) {
      printf("%s\n", block_array[i]->resources[j].resource_name);
    }
    XtFree((char*)block_array[i]->resources);
    XtFree((char*)block_array[i]);
  }
  XtFree((char*)block_array);
}
```

XmGetTabGroup—Returns the widget ID of a tab group

Synopsis

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

Widget XmGetTabGroup (widget)
        Widget          widget;
```

Description

XmGetTabGroup returns the widget ID of the tab group that contains the specified widget.

widget Specifies a widget ID within a tab group

Return Value

Returns the widget ID of a tab group or shell, determined as follows:

- If *widget* is a tab group or shell, returns *widget*
- If neither *widget* nor any ancestor up to the nearest shell is a tab group, returns the nearest ancestor of *widget* that is a shell
- Otherwise, returns the nearest ancestor of *widget* that is a tab group

Related Information

XmAddTabGroup(3X), **XmManager(3X)**, and **XmPrimitive(3X)**.

XmGetTearOffControl—A RowColumn function that obtains the widget ID for the tear-off control in a menu

Synopsis

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

Widget XmGetTearOffControl (menu)
        Widget          menu;
```

Description

XmGetTearOffControl provides the application with the means for obtaining the widget ID of the internally created tear-off control in a tear-off menu.

RowColumn creates a tear-off control for a PulldownMenu or PopupMenu when the **XmNtearOffModel** resource is initialized or set to **XmTEAR_OFF_ENABLED**. The tear-off control is a widget that appears as the first element in the menu. The user tears off the menu by means of mouse or keyboard events in the tear-off control.

The tear-off control has Separator-like behavior. Once the application has obtained the widget ID of the tear-off control, it can set resources to specify the appearance of the control. The application or user can also set these resources in a resource file by using the name of the control, which is **TearOffControl**. For a list of the resources the application or user can set, see **XmRowColumn(3X)**.

menu Specifies the widget ID of the RowColumn PulldownMenu or PopupMenu

For more information on tear-off menus and a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the widget ID for the tear-off control, or NULL if no tear-off control exists. An application should not assume that the returned widget will be of any particular class.

Related Information

XmRowColumn(3X).

XmGetVisibility—A function that determines if a widget is visible

Synopsis

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

XmVisibility XmGetVisibility (widget)
        Widget          widget;
```

Description

XmGetVisibility returns the visibility state of the specified widget.

widget Specifies the ID of the widget

Return Value

Returns one of the following values:

- **XmVISIBILITY_UNOBSCURED**
- **XmVISIBILITY_PARTIALLY_OBSCURED**
- **XmVISIBILITY_FULLY_OBSCURED**

Related Information

XmIsTraversable(3X), **XmManager(3X)**, and **XmProcessTraversal(3X)**.

XmGetXmDisplay—A Display function that returns the **XmDisplay** object ID for a specified display

Synopsis

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

Widget XmGetXmDisplay (display)
        Display         *display;
```

Description

XmGetXmDisplay returns the **XmDisplay** object ID associated with a display. The application can access Display resources with **XtGetValues**.

display Specifies the display for which the **XmDisplay** object ID is to be returned

For a complete definition of Display and its associated resources, see **XmDisplay(3X)**.

Return Value

Returns the **XmDisplay** object ID for the specified display.

Related Information

XmDisplay(3X).

XmGetXmScreen—A Screen function that returns the **XmScreen** object ID for a specified screen

Synopsis

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

Widget XmGetXmScreen (screen)
        Screen          *screen;
```

Description

XmGetXmScreen returns the **XmScreen** object ID associated with a screen. The application can access and manipulate Screen resources with **XtGetValues** and **XtSetValues**.

screen Specifies the screen for which the **XmScreen** ID is to be returned

For a complete definition of Screen and its associated resources, see **XmScreen(3X)**.

Return Value

Returns the **XmScreen** object ID.

Related Information

XmScreen(3X).

XmInstallImage—A pixmap caching function that adds an image to the image cache

Synopsis

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

Boolean XmInstallImage (image, image_name)
        XImage      * image;
        char        * image_name;
```

Description

XmInstallImage stores an image in an image cache that can later be used to generate a pixmap. Part of the installation process is to extend the resource converter used to reference these images. The resource converter is given the image name so that the image can be referenced in a **.Xdefaults** file. Since an image can be referenced by a widget through its pixmap resources, it is up to the application to ensure that the image is installed before the widget is created.

image Points to the image structure to be installed. The installation process does not make a local copy of the image. Therefore, the application should not destroy the image until it is uninstalled from the caching functions.

image_name Specifies a string that the application uses to name the image. After installation, this name can be used in a **.Xdefaults** file for referencing the image. A local copy of the name is created by the image caching functions.

The image caching functions provide a set of eight preinstalled images. These names can be used within a **.Xdefaults** file for generating pixmaps for the resource for which they are provided.

Image Name	Description
background	A tile of solid background
25_foreground	A tile of 25% foreground, 75% background
50_foreground	A tile of 50% foreground, 50% background
75_foreground	A tile of 75% foreground, 25% background
horizontal	A tile of horizontal lines of the two colors
vertical	A tile of vertical lines of the two colors
slant_right	A tile of slanting lines of the two colors
slant_left	A tile of slanting lines of the two colors

Return Value

Returns True when successful; returns False if NULL *image*, NULL *image_name*, or duplicate *image_name* is used as a parameter value.

Related Information

XmUninstallImage(3X), **XmGetPixmap(3X)**, and **XmDestroyPixmap(3X)**.

XmInternAtom—A function that returns an atom for a given name

Synopsis

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

Atom XmInternAtom (display, name, only_if_exists)
        Display     * display;
        String      name;
        Boolean     only_if_exists;
```

Description

XmInternAtom returns an atom for a given name. It mirrors the **Xlib** interfaces for atom management, but provides client-side caching. When and where caching is provided in **Xlib**, the routines will become pseudonyms for the **Xlib** routines.

display Specifies the connection to the X server

name Specifies the name associated with the atom you want returned

only_if_exists
Specifies a Boolean value that indicates whether **XInternAtom** creates the atom

Return Value

Returns an atom.

XmIsMotifWMRunning—A function that determines whether the window manager is running

Synopsis

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

Boolean XmIsMotifWMRunning (shell)
        Widget      shell;
```

Description

XmIsMotifWMRunning lets a user know whether the Motif Window Manager is running on a screen that contains a specific widget hierarchy. This function first sees whether the **_MOTIF_WM_INFO** property is present on the root window of the shell's screen. If it is, its window field is used to query for the presence of the specified window as a child of root.

shell Specifies the shell whose screen will be tested for **mwm**'s presence.

Return Value

Returns True if MWM is running.

XmIsTraversable—A function that identifies whether a widget can be traversed

Synopsis

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

Boolean XmIsTraversable (widget)
        Widget      widget;
```

Description

XmIsTraversable determines whether the specified widget is eligible to receive focus through keyboard traversal. In general, a widget is eligible to receive focus when all of the following conditions are true:

- The widget and its ancestors are not being destroyed, are sensitive, and have a value of True for **XmNtraversalOn**.
- The widget and its ancestors are realized, managed, and (except for gadgets) mapped.
- Some part of the widget's rectangular area is unobscured by the widget's ancestors, or some part of the widget's rectangular area is inside the work window (but possibly outside the clip window) of a ScrolledWindow whose **XmNscrollingPolicy** is **XmAUTOMATIC** and whose **XmNtraverseObscuredCallback** is not NULL.

Some widgets may not be eligible to receive focus even if they meet all these conditions. For example, most managers cannot receive focus through keyboard traversal. Some widgets may be eligible to receive focus under particular conditions. For example, a DrawingArea is eligible to receive focus if it meets the conditions above and has no child whose **XmNtraversalOn** resource is True.

widget Specifies the ID of the widget

Return Value

Returns True if the widget is eligible to receive focus through keyboard traversal; otherwise, returns False.

Related Information

XmGetVisibility(3X) and **XmProcessTraversal(3X)**.

XmLabel—The Label widget class

Synopsis

#include <Xm/Label.h>

Description

Label is an instantiable widget and is also used as a superclass for other button widgets, such as PushButton and ToggleButton. The Label widget does not accept any button or key input, and the help callback is the only callback defined. Label also receives enter and leave events.

Label can contain either text or a pixmap. Label text is a compound string. Refer to the *OSF/Motif Programmer's Guide* for more information on compound strings. The text can be multilingual, multiline, and/or multifont. When a Label is insensitive, its text is stippled, or the user-supplied insensitive pixmap is displayed.

Label supports both accelerators and mnemonics primarily for use in Label subclass widgets that are contained in menus. Mnemonics are available in a menu system when the button is visible. Accelerators in a menu system are accessible even when the button is not visible. The Label widget displays the mnemonic by underlining the first matching character in the text string. The accelerator is displayed as a text string adjacent to the label text or pixmap.

Label consists of many margin fields surrounding the text or pixmap. These margin fields are resources that may be set by the user, but Label subclasses and Manager parents also modify some of these fields. They tend to modify the **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** resources and leave the **XmNmarginWidth** and **XmNmarginHeight** resources as set by the application.

Label takes into account **XmNshadowThickness** in determining its layout but does not draw the shadow. That is, if **XmNshadowThickness** is greater than 0 (zero), Label leaves space for the shadow, but the shadow does not appear.

In a Label **XmNtraversalOn** and **XmNhighlightOnEnter** are forced to False inside Popup MenuPanes, Pulldown MenuPanes, and OptionMenus. Otherwise, these resources default to False.

Classes

Label inherits behavior and resources from **Core** and **XmPrimitive**.

The class pointer is **xmLabelWidgetClass**.

The class name is **XmLabel**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmLabel Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	CSG
XmNacceleratorText XmCAcceleratorText	NULL XmString	CSG
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	0 Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	0 Dimension	CSG
XmNmarginRight XmCMarginRight	0 Dimension	CSG
XmNmarginTop XmCMarginTop	0 Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG
XmNmnemonic XmCMnemonic	NULL KeySym	CSG

Name Class	Default Type	Access
XmNmnemonicCharSet XmCMnemonicCharSet	XmFONTLIST_DEFAULT_TAG String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmNaccelerator

Sets the accelerator on a button widget in a menu, which activates a visible or invisible, but managed, button from the keyboard. This resource is a string that describes a set of modifiers and the key that may be used to select the button. The format of this string is identical to that used by the translations manager, with the exception that only a single event may be specified and only **KeyPress** events are allowed.

Accelerators for buttons are supported only for PushButtons and ToggleButtons in Pulldown and Popup MenuPanes.

XmNacceleratorText

Specifies the text displayed for the accelerator. The text is displayed adjacent to the label string or pixmap. Accelerator text for buttons is displayed only for PushButtons and ToggleButtons in Pulldown and Popup Menus.

XmNalignment

Specifies the label alignment for text or pixmap.

XmALIGNMENT_BEGINNING (left alignment)

Causes the left sides of the lines of text to be vertically aligned with the left edge of the widget window. For a pixmap, its left side is vertically aligned with the left edge of the widget window.

XmALIGNMENT_CENTER (center alignment)

Causes the centers of the lines of text to be vertically aligned in the center of the widget window. For a pixmap, its center is vertically aligned with the center of the widget window.

XmALIGNMENT_END (right alignment)
Causes the right sides of the lines of text to be vertically aligned with the right edge of the widget window. For a pixmap, its right side is vertically aligned with the right edge of the widget window.

The preceding descriptions for text are correct when **XmNstringDirection** is **XmSTRING_DIRECTION_L_TO_R**. When that resource is **XmSTRING_DIRECTION_R_TO_L**, the descriptions for **XmALIGNMENT_BEGINNING** and **XmALIGNMENT_END** are switched.

If the parent is a RowColumn whose **XmNisAligned** resource is True, **XmNalignment** is forced to the same value as the RowColumn's **XmNentryAlignment** if the RowColumn's **XmNrowColumnType** is **XmWORK_AREA** or if the widget is a subclass of XmLabel. Otherwise, the default is **XmALIGNMENT_CENTER**.

XmNfontList Specifies the font of the text used in the widget. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or XmMenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNbuttonFontList** (for button subclasses) or **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on the creation and structure of a font list.

XmNlabelInsensitivePixmap
Specifies a pixmap used as the button face if **XmNlabelType** is **XmPIXMAP** and the button is insensitive. The default value, **XmUNSPECIFIED_PIXMAP**, displays an empty label.

XmNlabelPixmap
Specifies the pixmap when **XmNlabelType** is **XmPIXMAP**. The default value, **XmUNSPECIFIED_PIXMAP**, displays an empty label.

XmNlabelString
Specifies the compound string when **XmNlabelType** is **XmSTRING**. If this value is NULL, it is initialized by converting the name of the widget to a compound string. Refer to **XmString(3X)** for more information on the creation and structure of compound strings.

XmNlabelType
Specifies the label type.

XmSTRING Displays text using **XmNlabelString**.

XmPIXMAP Displays pixmap using **XmNlabelPixmap** or **XmNlabelInsensitivePixmap**.

XmNmarginBottom
Specifies the amount of spacing between the bottom of the label text and the top of the bottom margin specified by **XmNmarginHeight**. This may be modified by Label's subclasses. For example, CascadeButton may increase this field to make room for the cascade pixmap.

XmNmarginHeight
Specifies an equal amount of spacing above the margin defined by **XmNmarginTop** and below the margin defined by **XmNmarginBottom**. **XmNmarginHeight** specifies the amount of spacing between the top edge of the margin set by **XmNmarginTop** and the bottom edge of the top shadow, and the amount of spacing between the bottom edge of the margin specified by **XmNmarginBottom** and the top edge of the bottom shadow.

XmNmarginLeft
Specifies the amount of spacing between the left edge of the label text and the right side of the left margin (specified by **XmNmarginWidth**). This may be modified by Label's subclasses. For example, ToggleButton may increase this field to make room for the toggle indicator and for spacing between the indicator and label. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginRight
Specifies the amount of spacing between the right edge of the label text and the left side of the right margin (specified by **XmNmarginWidth**). This may be modified by Label's subclasses. For example, CascadeButton may increase this field to make room for the cascade pixmap. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginTop

Specifies the amount of spacing between the top of the label text and the bottom of the top margin specified by **XmNmarginHeight**. This may be modified by Label's subclasses. For example, CascadeButton may increase this field to make room for the cascade pixmap.

XmNmarginWidth

Specifies an equal amount of spacing to the left of the margin defined by **XmNmarginLeft** and to the right of the margin defined by **XmNmarginRight**. **XmNmarginWidth** specifies the amount of spacing between the left edge of the margin set by **XmNmarginLeft** and the right edge of the left shadow, and the amount of spacing between the right edge of the margin specified by **XmNmarginRight** and the left edge of the right shadow.

XmNmnemonic

Provides the user with an alternate means of activating a button. A button in a MenuBar, a Popup MenuPane, or a Pulldown MenuPane can have a mnemonic.

This resource contains a keysym as listed in the X11 keysym table. The first character in the label string that exactly matches the mnemonic in the character set specified in **XmNmnemonicCharSet** is underlined when the button is displayed.

When a mnemonic has been specified, the user activates the button by pressing the mnemonic key while the button is visible. If the button is a CascadeButton in a MenuBar and the MenuBar does not have the focus, the user must use the **MAlt** modifier while pressing the mnemonic. The user can activate the button by pressing either the shifted or the unshifted mnemonic key.

XmNmnemonicCharSet

Specifies the character set of the mnemonic for the label. The default is **XmFONTLIST_DEFAULT_TAG**.

XmNrecomputeSize

Specifies a Boolean value that indicates whether the widget shrinks or expands to accommodate its contents (label string or pixmap) as a result of an **XtSetValues** resource value that would change the size of the widget. If True, the widget shrinks or expands to exactly fit the label string or pixmap. If False, the widget never attempts to change size on its own.

XmNstringDirection

Specifies the direction in which the string is to be drawn:

XmSTRING_DIRECTION_L_TO_R	Left to right
XmSTRING_DIRECTION_R_TO_L	Right to left

The default for this resource is determined at creation time. If no value is specified for this resource and the widget's parent is a manager, the value is inherited from the parent; otherwise, it defaults to **XmSTRING_DIRECTION_L_TO_R**.

Inherited Resources

Label inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the reference page for that superclass.

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	0 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	0 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	False Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

XmLabel includes translations from **Primitive**. The **XmLabel** translations are described in the following list. These translations may not directly correspond to a translation table.

BTransfer Press: **ProcessDrag()**

KHelp: **Help()**

The translations used by subclasses of XmLabel for menu traversal are described in the following list. These translations may not directly correspond to a translation table.

KLeft: **MenuTraverseLeft()**

KRight: **MenuTraverseRight()**

KUp: **MenuTraverseUp()**

KDown: **MenuTraverseDown()**

MAny KCancel: **MenuEscape()**

Action Routines

The **XmLabel** action routines are

Help(): In a Popup or Pulldown MenuPane, unposts all menus in the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget that had the focus before the menu system was entered. Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MenuEscape():

In a MenuBar, disarms the CascadeButton and the menu and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget that had the focus before the menu was entered.

In a top-level Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu and moves the focus to its CascadeButton.

In a Popup MenuPane, unposts the menu and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget from which the menu was posted.

MenuTraverseDown():

If the current menu item has a submenu and is in a MenuBar, then this action posts the submenu, disarms the current menu item, and arms the submenu's first traversable menu item.

If the current menu item is in a MenuPane, then this action disarms the current menu item and arms the item below it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's bottom edge, then this action wraps to the topmost menu item in the column to the right, if one exists. When the current menu item is at the bottom, rightmost corner of the MenuPane, then this action wraps to the tear-off control, if present, or to the top, leftmost menu item.

MenuTraverseLeft():

When the current menu item is in a MenuBar, then this action disarms the current item and arms the MenuBar item to the left. This action wraps within the MenuBar.

In MenuPanes, if the current menu item is not at the left edge of a MenuPane, this action disarms the current item and arms the item to its left. If the current menu item is at the left edge of a submenu attached to a MenuBar item, then this action unposts the submenu and traverses to the MenuBar item to the left, wrapping if necessary. If that MenuBar item has a submenu, it posts the submenu and arms the first traversable item in the submenu. If the current menu item is at the left edge of a submenu not directly attached to a MenuBar item, then this action unposts the current submenu only.

In Popup or Torn-off MenuPanes, when the current menu item is at the left edge, this action wraps within the MenuPane. If the current menu item is at the left edge of the MenuPane and not in the top row, this action wraps to the rightmost menu item in the row above. If the current menu item is in the upper, leftmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, rightmost menu item in the MenuPane.

MenuTraverseRight():

If the current menu item is in a MenuBar, then this action disarms the current item and arms the MenuBar item to the right. This action wraps within the MenuBar.

In MenuPanes, if the current menu item is a CascadeButton, then this action posts its associated submenu. If the current menu item is not a CascadeButton and is not at the right edge of a MenuPane, this action disarms the current item and arms the item to its right, wrapping if necessary. If the current menu item is not a CascadeButton and is at the right edge of a submenu that is a descendent of a MenuBar, then this action unposts all submenus and traverses to the MenuBar item to the right. If that MenuBar item has a submenu, it posts the submenu and arms the submenu's first traversable item.

In Popup or Torn-off menus, if the current menu item is not a CascadeButton and is at the right edge of a row (except the bottom row), this action wraps to the leftmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, rightmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, leftmost menu item of the MenuPane.

MenuTraverseUp():

When the current menu item is in a MenuPane, then this action disarms the current menu item and arms the item above it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's top edge, then this action wraps to the bottommost menu item in the column to the left, if one exists. When the current menu item is at the top, leftmost corner of the MenuPane, then this action wraps to the tear-off control, if present, or to the bottom, rightmost menu item.

ProcessDrag():

Drags the contents of a Label, identified when **BTransfer** is pressed. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT** for a label type of **XmSTRING**; otherwise, it is set to **PIXMAP** if the label type is **XmPIXMAP**. This action is undefined for Labels used in a menu system.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateLabel(3X)**, **XmFontListAppendEntry(3X)**, **XmStringCreate(3X)**, **XmStringCreateLtoR(3X)**, and **XmPrimitive(3X)**.

XmLabelGadget—The LabelGadget widget class

Synopsis

#include <Xm/LabelG.h>

Description

LabelGadget is an instantiable widget and is also used as a superclass for other button gadgets, such as PushButtonGadget and ToggleButtonGadget.

LabelGadget can contain either text or a pixmap. LabelGadget text is a compound string. Refer to the *OSF/Motif Programmer's* Guide for more information on compound strings. The text can be multilingual, multiline, and/or multifont. When a LabelGadget is insensitive, its text is stippled, or the user-supplied insensitive pixmap is displayed.

LabelGadget supports both accelerators and mnemonics primarily for use in LabelGadget subclass widgets that are contained in menus. Mnemonics are available in a menu system when the button is visible. Accelerators in a menu system are accessible even when the button is not visible. The LabelGadget displays the mnemonic by underlining the first matching character in the text string. The accelerator is displayed as a text string adjacent to the label text or pixmap.

LabelGadget consists of many margin fields surrounding the text or pixmap. These margin fields are resources that may be set by the user, but LabelGadget subclasses and Manager parents also modify some of these fields. They tend to modify the **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** resources and leave the **XmNmarginWidth** and **XmNmarginHeight** resources as set by the application.

LabelGadget takes into account **XmNshadowThickness** in determining its layout but does not draw the shadow. That is, if **XmNshadowThickness** is greater than 0 (zero), LabelGadget leaves space for the shadow, but the shadow does not appear.

In a LabelGadget **XmNtraversalOn** and **XmNhighlightOnEnter** are forced to False inside Popup MenuPanes, Pulldown MenuPanes, and OptionMenus. Otherwise, these resources default to False.

Classes

LabelGadget inherits behavior and resources from **Object**, **RectObj** and **XmGadget** classes.

The class pointer is **xmLabelGadgetClass**.

The class name is **XmLabelGadget**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmLabelGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	CSG
XmNacceleratorText XmCAcceleratorText	NULL XmString	CSG
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	0 Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	0 Dimension	CSG
XmNmarginRight XmCMarginRight	0 Dimension	CSG
XmNmarginTop XmCMarginTop	0 Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG

Name Class	Default Type	Access
XmNmnemonic XmCMnemonic	NULL KeySym	CSG
XmNmnemonicCharSet XmCMnemonicCharSet	dynamic String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmNaccelerator

Sets the accelerator on a button widget in a menu, which activates a visible or invisible, but managed, button from the keyboard. This resource is a string that describes a set of modifiers and the key that may be used to select the button. The format of this string is identical to that used by the translations manager, with the exception that only a single event may be specified and only **KeyPress** events are allowed.

Accelerators for buttons are supported only for PushButtonGadgets and ToggleButtonGadgets in Pulldown and Popup menus.

XmNacceleratorText

Specifies the text displayed for the accelerator. The text is displayed adjacent to the label string or pixmap. Accelerator text for buttons is displayed only for PushButtonGadgets and ToggleButtonGadgets in Pulldown and Popup Menus.

XmNalignment

Specifies the label alignment for text or pixmap.

XmALIGNMENT_BEGINNING (left alignment)

Causes the left sides of the lines of text to be vertically aligned with the left edge of the gadget. For a pixmap, its left side is vertically aligned with the left edge of the gadget.

XmALIGNMENT_CENTER (center alignment)

Causes the centers of the lines of text to be vertically aligned in the center of the gadget. For a pixmap, its center is vertically aligned with the center of the gadget.

XmALIGNMENT_END (right alignment)
Causes the right sides of the lines of text to be vertically aligned with the right edge of the gadget. For a pixmap, its right side is vertically aligned with the right edge of the gadget.

The preceding descriptions for text are correct when **XmNstringDirection** is **XmSTRING_DIRECTION_L_TO_R**; the descriptions for **XmALIGNMENT_BEGINNING** and **XmALIGNMENT_END** are switched when the resource is **XmSTRING_DIRECTION_R_TO_L**.

If the parent is a RowColumn whose **XmNisAligned** resource is True, **XmNalignment** is forced to the same value as the RowColumn's **XmNentryAlignment** if the RowColumn's **XmNrowColumnType** is **XmWORK_AREA** or if the gadget is a subclass of **XmLabelGadget**. Otherwise, the default is **XmALIGNMENT_CENTER**.

XmNfontList Specifies the font of the text used in the gadget. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the **XmBulletinBoard**, **VendorShell**, or **XmMenuShell** widget class. If such an ancestor is found, the font list is initialized to the **XmNbuttonFontList** (for button gadget subclasses) or **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on the creation and structure of a font list.

XmNlabelInsensitivePixmap
Specifies a pixmap used as the button face if **XmNlabelType** is **XmPIXMAP** and the button is insensitive. The default value, **XmUNSPECIFIED_PIXMAP**, displays an empty label.

XmNlabelPixmap
Specifies the pixmap when **XmNlabelType** is **XmPIXMAP**. The default value, **XmUNSPECIFIED_PIXMAP**, displays an empty label.

XmNlabelString
Specifies the compound string when **XmNlabelType** is **XmSTRING**. If the value of this resource is NULL, it is initialized to name of the gadget converted to a compound string. Refer to **XmString(3X)** for more information on the creation and the structure of compound strings.

XmNlabelType
Specifies the label type.

XmSTRING	Text displays **XmNlabelString**
XmPIXMAP	Icon data in pixmap displays **XmNlabelPixmap** or **XmNlabelInsensitivePixmap**

XmNmarginBottom
Specifies the amount of spacing between the bottom of the label text and the top of the bottom margin specified by **XmNmarginHeight**. This may be modified by LabelGadget's subclasses. For example, CascadeButtonGadget may increase this field to make room for the cascade pixmap.

XmNmarginHeight
Specifies an equal amount of spacing above the margin defined by **XmNmarginTop** and below the margin defined by **XmNmarginBottom**. **XmNmarginHeight** specifies the amount of spacing between the top edge of the margin set by **XmNmarginTop** and the bottom edge of the top shadow, and the amount of spacing between the bottom edge of the margin specified by **XmNmarginBottom** and the top edge of the bottom shadow.

XmNmarginLeft
Specifies the amount of spacing between the left edge of the label text and the right side of the left margin (specified by **XmNmarginWidth**). This may be modified by LabelGadget's subclasses. For example, ToggleButtonGadget may increase this field to make room for the toggle indicator and for spacing between the indicator and label. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginRight
Specifies the amount of spacing between the right edge of the label text and the left side of the right margin (specified by **XmNmarginWidth**). This may be modified by LabelGadget's subclasses. For example, CascadeButtonGadget may increase this field to make room for the cascade pixmap. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginTop

Specifies the amount of spacing between the top of the label text and the bottom of the top margin specified by **XmNmarginHeight**. This may be modified by LabelGadget's subclasses. For example, CascadeButtonGadget may increase this field to make room for the cascade pixmap.

XmNmarginWidth

Specifies an equal amount of spacing to the left of the margin defined by **XmNmarginLeft** and to the right of the margin defined by **XmNmarginRight**. **XmNmarginWidth** specifies the amount of spacing between the left edge of the margin set by **XmNmarginLeft** and the right edge of the left shadow, and the amount of spacing between the right edge of the margin specified by **XmNmarginRight** and the left edge of the right shadow.

XmNmnemonic

Provides the user with an alternate means of activating a button. A button in a MenuBar, a Popup MenuPane, or a Pulldown MenuPane can have a mnemonic.

This resource contains a keysym as listed in the X11 keysym table. The first character in the label string that exactly matches the mnemonic in the character set specified in **XmNmnemonicCharSet** is underlined when the button is displayed.

When a mnemonic has been specified, the user activates the button by pressing the mnemonic key while the button is visible. If the button is a CascadeButtonGadget in a MenuBar and the MenuBar does not have the focus, the user must use the **MAlt** modifier while pressing the mnemonic. The user can activate the button by pressing either the shifted or the unshifted mnemonic key.

XmNmnemonicCharSet

Specifies the character set of the mnemonic for the label. The default is **XmFONTLIST_DEFAULT_TAG**.

XmNrecomputeSize

Specifies a Boolean value that indicates whether the gadget shrinks or expands to accommodate its contents (label string or pixmap) as a result of an **XtSetValues** resource value that would change the size of the gadget. If True, the gadget shrinks or expands to exactly fit the label string or pixmap. If False, the gadget never attempts to change size on its own.

XmNstringDirection

Specifies the direction in which the string is to be drawn.

XmSTRING_DIRECTION_L_TO_R	Left to right
XmSTRING_DIRECTION_R_TO_L	Right to left

The default for this resource is determined at creation time. If no value is specified for this resource and the widget's parent is a manager, the value is inherited from the parent; otherwise, it defaults to **XmSTRING_DIRECTION_L_TO_R**.

Inherited Resources

LabelGadget inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	G
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	G
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	0 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	0 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	G
XmNtraversalOn XmCTraversalOn	False Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

RectObj Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	0 Dimension	N/A
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Behavior

XmLabelGadget includes behavior from **XmGadget**. Additional **XmLabelGadget** behavior is described in the following list:

BTransfer Press:
Drags the contents of a LabelGadget, identified with **BTransfer**. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT** for a label type of **XmSTRING**; otherwise it is set to, **PIXMAP** if the label type is **XmPIXMAP**. This action is undefined for LabelGadgets used in a menu system.

KHelp:
In a Popup or Pulldown MenuPane, unposts all menus in the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget that had the focus before the menu system was entered. Calls the callbacks for

XmNhelpCallback if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MAny KCancel:

In a MenuBar, disarms the CascadeButton and the menu and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget that had the focus before the menu was entered.

In a toplevel Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu.

In a Popup MenuPane, unposts the menu and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget from which the menu was posted.

KDown: If the current menu item has a submenu and is in a MenuBar, then this action posts the submenu, disarms the current menu item, and arms the submenu's first traversable menu item.

If the current menu item is in a MenuPane, then this action disarms the current menu item and arms the item below it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's bottom edge, then this action wraps to the topmost menu item in the column to the right, if one exists. When the current menu item is at the bottom, rightmost corner of the MenuPane, then this action wraps to the tear-off control, if present, or to the top, leftmost menu item.

KLeft: When the current menu item is in a MenuBar, then this action disarms the current item and arms the MenuBar item to the left. This action wraps within the MenuBar.

In MenuPanes, if the current menu item is not at the left edge of a MenuPane, this action disarms the current item and arms the item to its left. If the current menu item is at the left edge of a submenu attached to a MenuBar item, then this action unposts the submenu and traverses to the MenuBar item to the left, wrapping if necessary. If that MenuBar item has a submenu, it posts the submenu and arms the first traversable item in the submenu. If the current menu item is at the left edge of a submenu not directly attached to a MenuBar item, then this action unposts the current submenu only.

In Popup or Torn-off MenuPanes, when the current menu item is at the left edge, this action wraps within the MenuPane. If the current menu item is at the left edge of the MenuPane and not in the top row, this action wraps to the rightmost menu item in the row above. If the current menu item is in the upper, leftmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, rightmost menu item in the MenuPane.

KRight: If the current menu item is in a MenuBar, then this action disarms the current item and arms the MenuBar item to the right. This action wraps within the MenuBar.

In MenuPanes, if the current menu item is a CascadeButton, then this action posts its associated submenu. If the current menu item is not a CascadeButton and is not at the right edge of a MenuPane, this action disarms the current item and arms the item to its right, wrapping if necessary. If the current menu item is not a CascadeButton and is at the right edge of a submenu that is a descendent of a MenuBar, then this action unposts all submenus and traverses to the MenuBar item to the right. If that MenuBar item has a submenu, it posts the submenu and arms the submenu's first traversable item.

In Popup or Torn-off menus, if the current menu item is not a CascadeButton and is at the right edge of a row (except the bottom row), this action wraps to the leftmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, rightmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, leftmost menu item of the MenuPane.

KUp: When the current menu item is in a MenuPane, then this action disarms the current menu item and arms the item above it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's top edge, then this action wraps to the bottommost menu item in the column to the left, if one exists. When the current menu item is at the top, leftmost corner of the MenuPane, then this action wraps to the tear-off control, if present, or to the bottom, rightmost menu item.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Object(3X), **RectObj(3X)**, **XmCreateLabelGadget(3X)**, **XmFontListCreate(3X)**, **XmStringCreate(3X)**, **XmStringCreateLtoR(3X)**, and **XmGadget(3X)**.

XmList—The List widget class

Synopsis

#include <Xm/List.h>

Description

List allows a user to select one or more items from a group of choices. Items are selected from the list in a variety of ways, using both the pointer and the keyboard. List operates on an array of compound strings that are defined by the application. Each compound string becomes an item in the List, with the first compound string becoming the item in position 1, the second becoming the item in position 2, and so on.

Specifying the number of items that are visible sets the size of the List. If the number of visible items is not specified, the height of the list controls the number of visible items. Each item assumes the height of the tallest element in the list. To create a list that allows the user to scroll easily through a large number of items, use the **XmCreateScrolledList** convenience function.

To select items, move the pointer or cursor to the desired item and press the **BSelect** mouse button or the key defined as **KSelect**. There are several styles of selection behavior, and they all highlight the selected item or items by displaying them in inverse colors. An appropriate callback is invoked to notify the application of the user's choice. The application then takes whatever action is required for the specified selection. When a List is insensitive, all of the list items are displayed in a stippled fill pattern.

Selection

Each list has one of four selection models:

- Single Select
- Browse Select
- Multiple Select
- Extended Select

In Single Select and Browse Select, at most one item is selected at a time. In Single Select, pressing **BSelect** on an item toggles its selection state and deselects any other selected item. In Browse Select, pressing **BSelect** on an item selects it and deselects any other selected item; dragging **BSelect** moves the selection as the pointer is moved. Releasing **BSelect** on an item moves the location cursor to that item.

In Multiple Select, any number of items can be selected at a time. Pressing **BSelect** on an item toggles its selection state but does not deselect any other selected items.

In Extended Select, any number of items can be selected at a time, and the user can easily select ranges of items. Pressing **BSelect** on an item selects it and deselects any other selected item. Dragging **BSelect** or pressing or dragging **BExtend** following a **BSelect** action selects all items between the item under the pointer and the item on which **BSelect** was pressed. This action also deselects any other selected items outside that range.

Extended Select also allows the user to select and deselect discontiguous ranges of items. Pressing **BToggle** on an item toggles its selection state but does not deselect any other selected items. Dragging **BToggle** or pressing or dragging **BExtend** following a **BToggle** action sets the selection state of all items between the item under the pointer and the item on which **BToggle** was pressed to the state of the item on which **BToggle** was pressed. This action does not deselect any other selected items outside that range.

All selection operations available from the mouse are also available from the keyboard. List has two keyboard selection modes, Normal Mode and Add Mode. In Normal Mode, navigation operations and **KSelect** select the item at the location cursor and deselect any other selected items. In Add Mode, navigation operations have no effect on selection, and **KSelect** toggles the selection state of the item at the location cursor without deselecting any other selected items, except in Single Select.

Single and Multiple Select use Add Mode, and Browse Select uses Normal Mode.

Extended Select can use either mode; the user changes modes by pressing **KAddMode**. In Extended Select Normal Mode, pressing **KSelect** has the same effect as pressing **BSelect**; **KExtend** and shifted navigation have the same effect as pressing **BExtend** following a **BSelect** action. In Extended Select Add Mode, pressing **KSelect** has the same effect as pressing **BToggle**; **KExtend** and shifted navigation have the same effect as pressing **BExtend** following a **BToggle** action.

Normal Mode is indicated by a solid location cursor, and Add Mode is indicated by a dashed location cursor.

Classes

List inherits behavior and resources from **Core** and **XmPrimitive** classes.

The class pointer is **xmListWidgetClass**.

The class name is **XmList**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmList Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNautomaticSelection XmCAutomaticSelection	False Boolean	CSG
XmNbrowseSelectionCallback XmCCallback	NULL XtCallbackList	C
XmNdefaultActionCallback XmCCallback	NULL XtCallbackList	C
XmNdoubleClickInterval XmCDoubleClickInterval	dynamic int	CSG
XmNextendedSelectionCallback XmCCallback	NULL XtCallbackList	C
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNitemCount XmCItemCount	0 int	CSG
XmNitems XmCItems	NULL XmStringTable	CSG
XmNlistMarginHeight XmCListMarginHeight	0 Dimension	CSG
XmNlistMarginWidth XmCListMarginWidth	0 Dimension	CSG
XmNlistSizePolicy XmCListSizePolicy	XmVARIABLE unsigned char	CG
XmNlistSpacing XmCListSpacing	0 Dimension	CSG
XmNmultipleSelectionCallback XmCCallback	NULL XtCallbackList	C
XmNscrollBarDisplayPolicy XmCScrollBarDisplayPolicy	XmAS_NEEDED unsigned char	CSG
XmNselectedItemCount XmCSelectedItemCount	0 int	CSG

Name Class	Default Type	Access
XmNselectedItems XmCSelectedItems	NULL XmStringTable	CSG
XmNselectionPolicy XmCSelectionPolicy	XmBROWSE_SELECT unsigned char	CSG
XmNsingleSelectionCallback XmCCallback	NULL XtCallbackList	C
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG
XmNtopItemPosition XmCTopItemPosition	1 int	CSG
XmNvisibleItemCount XmCVisibleItemCount	dynamic int	CSG

XmNautomaticSelection

Invokes either **XmNbrowseSelectionCallback** or **XmNextendedSelectionCallback** when BSelect is pressed and the items that are shown as selected change if the value is True and the selection mode is either **XmBROWSE_SELECT** or **XmEXTENDED_SELECT** respectively. If False, no selection callbacks are invoked until the user releases the mouse button. See **Behavior** for further details on the interaction of this resource with the selection modes.

XmNbrowseSelectionCallback

Specifies a list of callbacks that is called when an item is selected in the browse selection mode. The reason is **XmCR_BROWSE_SELECT**.

XmNdefaultActionCallback

Specifies a list of callbacks that is called when an item is double clicked or **KActivate** is pressed. The reason is **XmCR_DEFAULT_ACTION**.

XmNdoubleClickInterval

If a button click is followed by another button click within the time span specified by this resource (in milliseconds), the button clicks are considered a double-click action, rather than two single-click actions. The value must not be negative. The default value is the display's multiclick time.

XmNextendedSelectionCallback
Specifies a list of callbacks that is called when items are selected using the extended selection mode. The reason is **XmCR_EXTENDED_SELECT**.

XmNfontList Specifies the font list associated with the list items. This is used in conjunction with the **XmNvisibleItemCount** resource to determine the height of the List widget. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard or VendorShell widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on a font list structure.

XmNitemCount
Specifies the total number of items. The value must be the number of items in **XmNitems** and must not be negative. It is automatically updated by the list whenever an item is added to or deleted from the list.

XmNitems Points to an array of compound strings that are to be displayed as the list items. Refer to **XmString(3X)** for more information on the creation and structure of compound strings. **XtGetValues** for this resource returns the list items themselves, not a copy of the list items. The application must not free the returned items.

XmNlistMarginHeight
Specifies the height of the margin between the list border and the items.

XmNlistMarginWidth
Specifies the width of the margin between the list border and the items.

XmNlistSizePolicy
Controls the reaction of the List when an item grows horizontally beyond the current size of the list work area. If the value is **XmCONSTANT**, the list viewing area does not grow, and a horizontal ScrollBar is added for a ScrolledList. If this resource is set to **XmVARIABLE**, the List grows to match the size of the longest item, and no horizontal ScrollBar appears.

When the value of this resource is **XmRESIZE_IF_POSSIBLE**, the List attempts to grow or shrink to match the width of the widest item. If it cannot grow to match the widest size, a horizontal ScrollBar is added for a ScrolledList if the longest item is wider than the list viewing area.

The size policy must be set at the time the List widget is created. It cannot be changed at a later time through **XtSetValues**.

XmNlistSpacing

Specifies the spacing between list items. This spacing increases by the value of the **XmNhighlightThickness** resource in Primitive.

XmNmultipleSelectionCallback

Specifies a list of callbacks that is called when an item is selected in multiple selection mode. The reason is **XmCR_MULTIPLE_SELECT**.

XmNscrollBarDisplayPolicy

Controls the display of vertical ScrollBars in a ScrolledList. When the value of this resource is **XmAS_NEEDED**, a vertical ScrollBar is displayed only when the number of items in the List exceeds the number of visible items. When the value is **XmSTATIC**, a vertical ScrollBar is always displayed.

XmNselectedItemCount

Specifies the number of strings in the selected items list. The value must be the number of items in **XmNselectedItems** and must not be negative.

XmNselectedItems

Points to an array of compound strings that represents the list items that are currently selected, either by the user or by the application. **XtGetValues** for this resource returns the list items themselves, not a copy of the list items. The application must not free the returned items.

Setting **XmNselectedItems** selects those list items that exactly match items in the given **XmNselectedItems** list. There may be additional items in **XmNselectedItems** that do not match items in the list. These items remain until **XmNselectedItems** is updated. If **XmNitems** is changed such that the list now contains items matching previously unmatched items in **XmNselectedItems**, those new items will also appear selected.

Any user interaction with the list that causes at least one item to be selected or deselected and any call to **XmListDeselectAllItems**, **XmListDeselectAllItems**, **XmListDeselectPos**, **XmListSelectItem**, **XmListSelectPos**, or **XmListUpdateSelectedList** cause **XmNselectedItems** to be updated immediately to exactly reflect the visual state of the list. Calls to any other **XmList** functions do not affect **XmNselectedItems**.

XmNselectionPolicy

Defines the interpretation of the selection action. This can be one of the following:

XmSINGLE_SELECT
Allows only single selections

XmMULTIPLE_SELECT
Allows multiple selections

XmEXTENDED_SELECT
Allows extended selections

XmBROWSE_SELECT
Allows drag-and-browse functionality

XmNsingleSelectionCallback

Specifies a list of callbacks that is called when an item is selected in single selection mode. The reason is **XmCR_SINGLE_SELECT**.

XmNstringDirection

Specifies the initial direction to draw the string. The values for this resource are **XmSTRING_DIRECTION_L_TO_R** and **XmSTRING_DIRECTION_R_TO_L**. The value of this resource is determined at creation time. If the widget's parent is a manager, this value is inherited from the widget's parent, otherwise it is set to **XmSTRING_DIRECTION_L_TO_R**.

XmNtopItemPosition

Specifies the position of the item that is the first visible item in the list. Setting this resource is equivalent to calling the **XmListSetPos** function. The position of the first item in the list is 1; the position of the second item is 2; and so on. A position of 0 (zero) specifies the last item in the list. The value must not be negative.

XmNvisibleItemCount

Specifies the number of items that can fit in the visible space of the list work area. The List uses this value to determine its height. The value must be greater than 0 (zero).

Inherited Resources

List inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

List defines a new callback structure. The application must first look at the reason field and use only the structure members that are valid for that particular reason, because not all fields are relevant for every possible reason. The callback structure is defined as follows:

```
typedef struct
{
    int             reason;
    XEvent          *event;
    XmString        item;
    int             item_length;
    int             item_position;
    XmString        *selected_items;
    int             selected_item_count;
    int             *selected_item_positions;
    char            selection_type;
} XmListCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback. It can be NULL.

item The last item selected at the time of the *event* that caused the callback. *item* points to a temporary storage space that is reused after the callback is finished. Therefore, if an application needs to save the item, it should copy the item into its own data space.

item_length The length in bytes of *item*.

item_position
The position of *item* in the List's **XmNitems** array.

selected_items
: A list of items selected at the time of the *event* that caused the callback. *selected_items* points to a temporary storage space that is reused after the callback is finished. Therefore, if an application needs to save the selected list, it should copy the list into its own data space.

selected_item_count
: The number of items in the *selected_items* list. This number must be nonnegative.

selected_item_positions
: An array of integers, one for each selected item, representing the position of each selected item in the List's **XmNitems** array. *selected_item_positions* points to a temporary storage space that is reused after the callback is finished. Therefore, if an application needs to save this array, it should copy the array into its own data space.

selection_type
: Indicates that the most recent extended selection was the initial selection (**XmINITIAL**), a modification of an existing selection (**XmMODIFICATION**), or an additional noncontiguous selection (**XmADDITION**).

The following table describes the reasons for which the individual callback structure fields are valid.

Reason	Valid Fields
XmCR_SINGLE_SELECT	*reason, event, item, item_length, item_position*
XmCR_DEFAULT_ACTION	*reason, event, item, item_length, item_position, selected_items, selected_item_count, selected_item_positions*
XmCR_BROWSE_SELECT	*reason, event, item, item_length, item_position*
XmCR_MULTIPLE_SELECT	*reason, event, item, item_length, item_position, selected_items, selected_item_count, selected_item_positions*
XmCR_EXTENDED_SELECT	*reason, event, item, item_length, item_position, selected_items, selected_item_count, selected_item_positions, selection_type*

Translations

XmList includes translations from Primitive. The **XmList** translations are described in the following list. These translations may not directly correspond to a translation table.

BSelect Press: **ListBeginSelect()**

BSelect Motion: **ListButtonMotion()**

BSelect Release: **ListEndSelect()**

BExtend Press: **ListBeginExtend()**

BExtend Motion: **ListButtonMotion()**

BExtend Release: **ListEndExtend()**

BToggle Press: **ListBeginToggle()**

BToggle Motion: **ListButtonMotion()**

BToggle Release: **ListEndToggle()**

BTransfer Press: **ListProcessDrag()**

KUp: **ListPrevItem()**

MShift KUp: **ListExtendPrevItem()**

KDown: **ListNextItem()**

MShift KDown: **ListExtendNextItem()**

KLeft:	**ListLeftChar()**
MCtrl KLeft:	**ListLeftPage()**
KRight:	**ListRightChar()**
MCtrl KRight:	**ListRightPage()**
KPageUp:	**ListPrevPage()**
KPageDown:	**ListNextPage()**
KPageLeft:	**ListLeftPage()**
KPageRight:	**ListRightPage()**
KBeginLine:	**ListBeginLine()**
KEndLine:	**ListEndLine()**
KBeginData:	**ListBeginData()**
MShift KBeginData:	**ListBeginDataExtend()**
KEndData:	**ListEndData()**
MShift KEndData:	**ListEndDataExtend()**
KAddMode:	**ListAddMode()**
KActivate:	**ListKbdActivate()**
KCopy Press:	**ListCopyToClipboard()**
KSelect Press:	**ListKbdBeginSelect()**
KSelect Release:	**ListKbdEndSelect()**
KExtend Press:	**ListKbdBeginExtend()**
KExtend Release:	**ListKbdEndExtend()**
MAny KCancel:	**ListKbdCancel()**
KSelectAll:	**ListKbdSelectAll()**
KDeselectAll:	**ListKbdDeSelectAll()**
KHelp:	**PrimitiveHelp()**

KNextField	**PrimitiveNextTabGroup()**
KPrevField	**PrimitivePrevTabGroup()**

Action Routines

The **XmList** action routines are described in the following list. The current selection is always shown with inverted colors.

ListAddMode():

Toggles the state of Add Mode for keyboard selection.

ListBeginData():

Moves the location cursor to the first item in the list. In Normal Mode, this also deselects any current selection, selects the first item in the list, and calls the appropriate selection callbacks (**XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**).

ListBeginDataExtend():

If **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT** or **XmEXTENDED_SELECT**, this action moves the location cursor to the first item in the list.

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action does the following: If an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done; changes the selection state of the first item and all items between it and the current anchor point to the state of the item at the current anchor point; calls the **XmNextendedSelectionCallback** callbacks.

ListBeginExtend():

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action does the following: If an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done, and changes the selection state of the item under the pointer and all items between it and the current anchor point to the state of the item at the current anchor point. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListBeginLine():
Moves the horizontal scroll region to the beginning of the line.

ListBeginSelect():
If **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, deselects any current selection and toggles the selection state of the item under the pointer.

If **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, deselects any current selection and selects the item under the pointer. If **XmNautomaticSelection** is set to True, this action calls the **XmNbrowseSelectionCallback** callbacks.

If **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT**, toggles the selection state of the item under the pointer. Any previous selections remain.

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action deselects any current selection, selects the item under the pointer, and sets the current anchor at that item. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListBeginToggle():
If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action moves the current anchor to the item under the pointer without changing the current selection. If the item is unselected, this action selects it; if the item is selected, this action unselects it. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListButtonMotion():
If **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, this action deselects any current selection and selects the item under the pointer. If **XmNautomaticSelection** is set to True and the pointer has entered a new list item, this action calls the **XmNbrowseSelectionCallback** callbacks.

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action does the following: If an extended selection is being made and an extended selection has previously been made from the current anchor point, restores the selection state of the items in that range to their state before the previous extended selection was done and changes the selection state of the item under the pointer and all

items between it and the current anchor point to the state of the item at the current anchor point. If **XmNautomaticSelection** is set to True and the pointer has entered a new list item, calls the **XmNextendedSelectionCallback** callbacks.

If the pointer leaves a scrolled list, this action scrolls the list in the direction of the pointer motion.

ListCopyToClipboard()
Copies the content of the selected items to the clipboard as a single compound string with each item separated by a newline.

ListEndData():
Moves the location cursor to the last item in the list. In Normal Mode, this also deselects any current selection, selects the last item in the list, and calls the appropriate selection callbacks (**XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**).

ListEndDataExtend():
If **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT** or **XmEXTENDED_SELECT**, this action moves the location cursor to the last item in the list.

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action does the following: If an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done; changes the selection state of the last item and all items between it and the current anchor point to the state of the item at the current anchor point; calls the **XmNextendedSelectionCallback** callbacks.

ListEndExtend():
If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action moves the location cursor to the last item selected or deselected and, if **XmNautomaticSelection** is set to False, calls the **XmNextendedSelectionCallback** callbacks.

ListEndLine():
Moves the horizontal scroll region to the end of the line.

ListEndSelect():

If **XmNselectionPolicy** is set to **XmSINGLE_SELECT** or **XmMULTIPLE_SELECT**, this action moves the location cursor to the last item selected or deselected and calls the appropriate selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT**).

If **XmNselectionPolicy** is set to **XmBROWSE_SELECT** or **XmEXTENDED_SELECT**, moves the location cursor to the last item selected or deselected and, if **XmNautomaticSelection** is set to False, calls the appropriate selection callbacks (**XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**).

ListEndToggle():

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, moves the location cursor to the last item selected or deselected and, if **XmNautomaticSelection** is set to False, calls the **XmNextendedSelectionCallback** callbacks.

ListExtendNextItem():

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action does the following: If an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done; moves the location cursor to the next item and changes the selection state of the item and all items between it and the current anchor point to the state of the item at the current anchor point; calls the **XmNextendedSelectionCallback** callbacks.

ListExtendPrevItem():

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action does the following: If an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done; moves the location cursor to the previous item and changes the selection state of the item and all items between it and the current anchor point to the state of the item at the current anchor point; calls the **XmNextendedSelectionCallback** callbacks.

ListScrollCursorVertically(*percentage*):
Scrolls the line containing the insertion cursor vertically to an intermediate position in the visible window based on an input percentage. A value of 0 (zero) indicates the top of the window; a value of 100, the bottom of the window. If this action is called with no argument, the line containing the insertion cursor is scrolled vertically to a new position designated by the *y* event passed to the routine.

ListKbdActivate():
Calls the callbacks for **XmNdefaultActionCallback**. If the List's parent is a manager, this action passes the event to the parent.

ListKbdBeginExtend():
If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, does the following: If an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done; changes the selection state of the item at the location cursor and all items between it and the current anchor point to the state of the item at the current anchor point. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListKbdBeginSelect():
If the **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, deselects any current selection and toggles the state of the item at the location cursor.

If the **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, deselects any current selection and selects the item at the location cursor. If **XmNautomaticSelection** is set to True, this action calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT**, toggles the selection state of the item at the location cursor. Any previous selections remain.

If the **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, moves the current anchor to the item at the location cursor. In Normal Mode, this action deselects any current selection and selects the item at the location cursor. In Add Mode, this action toggles the selection state of the item at the location cursor and leaves the current selection unchanged. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListKbdCancel():
If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** and an extended selection is being made from the current anchor point, this action cancels the new selection and restores the selection state of the items in that range to their state before the extended selection was done. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks; otherwise, if the parent is a manager, it passes the event to the parent.

ListKbdDeSelectAll():
If the **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, **XmMULTIPLE_SELECT**, or **XmEXTENDED_SELECT** in Add Mode, this action deselects all items in the list. If the **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** in Normal Mode, this action deselects all items in the list (except the item at the location cursor if the shell's **XmNkeyboardFocusPolicy** is **XmEXPLICIT**). This action also calls the appropriate selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT**, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**).

ListKbdEndExtend():
If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** and if **XmNautomaticSelection** is set to False, this action calls the **XmNextendedSelectionCallback** callbacks.

ListKbdEndSelect():
If **XmNselectionPolicy** is set to **XmSINGLE_SELECT** or **XmMULTIPLE_SELECT** or if **XmNautomaticSelection** is set to False, calls the appropriate selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, **XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT**, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**).

ListKbdSelectAll():
If **XmNselectionPolicy** is set to **XmSINGLE_SELECT** or **XmBROWSE_SELECT**, this action selects the item at the location cursor. If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** or **XmMULTIPLE_SELECT**, it selects all items in the list. This action also calls the appropriate

selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, **XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT**, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**).

ListLeftChar():
Scrolls the list one character to the left.

ListLeftPage():
Scrolls the list one page to the left.

ListNextItem():
Moves the location cursor to the next item in the list.

If the **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, this action also selects the next item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action in Normal Mode also selects the next item, deselects any current selection, moves the current anchor to the next item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode, this action does not affect the selection or the anchor.

ListNextPage():
Scrolls the list to the next page, moving the location cursor to a new item.

If the **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, this action also selects the new item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action in Normal Mode also selects the new item, deselects any current selection, moves the current anchor to the new item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode, this action does not affect the selection or the anchor.

ListPrevItem():
Moves the location cursor to the previous item in the list.

If the **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, this action also selects the previous item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action in Normal Mode also selects the previous item, deselects any current selection, moves the current anchor to the previous item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode, this action does not affect the selection or the anchor.

ListPrevPage():

Scrolls the list to the previous page, moving the location cursor to a new item.

If the **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, this action also selects the new item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, this action in Normal Mode also selects the new item, deselects any current selection, moves the current anchor to the new item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode this action does not affect the selection or the anchor.

ListProcessDrag():

Drags the contents of one or more selected list items. Each item is separated by a newline. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT**; and the **XmNclientData** resource is set to the index of the item in the list. If **BTransfer** is pressed on an unselected item, this action drags only that item, excluding any other selected items.

ListRightChar():

Scrolls the list one character to the right.

ListRightPage():

Scrolls the list one page to the right.

PrimitiveHelp():

Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

PrimitiveNextTabGroup():

Moves the focus to the first item contained within the next tab group. If the current tab group is the last entry in the tab group list,

it wraps to the beginning of the tab group list.

PrimitivePrevTabGroup():
Moves the focus to the first item contained within the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

Additional Behavior

The List widget has the following additional behavior:

<Double Click>
If a button click is followed by another button click within the time span specified by the display's multiclick time, the List interprets that as a double click and calls the callbacks for **XmNdefaultActionCallback**. The item's colors invert to indicate that it is selected. The **XmNdoubleClickInterval** resource can be used to specify a time span that overrides the display's multi-click time.

<FocusIn>: If the focus policy is Explicit, this action sets the focus and draw the location cursor.

<FocusOut>: If the focus policy is Explicit, this action removes the focus and erase the location cursor.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateList(3X)**, **XmCreateScrolledList(3X)**, **XmFontListAppendEntry(3X)**, **XmListAddItem(3X)**, **XmListAddItems(3X)**, **XmListAddItemUnselected(3X)**, **XmListAddItemsUnselected(3X)**, **XmListDeleteAllItems(3X)**, **XmListDeleteItem(3X)**, **XmListDeleteItems(3X)**, **XmListDeleteItemsPos(3X)**, **XmListDeletePos(3X)**, **XmListDeletePositions(3X)**, **XmListDeselectAllItems(3X)**, **XmListDeselectItem(3X)**, **XmListDeselectPos(3X)**, **XmListGetKbdItemPos** **XmListGetMatchPos(3X)**, **XmListGetSelectedPos(3X)**, **XmListItemExists(3X)**, **XmListItemPos(3X)**, **XmListPosToBounds(3X)**, **XmListReplaceItems(3X)**, **XmListReplaceItemsPos(3X)**, **XmListReplaceItemsPositions(3X)**, **XmListReplaceItemsPosUnselected(3X)**, **XmListReplaceItemsUnselected(3X)**, **XmListSelectItem(3X)**, **XmListSelectPos(3X)**, **XmListSetAddMode(3X)**, **XmListSetBottomItem(3X)**, **XmListSetBottomPos(3X)**, **XmListSetHorizPos(3X)**, **XmListSetItem(3X)**, **XmListSetKbdItemPos(3X)**, **XmListSetPos(3X)**, **XmListUpdateSelectedList(3X)**, **XmListYToPos(3X)**, **XmPrimitive(3X)** and **XmStringCreate(3X)**.

XmListAddItem—A List function that adds an item to the list

Synopsis

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

void XmListAddItem (widget, item, position)
        Widget      widget;
        XmString    item;
        int         position;
```

Description

XmListAddItem adds an item to the list at the given position. When the item is inserted into the list, it is compared with the current **XmNselectedItems** list. If the new item matches an item on the selected list, it appears selected.

widget Specifies the ID of the List to which an item is added.

item Specifies the item to be added to the list.

position Specifies the position of the new item in the list. A value of 1 makes the new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the new item the last item in the list.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListAddItemUnselected—A List function that adds an item to the list

Synopsis

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

void XmListAddItemUnselected (widget, item, position)
        Widget      widget;
        XmString    item;
        int         position;
```

Description

XmListAddItemUnselected adds an item to the list at the given position. The item does not appear selected, even if it matches an item in the current **XmNselectedItems** list.

widget Specifies the ID of the List from whose list an item is added.

item Specifies the item to be added to the list.

position Specifies the position of the new item in the list. A value of 1 makes the new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the new item the last item in the list.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListAddItems—A List function that adds items to the list

Synopsis

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

void XmListAddItems (widget, items, item_count, position)
        Widget      widget;
        XmString    *items;
        int         item_count;
        int         position;
```

Description

XmListAddItems adds the specified items to the list at the given position. The first *item_count* items of the *items* array are added to the list. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. If any of the new items matches an item on the selected list, it appears selected.

widget Specifies the ID of the List to which an item is added.

items Specifies a pointer to the items to be added to the list.

item_count Specifies the number of items in *items*. This number must be nonnegative.

position Specifies the position of the first new item in the list. A value of 1 makes the first new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the first new item follow the last item in the list.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListAddItemsUnselected—A List function that adds items to a list

Synopsis

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

void XmListAddItemsUnselected (widget, items, item_count, position)
        Widget      widget;
        XmString    *items;
        int         item_count;
        int         position;
```

Description

XmListAddItemsUnselected adds the specified items to the list at the given position. The inserted items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget Specifies the ID of the List widget to add items to.

items Specifies a pointer to the items to be added to the list.

item_count Specifies the number of elements in *items*. This number must be nonnegative.

position Specifies the position of the first new item in the list. A value of 1 makes the first new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the first new item follow the last item of the list.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeleteAllItems—A List function that deletes all items from the list

Synopsis

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

void XmListDeleteAllItems (widget)
        Widget          widget;
```

Description

XmListDeleteAllItems deletes all items from the list.

widget Specifies the ID of the List from whose list the items are deleted

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeleteItem—A List function that deletes an item from the list

Synopsis

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

void XmListDeleteItem (widget, item)
        Widget      widget;
        XmString    item;
```

Description

XmListDeleteItem deletes the first item in the list that matches *item*. A warning message appears if the item does not exist.

widget Specifies the ID of the List from whose list an item is deleted

item Specifies the text of the item to be deleted from the list

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeleteItems—A List function that deletes items from the list

Synopsis

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

void XmListDeleteItems (widget, items, item_count)
        Widget      widget;
        XmString    *items;
        int         item_count;
```

Description

XmListDeleteItems deletes the specified items from the list. For each element of *items*, the first item in the list that matches that element is deleted. A warning message appears if any of the items do not exist.

widget Specifies the ID of the List from whose list an item is deleted.

items Specifies a pointer to items to be deleted from the list.

item_count Specifies the number of elements in *items* This number must be nonnegative.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeleteItemsPos—A List function that deletes items from the list starting at the given position

Synopsis

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

void XmListDeleteItemsPos (widget, item_count, position)
        Widget      widget;
        int         item_count;
        int         position;
```

Description

XmListDeleteItemsPos deletes the specified number of items from the list starting at the specified position.

widget Specifies the ID of the List from whose list an item is deleted.

item_count Specifies the number of items to be deleted. This number must be nonnegative.

position Specifies the position in the list of the first item to be deleted. A value of 1 indicates that the first deleted item is the first item in the list; a value of 2 indicates that it is the second item; and so on.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeletePos—A List function that deletes an item from a list at a specified position

Synopsis

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

void XmListDeletePos (widget, position)
        Widget      widget;
        int         position;
```

Description

XmListDeletePos deletes an item at a specified position. A warning message appears if the position does not exist.

widget Specifies the ID of the List from which an item is to be deleted.

position Specifies the position of the item to be deleted. A value of 1 indicates that the first item in the list is deleted; a value of 2 indicates that the second item is deleted; and so on. A value of 0 (zero) indicates that the last item in the list is deleted.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeletePositions—A List function that deletes items from a list based on an array of positions

Synopsis

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

void XmListDeletePositions (widget, position_list, position_count)
        Widget      widget;
        int         *position_list;
        int         position_count;
```

Description

XmListDeletePositions deletes noncontiguous items from a list. The function deletes all items whose corresponding positions appear in the *position_list* array. A warning message is displayed if a specified position is invalid; that is, the value is 0 (zero), a negative integer, or a number greater than the number of items in the list.

widget Specifies the ID of the List widget.

position_list Specifies an array of the item positions to be deleted. The position of the first item in the list is 1; the position of the second item is 2; and so on.

position_count Specifies the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeselectAllItems—A List function that unhighlights and removes all items from the selected list

Synopsis

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

void XmListDeselectAllItems (widget)
        Widget          widget;
```

Description

XmListDeselectAllItems unhighlights and removes all items from the selected list.

widget Specifies the ID of the List widget from whose list all selected items are deselected

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeselectItem—A List function that deselects the specified item from the selected list

Synopsis

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

void XmListDeselectItem (widget, item)
        Widget      widget;
        XmString    item;
```

Description

XmListDeselectItem unhighlights and removes from the selected list the first item in the list that matches *item*.

widget Specifies the ID of the List from whose list an item is deselected

item Specifies the item to be deselected from the list

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListDeselectPos—A List function that deselects an item at a specified position in the list

Synopsis

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

void XmListDeselectPos (widget, position)
        Widget      widget;
        int         position;
```

Description

XmListDeselectPos unhighlights the item at the specified position and deletes it from the list of selected items.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be deselected. A value of 1 indicates that the first item in the list is deselected; a value of 2 indicates that the second item is deselected; and so on. A value of 0 (zero) indicates that the last item in the list is deselected.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListGetKbdItemPos—A List function that returns the position of the item at the location cursor

Synopsis

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

int XmListGetKbdItemPos (widget)
        Widget          widget;
```

Description

XmListGetKbdItemPos returns the position of the list item at the location cursor.

widget Specifies the ID of the List widget

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns the position of the current keyboard item. A value of 1 indicates that the location cursor is at the first item of the list; a value of 2 indicates that it is at the second item; and so on. A value of 0 (zero) indicates the List widget is empty.

Related Information

XmList(3X).

XmListGetMatchPos—A List function that returns all instances of an item in the list

Synopsis

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

Boolean XmListGetMatchPos (widget, item, position_list, position_count)
        Widget      widget;
        XmString    item;
        int         **position_list;
        int         *position_count;
```

Description

XmListGetMatchPos is a Boolean function that returns an array of positions where a specified item is found in a List.

widget Specifies the ID of the List widget.

item Specifies the item to search for.

position_list Returns an array of positions at which the item occurs in the List. The position of the first item in the list is 1; the position of the second item is 2; and so on. When the return value is True, **XmListGetMatchPos** allocates memory for this array. The caller is responsible for freeing this memory.

position_count
Returns the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns True if the specified item is present in the list, and False if it is not.

Related Information

XmList(3X).

XmListGetSelectedPos—A List function that returns the position of every selected item in the list

Synopsis

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

Boolean XmListGetSelectedPos (widget, position_list, position_count)
        Widget      widget;
        int         **position_list;
        int         *position_count;
```

Description

XmListGetSelectedPos is a Boolean function that returns an array of the positions of the selected items in a List.

widget Specifies the ID of the List widget.

position_list Returns an array of the positions of the selected items in the List. The position of the first item in the list is 1; the position of the second item is 2; and so on. When the return value is True, **XmListGetSelectedPos** allocates memory for this array. The caller is responsible for freeing this memory.

position_count
Returns the number of elements in the *position_list.*

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns True if the list has any selected items, and False if it does not.

Related Information

XmList(3X).

XmListItemExists—A List function that checks if a specified item is in the list

Synopsis

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

Boolean XmListItemExists (widget, item)
        Widget      widget;
        XmString    item;
```

Description

XmListItemExists is a Boolean function that checks if a specified item is present in the list.

widget Specifies the ID of the List widget

item Specifies the item whose presence is checked

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns True if the specified item is present in the list.

Related Information

XmList(3X).

XmListItemPos—A List function that returns the position of an item in the list

Synopsis

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

int XmListItemPos (widget, item)
        Widget      widget;
        XmString    item;
```

Description

XmListItemPos returns the position of the first instance of the specified item in a list.

widget Specifies the ID of the List widget

item Specifies the item whose position is returned

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns the position in the list of the first instance of the specified item. The position of the first item in the list is 1; the position of the second item is 2; and so on. This function returns 0 (zero) if the item is not found.

Related Information

XmList(3X).

XmListPosSelected—A List function that determines if the list item at a specified position is selected

Synopsis

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

Boolean XmListPosSelected (widget, position)
        Widget      widget;
        int         position;
```

Description

XmPosSelected determines if the list item at the specified position is selected or not.

widget Specifies the ID of the List widget.

position Specifies the position of the list item. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) specifies the last item in the list.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns True if the list item is selected; otherwise, returns False if the item is not selected or the specified position is invalid.

Related Information

XmList(3X).

XmListPosToBounds—A List function that returns the bounding box of an item at a specified position in a list

Synopsis

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

Boolean XmListPosToBounds (widget, position, x, y, width, height)
        Widget      widget;
        int         position;
        Position    *x;
        Position    *y;
        Dimension   *width;
        Dimension   *height;
```

Description

XmListPosToBounds returns the coordinates of an item within a list and the dimensions of its bounding box. The function returns the associated x and y-coordinates of the upper left corner of the bounding box relative to the upper left corner of the List widget, as well as the width and the height of the box. The caller can pass a NULL value for the *x*, *y*, *width*, or *height* parameters to indicate that the return value for that parameter is not requested.

widget Specifies the ID of the List widget.

position Specifies the position of the specified item. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) specifies the last item in the list.

x Specifies a pointer to the returned x-coordinate of the item.

y Specifies the pointer to the returned y-coordinate of the item.

width Specifies the pointer to the returned width of the item.

height Specifies the pointer to the returned height of the item.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

If the item at the specified position is not visible, returns False, and the returned values (if any) are undefined. Otherwise, this function returns True.

Related Information

XmList(3X) and **XmListYToPos(3X)**.

XmListReplaceItems—A List function that replaces the specified elements in the list

Synopsis

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

void XmListReplaceItems (widget, old_items, item_count, new_items)
        Widget      widget;
        XmString    *old_items;
        int         item_count;
        XmString    *new_items;
```

Description

XmListReplaceItems replaces each specified item of the list with a corresponding new item. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. If any of the new items matches an item on the selected list, it appears selected.

widget Specifies the ID of the List widget.

old_items Specifies the items to be replaced.

item_count Specifies the number of items in *old_items* and *new_items*. This number must be nonnegative.

new_items Specifies the replacement items.

Every occurrence of each element of *old_items* is replaced with the corresponding element from *new_items*. That is, the first element of *old_items* is replaced with the first element of *new_items*. The second element of *old_items* is replaced with the second element of *new_items*, and so on until *item_count* is reached.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListReplaceItemsPos—A List function that replaces the specified elements in the list

Synopsis

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

void XmListReplaceItemsPos (widget, new_items, item_count, position)
        Widget      widget;
        XmString    *new_items;
        int         item_count;
        int         position;
```

Description

XmListReplaceItemsPos replaces the specified number of items of the List with new items, starting at the specified position in the List. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. If any of the new items matches an item on the selected list, it appears selected.

widget Specifies the ID of the List widget.

new_items Specifies the replacement items.

item_count Specifies the number of items in *new_items* and the number of items in the list to replace. This number must be nonnegative.

position Specifies the position of the first item in the list to be replaced. A value of 1 indicates that the first item replaced is the first item in the list; a value of 2 indicates that it is the second item; and so on.

Beginning with the item specified in *position*, *item_count* items in the list are replaced with the corresponding elements from *new_items*. That is, the item at *position* is replaced with the first element of *new_items*; the item after *position* is replaced with the second element of *new_items*; and so on, until *item_count* is reached.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListReplaceItemsPosUnselected—A List function that replaces items in a list without selecting the replacement items

Synopsis

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

void XmListReplaceItemsPosUnselected (widget, new_items, item_count, position)
        Widget      widget;
        XmString    *new_items;
        int         item_count;
        int         position;
```

Description

XmListReplaceItemsPosUnselected replaces the specified number of items in the list with new items, starting at the given position. The replacement items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget Specifies the ID of the List widget to replace items in.

new_items Specifies a pointer to the replacement items.

item_count Specifies the number of elements in *new_items* and the number of items in the list to replace. This number must be nonnegative.

position Specifies the position of the first item in the list to be replaced. A value of 1 indicates that the first item replaced is the first item in the list; a value of 2 indicates that it is the second item; and so on.

Beginning with the item specified in *position*, *item_count* items in the list are replaced with the corresponding elements from *new_items*. That is, the item at *position* is replaced with the first element of *new_items*; the item after *position* is replaced with the second element of *new_items*; and so on, until *item_count* is reached.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListReplaceItemsUnselected—A List function that replaces items in a list

Synopsis

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

void XmListReplaceItemsUnselected (widget, old_items, item_count, new_items)
        Widget      widget;
        XmString    *old_items;
        int         item_count;
        XmString    *new_items;
```

Description

XmListReplaceItemsUnselected replaces each specified item in the list with a corresponding new item. The replacement items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget Specifies the ID of the List widget to replace items in.

old_items Specifies a pointer to the list items to be replaced.

item_count Specifies the number of elements in *old_items* and *new_items*. This number must be nonnegative.

new_items Specifies a pointer to the replacement items. Every occurrence of each element of *old_items* is replaced with the corresponding element from *new_items*. That is, the first element of *old_items* is replaced with the first element of *new_items*. The second element of *old_items* is replaced with the second element of *new_items*, and so on until *item_count* is reached. If an element in *old_items* does not exist in the list, the corresponding entry in *new_items* is skipped.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListReplacePositions—A List function that replaces items in a list based on position

Synopsis

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

void XmListReplacePositions (widget, position_list, item_list, item_count)
        Widget      widget;
        int         *position_list;
        XmString    *item_list;
        int         item_count;
```

Description

XmListReplacePositions replaces noncontiguous items in a list. The item at each position specified in *position_list* is replaced with the corresponding entry in *item_list*. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. Any of the new items that match items on the selected list appear selected. A warning message is displayed if a specified position is invalid; that is, the value is 0 (zero), a negative integer, or a number greater than the number of items in the list.

widget Specifies the ID of the List widget.

position_list Specifies an array of the positions of items to be replaced. The position of the first item in the list is 1; the position of the second item is 2; and so on.

item_list Specifies an array of the replacement items.

item_count Specifies the number of elements in *position_list* and *item_list*. This number must be nonnegative.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSelectItem—A List function that selects an item in the list

Synopsis

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

void XmListSelectItem (widget, item, notify)
        Widget      widget;
        XmString    item;
        Boolean     notify;
```

Description

XmListSelectItem highlights and adds to the selected list the first item in the list that matches *item*.

widget Specifies the ID of the List widget from whose list an item is selected.

item Specifies the item to be selected in the List widget.

notify Specifies a Boolean value that when True invokes the selection callback for the current mode. From an application interface view, calling this function with *notify* True is indistinguishable from a user-initiated selection action.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSelectPos—A List function that selects an item at a specified position in the list

Synopsis

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

void XmListSelectPos (widget, position, notify)
        Widget      widget;
        int         position;
        Boolean     notify;
```

Description

XmListSelectPos highlights a List item at the specified position and adds it to the list of selected items.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be selected. A value of 1 indicates that the first item in the list is selected; a value of 2 indicates that the second item is selected; and so on. A value of 0 indicates that the last item in the list is selected.

notify Specifies a Boolean value that when True invokes the selection callback for the current mode. From an application interface view, calling this function with *notify* True is indistinguishable from a user-initiated selection action.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSetAddMode—A List function that sets add mode in the list

Synopsis

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

void XmListSetAddMode (widget, state)
        Widget      widget;
        Boolean     state;
```

Description

XmListSetAddMode allows applications control over Add Mode in the extended selection model.

widget Specifies the ID of the List widget.

state Specifies whether to activate or deactivate Add Mode. If *state* is True, Add Mode is activated. If *state* is False, Add Mode is deactivated.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSetBottomItem—A List function that makes an existing item the last visible item in the list

Synopsis

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

void XmListSetBottomItem (widget, item)
        Widget      widget;
        XmString    item;
```

Description

XmListSetBottomItem makes the first item in the list that matches *item* the last visible item in the list.

widget Specifies the ID of the List widget from whose list an item is made the last visible

item Specifies the item

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSetBottomPos—A List function that makes a specified item the last visible item in the list

Synopsis

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

void XmListSetBottomPos (widget, position)
        Widget      widget;
        int         position;
```

Description

XmListSetBottomPos makes the item at the specified position the last visible item in the List.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be made the last visible item in the list. A value of 1 indicates that the first item in the list is the last visible item; a value of 2 indicates that the second item is the last visible item; and so on. A value of 0 (zero) indicates that the last item in the list is the last visible item.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSetHorizPos—A List function that scrolls to the specified position in the list

Synopsis

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

void XmListSetHorizPos (widget, position)
        Widget      widget;
        int         position;
```

Description

XmListSetHorizPos sets the **XmNvalue** resource of the horizontal ScrollBar to the specified position and updates the visible portion of the list with the new value if the List widget's **XmNlistSizePolicy** is set to **XmCONSTANT** or **XmRESIZE_IF_POSSIBLE** and the horizontal ScrollBar is currently visible. This is equivalent to moving the horizontal ScrollBar to the specified position.

widget Specifies the ID of the List widget

position Specifies the horizontal position

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSetItem—A List function that makes an existing item the first visible item in the list

Synopsis

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

void XmListSetItem (widget, item)
        Widget      widget;
        XmString    item;
```

Description

XmListSetItem makes the first item in the list that matches *item* the first visible item in the list.

widget Specifies the ID of the List widget from whose list an item is made the first visible

item Specifies the item

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListSetKbdItemPos—A List function that sets the location cursor at a specified position

Synopsis

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

Boolean XmListSetKbdItemPos (widget, position)
        Widget      widget;
        int         position;
```

Description

XmListSetKbdItemPos sets the location cursor at the item specified by *position.* This function does not determine if the item at the specified position is selected or not.

widget Specifies the ID of the List widget.

position Specifies the position of the item at which the location cursor is set. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) sets the location cursor at the last item in the list.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns False if no item exists at the specified position or if the list is empty; otherwise, returns True.

Related Information

XmList(3X).

XmListSetPos—A List function that makes the item at the given position the first visible position in the list

Synopsis

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

void XmListSetPos (widget, position)
        Widget      widget;
        int         position;
```

Description

XmListSetPos makes the item at the given position the first visible position in the list.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be made the first visible item in the list. A value of 1 indicates that the first item in the list is the first visible item; a value of 2 indicates that the second item is the first visible item; and so on. A value of 0 (zero) indicates that the last item in the list is the first visible item.

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListUpdateSelectedList—A List function that updates the **XmNselectedItems** resource

Synopsis

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

void XmListUpdateSelectedList (widget)
        Widget          widget;
```

Description

XmListUpdateSelectedList frees the contents of the current **XmNselectedItems** list. The routine traverses the **XmNitems** list and adds each currently selected item to the **XmNselectedItems** list. For each selected item, there is a corresponding entry in the updated **XmNselectedItems** list.

widget Specifies the ID of the List widget to update

For a complete definition of List and its associated resources, see **XmList(3X)**.

Related Information

XmList(3X).

XmListYToPos—A List function that returns the position of the item at a specified y-coordinate

Synopsis

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

int XmListYToPos (widget, y)
        Widget      widget;
        Position    y;
```

Description

XmListYToPos returns the position of the item at the given y-coordinate within the list.

widget Specifies the ID of the List widget

y Specifies the y-coordinate in the list's coordinate system

For a complete definition of List and its associated resources, see **XmList(3X)**.

Return Value

Returns the position of the item at the specified y coordinate. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) indicates that no item exists at the specified y-coordinate.

Related Information

XmList(3X) and **XmListPosToBounds(3X)**.

XmMainWindow—The MainWindow widget class

Synopsis

#include <Xm/MainW.h>

Description

MainWindow provides a standard layout for the primary window of an application. This layout includes a MenuBar, a CommandWindow, a work region, a MessageWindow, and ScrollBars. Any or all of these areas are optional. The work region and ScrollBars in the MainWindow behave identically to the work region and ScrollBars in the ScrolledWindow widget. The user can think of the MainWindow as an extended ScrolledWindow with an optional MenuBar and optional CommandWindow and MessageWindow.

In a fully loaded MainWindow, the MenuBar spans the top of the window horizontally. The CommandWindow spans the MainWindow horizontally just below the MenuBar, and the work region lies below the CommandWindow. The MessageWindow is below the work region. Any space remaining below the MessageWindow is managed in a manner identical to ScrolledWindow. The behavior of ScrolledWindow can be controlled by the ScrolledWindow resources. To create a MainWindow, first create the work region elements, a MenuBar, a CommandWindow, a MessageWindow, a horizontal ScrollBar, and a vertical ScrollBar widget, and then call **XmMainWindowSetAreas** with those widget IDs.

MainWindow can also create three Separator widgets that provide a visual separation of MainWindow's four components. The user can specify resources in a resource file for the automatically created gadgets that contain the MainWindow separators. The name of the first separator gadget is **Separator1**; the second is **Separator2**; and the third is **Separator3**.

Classes

MainWindow inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **ScrolledWindow**.

The class pointer is **xmMainWindowWidgetClass**.

The class name is **XmMainWindow**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by

class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmMainWindow Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNcommandWindow XmCCommandWindow	NULL Widget	CSG
XmNcommandWindowLocation XmCCommandWindowLocation	ABOVE (SeeDesc.) unsigned char	CG
XmNmainWindowMarginHeight XmCMainWindowMarginHeight	0 Dimension	CSG
XmNmainWindowMarginWidth XmCMainWindowMarginWidth	0 Dimension	CSG
XmNmenuBar XmCMenuBar	NULL Widget	CSG
XmNmessageWindow XmCMessageWindow	NULL Widget	CSG
XmNshowSeparator XmCShowSeparator	False Boolean	CSG

XmNcommandWindow

Specifies the widget to be laid out as the CommandWindow. This widget must have been previously created and managed as a child of MainWindow.

XmNcommandWindowLocation

Controls the position of the command window. **XmCOMMAND_ABOVE_WORKSPACE** locates the command window between the menu bar and the work window. **XmCOMMAND_BELOW_WORKSPACE** locates the command window between the work window and the message window.

XmNmainWindowMarginHeight
Specifies the margin height on the top and bottom of MainWindow. This resource overrides any setting of the ScrolledWindow resource **XmNscrolledWindowMarginHeight**.

XmNmainWindowMarginWidth
Specifies the margin width on the right and left sides of MainWindow. This resource overrides any setting of the ScrolledWindow resource **XmNscrolledWindowMarginWidth**.

XmNmenuBar
Specifies the widget to be laid out as the MenuBar. This widget must have been previously created and managed as a child of MainWindow.

XmNmessageWindow
Specifies the widget to be laid out as the MessageWindow. This widget must have been previously created and managed as a child of MainWindow. The MessageWindow is positioned at the bottom of the MainWindow. If this value is NULL, no message window is included in the MainWindow.

XmNshowSeparator
Displays separators between the components of the MainWindow when set to True. If set to False, no separators are displayed.

Inherited Resources

MainWindow inherits behavior and resources from the superclasses described in the following table. For a complete description of each resource, refer to the reference page for that superclass.

XmScrolledWindow Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNclipWindow XmCClipWindow	dynamic Widget	G
XmNhorizontalScrollBar XmCHorizontalScrollBar	dynamic Widget	CSG
XmNscrollBarDisplayPolicy XmCScrollBarDisplayPolicy	dynamic unsigned char	CSG
XmNscrollBarPlacement XmCScrollBarPlacement	XmBOTTOM_RIGHT unsigned char	CSG
XmNscrolledWindowMarginHeight XmCScrolledWindowMarginHeight	0 Dimension	N/A
XmNscrolledWindowMarginWidth XmCScrolledWindowMarginWidth	0 Dimension	N/A
XmNscrollingPolicy XmCScrollingPolicy	XmAPPLICATION_DEFINED unsigned char	CG
XmNspacing XmCSpacing	4 Dimension	CSG
XmNtraverseObscuredCallback XmCCallback	NULL XtCallbackList	CSG
XmNverticalScrollBar XmCVerticalScrollBar	dynamic Widget	CSG
XmNvisualPolicy XmCVisualPolicy	dynamic unsigned char	G
XmNworkWindow XmCWorkWindow	NULL Widget	CSG

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	0 Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

MainWindow inherits translations from ScrolledWindow.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreateMainWindow(3X)**, **XmMainWindowSep1(3X)**, **XmMainWindowSep2(3X)**, **XmMainWindowSep3(3X)**, **XmMainWindowSetAreas(3X)**, **XmManager(3X)**, and **XmScrolledWindow(3X)**

XmMainWindowSep1—A MainWindow function that returns the widget ID of the first Separator widget

Synopsis

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

Widget XmMainWindowSep1 (widget)
        Widget          widget;
```

Description

XmMainWindowSep1 returns the widget ID of the first Separator widget in the MainWindow. The first Separator widget is located between the MenuBar and the Command widget. This Separator is visible only when **XmNshowSeparator** is True.

widget Specifies the MainWindow widget ID.

For a complete definition of MainWindow and its associated resources, see **XmMainWindow(3X)**.

Return Value

Returns the widget ID of the first Separator.

Related Information

XmMainWindow(3X).

XmMainWindowSep2—A MainWindow function that returns the widget ID of the second Separator widget

Synopsis

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

Widget XmMainWindowSep2 (widget)
        Widget          widget;
```

Description

XmMainWindowSep2 returns the widget ID of the second Separator widget in the MainWindow. The second Separator widget is located between the Command widget and the ScrolledWindow. This Separator is visible only when **XmNshowSeparator** is True.

widget Specifies the MainWindow widget ID.

For a complete definition of MainWindow and its associated resources, see **XmMainWindow(3X)**.

Return Value

Returns the widget ID of the second Separator.

Related Information

XmMainWindow(3X).

XmMainWindowSep3—A MainWindow function that returns the widget ID of the third Separator widget

Synopsis

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

Widget XmMainWindowSep3 (widget)
        Widget          widget;
```

Description

XmMainWindowSep3 returns the widget ID of the third Separator widget in the MainWindow. The third Separator widget is located between the message window and the widget above it. This Separator is visible only when **XmNshowSeparator** is True.

widget Specifies the MainWindow widget ID

For a complete definition of MainWindow and its associated resources, see **XmMainWindow(3X)**.

Return Value

Returns the widget ID of the third Separator.

Related Information

XmMainWindow(3X).

XmMainWindowSetAreas—A MainWindow function that identifies manageable children for each area

Synopsis

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

void XmMainWindowSetAreas (widget, menu_bar, command_window,
                           horizontal_scrollbar, vertical_scrollbar, work_region)
        Widget      widget;
        Widget      menu_bar;
        Widget      command_window;
        Widget      horizontal_scrollbar;
        Widget      vertical_scrollbar;
        Widget      work_region;
```

Description

XmMainWindowSetAreas identifies which of the valid children for each area (such as the MenuBar and work region) are to be actively managed by MainWindow. This function also sets up or adds the MenuBar, work window, command window, and ScrollBar widgets to the application's main window widget.

Each area is optional; therefore, the user can pass NULL to one or more of the following arguments. The window manager provides the title bar.

widget Specifies the MainWindow widget ID.

menu_bar Specifies the widget ID for the MenuBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNmenuBar**.

command_window
Specifies the widget ID for the command window to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNcommandWindow**.

horizontal_scrollbar
Specifies the ScrollBar widget ID for the horizontal ScrollBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNhorizontalScrollBar**.

vertical_scrollbar
: Specifies the ScrollBar widget ID for the vertical ScrollBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNverticalScrollBar**.

work_region
: Specifies the widget ID for the work window to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNworkWindow**.

For a complete definition of MainWindow and its associated resources, see **XmMainWindow(3X)**.

Related Information

XmMainWindow(3X).

XmManager—The Manager widget class

Synopsis

#include <Xm/Xm.h>

Description

Manager is a widget class used as a supporting superclass for other widget classes. It supports the visual resources, graphics contexts, and traversal resources necessary for the graphics and traversal mechanisms.

Classes

Manager inherits behavior and resources from **Core**, **Composite**, and **Constraint**.

The class pointer is **xmManagerWidgetClass**.

The class name is **XmManager**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	0 Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

XmNbottomShadowColor
Specifies the color to use to draw the bottom and right sides of the border shadow. This color is used if the **XmNbottomShadowPixmap** resource is NULL.

XmNbottomShadowPixmap
Specifies the pixmap to use to draw the bottom and right sides of the border shadow.

XmNforeground
Specifies the foreground drawing color used by manager widgets.

XmNhelpCallback
Specifies the list of callbacks that are called when the help key sequence is pressed. The reason sent by this callback is **XmCR_HELP**.

XmNhighlightColor
Specifies the color of the highlighting rectangle. This color is used if the highlight pixmap resource is **XmUNSPECIFIED_PIXMAP**.

XmNhighlightPixmap
Specifies the pixmap used to draw the highlighting rectangle.

XmNinitialFocus
Specifies the ID of a widget descendant of the manager. The widget must meet these conditions:

- The widget must be either a tab group or a non-tab-group widget that can receive keyboard focus. For the definition of a tab group, see the description of the Manager, Primitive, and Gadget **XmNnavigationType** resources. In general a widget can receive keyboard focus when it is a primitive, a gadget, or a manager (such as a DrawingArea with no traversable children) that acts as a primitive.
- The widget must not be a descendant of a tab group that is itself a descendant of the manager. That is, the widget cannot be contained within a tab group that is nested inside the manager.
- The widget and its ancestors must have a value of True for their **XmNtraversalOn** resources.

If the widget does not meet these conditions, **XmNinitialFocus** is treated as if the value were NULL.

This resource is meaningful only when the nearest shell ancestor's **XmNkeyboardFocusPolicy** is **XmEXPLICIT**. It is used to determine which widget receives focus in these situations:

- When the manager is the child of a shell and the shell hierarchy receives focus for the first time
- When focus is inside the shell hierarchy, the manager is a composite tab group, and the user traverses to the manager via the keyboard

Focus is then determined as follows:

- If **XmNinitialFocus** is a traversable non-tab-group widget, that widget receives focus.
- If **XmNinitialFocus** is a traversable tab group, that tab group receives focus. If that tab group is a composite with descendant tab groups or traversable non-tab-group widgets, these procedures are used recursively to assign focus to a descendant of that tab group.
- If **XmNinitialFocus** is NULL, the first traversable non-tab-group widget that is not contained within a nested tab group receives focus.
- If **XmNinitialFocus** is NULL and no traversable non-tab-group widget exists, the first traversable tab group that is not contained within a nested tab group receives focus. If that tab group is a composite with descendant tab groups or traversable non-tab-group widgets, these procedures are used recursively to assign focus to a descendant of that tab group.

If a shell hierarchy regains focus after losing it, focus returns to the widget that had the focus at the time it left the hierarchy.

The use of **XmNinitialFocus** is undefined if the manager is a MenuBar, PulldownMenu, PopupMenu, or OptionMenu.

XmNnavigationType
Determines whether the widget is a tab group.

XmNONE Indicates that the widget is not a tab group.

XmTAB_GROUP
Indicates that the widget is a tab group, unless the **XmNnavigationType** of another widget in the hierarchy is **XmEXCLUSIVE_TAB_GROUP**.

XmSTICKY_TAB_GROUP
Indicates that the widget is a tab group, even if the **XmNnavigationType** of another widget in the hierarchy is **XmEXCLUSIVE_TAB_GROUP**.

XmEXCLUSIVE_TAB_GROUP
Indicates that the widget is a tab group and that widgets in the hierarchy whose **XmNnavigationType** is **XmTAB_GROUP** are not tab groups.

When a parent widget has an **XmNnavigationType** of **XmEXCLUSIVE_TAB_GROUP**, traversal of non-tab-group widgets within the group is based on the order of those widgets in their parent's **XmNchildren** list.

When the **XmNnavigationType** of any widget in a hierarchy is **XmEXCLUSIVE_TAB_GROUP**, traversal of tab groups in the hierarchy proceeds to widgets in the order in which their **XmNnavigationType** resources were specified as **XmEXCLUSIVE_TAB_GROUP** or **XmSTICKY_TAB_GROUP**, whether by creating the widgets with that value, by calling **XtSetValues**, or by calling **XmAddTabGroup**.

XmNshadowThickness
Specifies the thickness of the drawn border shadow. **XmBulletinBoard** and its descendants set this value dynamically. If the widget is a top level window, this value is set to 1. If it is not a top level window, this value is set to 0 (zero).

XmNstringDirection

Specifies the initial direction to draw strings. The values for this resource are **XmSTRING_DIRECTION_L_TO_R** and **XmSTRING_DIRECTION_R_TO_L**. The value of this resource is determined at creation time. If the widget's parent is a manager, this value is inherited from the widget's parent, otherwise it is set to **XmSTRING_DIRECTION_L_TO_R**.

XmNtopShadowColor

Specifies the color to use to draw the top and left sides of the border shadow. This color is used if the **XmNtopShadowPixmap** resource is NULL.

XmNtopShadowPixmap

Specifies the pixmap to use to draw the top and left sides of the border shadow.

XmNtraversalOn

Specifies whether traversal is activated for this widget.

XmNunitType

Provides the basic support for resolution independence. It defines the type of units a widget uses with sizing and positioning resources. If the widget's parent is a subclass of **XmManager** and if the **XmNunitType** resource is not explicitly set, it defaults to the unit type of the parent widget. If the widget's parent is not a subclass of **XmManager**, the resource has a default unit type of **XmPIXELS**.

XmNunitType can have the following values:

XmPIXELS All values provided to the widget are treated as normal pixel values.

Xm100TH_MILLIMETERS

All values provided to the widget are treated as 1/100 of a millimeter.

Xm1000TH_INCHES

All values provided to the widget are treated as 1/1000 of an inch.

Xm100TH_POINTS
All values provided to the widget are treated as 1/100 of a point. A point is a unit used in text processing applications and is defined as 1/72 of an inch.

Xm100TH_FONT_UNITS
All values provided to the widget are treated as 1/100 of a font unit. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

XmNuserData
Allows the application to attach any necessary specific data to the widget. This is an internally unused resource.

Dynamic Color Defaults

The foreground, background, top shadow, bottom shadow, and highlight color resources are dynamically defaulted. If no color data is specified, the colors are automatically generated. On a single-plane system, a black and white color scheme is generated. Otherwise, four colors are generated, which display the correct shading for the 3-D visuals. If the background is the only color specified for a widget, the top shadow and bottom shadow colors are generated to give the 3-D appearance. Foreground and highlight colors are generated to provide sufficient contrast with the background color.

Colors are generated only at creation. Resetting the background through **XtSetValues** does not regenerate the other colors. **XmChangeColor** can be used to recalculate all associated colors based on a new background color.

Inherited Resources

Manager inherits the following resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked. For this callback, *reason* is set to **XmCR_HELP**.

event Points to the **XEvent** that triggered the callback.

Translations

The following set of translations are used by Manager widgets that have Gadget children. Because Gadgets cannot have translations associated with them, it is the responsibility of the Manager widget to intercept the events of interest and pass them to any Gadget child with focus. These events are ignored if no Gadget child has the focus. These translations may not directly correspond to a translation table.

BAny Motion: **ManagerGadgetButtonMotion()**

BSelect Press: **ManagerGadgetArm()**

BSelect Click: **ManagerGadgetActivate()**

BSelect Release: **ManagerGadgetActivate()**

BSelect Press 2+: **ManagerGadgetMultiArm()**

BSelect Release 2+: **ManagerGadgetMultiActivate()**

BTransfer Press:	**ManagerGadgetDrag()**
KSelect:	**ManagerGadgetSelect()**
KActivate:	**ManagerParentActivate()**
KCancel:	**ManagerParentCancel()**
KPrevField:	**ManagerGadgetPrevTabGroup()**
KNextField:	**ManagerGadgetNextTabGroup()**
KUp:	**ManagerGadgetTraverseUp()**
KDown:	**ManagerGadgetTraverseDown()**
KLeft:	**ManagerGadgetTraverseLeft()**
KRight:	**ManagerGadgetTraverseRight()**
KBeginLine:	**ManagerGadgetTraverseHome()**
KHelp:	**ManagerGadgetHelp()**
KAny:	**ManagerGadgetKeyInput()**

Action Routines

The **XmManager** action routines are

ManagerGadgetActivate():
Causes the current gadget to be activated.

ManagerGadgetArm():
Causes the current gadget to be armed.

Causes the current gadget to process a mouse motion event.

ManagerGadgetDrag():
Drags the contents of a gadget label, identified by pressing **BTransfer**. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT** for a label type of **XmSTRING**; otherwise, **PIXMAP** if the label type is **XmPIXMAP**. This action is undefined for gadgets used in a menu system.

ManagerGadgetHelp():
Calls the callbacks for the current gadget's **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

ManagerGadgetKeyInput():
Causes the current gadget to process a keyboard event.

ManagerGadgetMultiActivate():
Causes the current gadget to process a multiple mouse click.

ManagerGadgetMultiArm():
Causes the current gadget to process a multiple mouse button press.

ManagerGadgetNextTabGroup():
Traverses to the first item in the next tab group. If the current tab group is the last entry in the tab group list, it wraps to the beginning of the tab group list.

ManagerGadgetPrevTabGroup():
Traverses to the first item in the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

ManagerGadgetSelect():
Causes the current gadget to be armed and activated.

ManagerGadgetTraverseDown():
Traverses to the next item below the current gadget in the current tab group, wrapping if necessary.

ManagerGadgetTraverseHome():
Traverses to the first widget or gadget in the current tab group.

ManagerGadgetTraverseLeft():
Traverses to the next item to the left of the current gadget in the current tab group, wrapping if necessary.

ManagerGadgetTraverseNext():
Traverses to the next item in the current tab group, wrapping if necessary.

ManagerGadgetTraversePrev():
Traverses to the previous item in the current tab group, wrapping if necessary.

ManagerGadgetTraverseRight()
Traverses to the next item to the right of the current gadget in the current tab group, wrapping if necessary.

ManagerGadgetTraverseUp():
Traverses to the next item above the current gadget in the current tab group, wrapping if necessary.

ManagerParentActivate():
If the parent is a manager, passes the **KActivate** event received by the current widget/gadget to its parent.

ManagerParentCancel():
If the parent is a manager, passes the **KCancel** event received by the current widget/gadget to its parent.

Additional Behavior

This widget has the following additional behavior:

<FocusIn>: If the shell's keyboard focus policy is **XmEXPLICIT** and the event occurs in a gadget, causes the gadget to be highlighted and to take the focus.

<FocusOut>: If the shell's keyboard focus policy is **XmEXPLICIT** and the event occurs in a gadget, causes the gadget to be unhighlighted and to lose the focus.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmChangeColor(3X)**, **XmGadget(3X)**, and **XmScreen(3X)**.

XmMapSegmentEncoding—A compound string function that returns the compound text encoding format associated with the specified font list tag

Synopsis

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

char * XmMapSegmentEncoding (fontlist_tag)
        char            *fontlist_tag;
```

Description

XmMapSegmentEncoding searches the segment encoding registry for an entry that matches the specified font list tag and returns a copy of the associated compound text encoding format. The application is responsible for freeing the storage associated with the returned data by calling **XtFree**.

fontlist_tag Specifies the compound string font list tag

Return Value

Returns a copy of the associated compound text encoding format if the font list tag is found in the registry; otherwise, returns NULL.

Related Information

XmCvtXmStringToCT(3X), **XmFontList(3X)**, **XmRegisterSegmentEncoding(3X)**, and **XmString(3X)**.

XmMenuPosition—A RowColumn function that positions a Popup MenuPane

Synopsis

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

void XmMenuPosition (menu, event)
        Widget                  menu;
        XButtonPressedEvent     * event;
```

Description

XmMenuPosition positions a Popup MenuPane using the information in the specified event. Unless an application is positioning the MenuPane itself, it must first invoke this function before managing the PopupMenu. The *x_root* and *y_root* values in the specified event are used to determine the menu position.

menu	Specifies the PopupMenu to be positioned
event	Specifies the event passed to the action procedure which manages the PopupMenu

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Related Information

XmRowColumn(3X).

XmMenuShell—The MenuShell widget class

Synopsis

#include <Xm/MenuShell.h>

Description

The MenuShell widget is a custom OverrideShell widget. An OverrideShell widget bypasses **mwm** when displaying itself. It is designed specifically to contain Popup or Pulldown MenuPanes.

Most application writers never encounter this widget if they use the menu-system convenience functions, **XmCreatePopupMenu** or **XmCreatePulldown Menu**, to create a Popup or Pulldown MenuPane. The convenience functions automatically create a MenuShell widget as the parent of the MenuPane. However, if the convenience functions are not used, the application programmer must create the required MenuShell. In this case, it is important to note that the parent of the MenuShell depends on the type of menu system being built.

- If the MenuShell is for the top-level Popup MenuPane, the MenuShell's parent must be the widget from which the Popup MenuPane is popped up.
- If the MenuShell is for a MenuPane that is pulled down from a Popup or another Pulldown MenuPane, the MenuShell's parent must be the Popup or Pulldown MenuPane.
- If the MenuShell is for a MenuPane that is pulled down from a MenuBar, the MenuShell's parent must be the MenuBar.
- If the MenuShell is for a Pulldown MenuPane in an OptionMenu, the MenuShell's parent must be the OptionMenu's parent.

Setting **XmNheight**, **XmNwidth**, or **XmNborderWidth** for either a MenuShell or its child sets that resource to the same value in both the parent and the child. An application should always specify these resources for the child, not the parent.

For the managed child of a MenuShell, regardless of the value of the shell's **XmNallowShellResize**, setting **XmNx** or **XmNy** sets the corresponding resource of the parent but does not change the child's position relative to the parent. **XtGetValues** for the child's **XmNx** or **XmNy** yields the value of the corresponding resource in the parent. The x and y-coordinates of the child's upper left outside corner relative to the parent's upper left inside corner are both 0 (zero) minus the value of **XmNborderWidth**.

Classes

MenuShell inherits behavior and resources from **Core**, **Composite**, **Shell**, and **OverrideShell**.

The class pointer is **xmMenuShellWidgetClass**.

The class name is **XmMenuShell**.

New Resources

MenuShell overrides the **XmNallowShellResize** resource in Shell. The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmMenuShell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNdefaultFontList XmCDefaultFontList	dynamic XmFontList	CG
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG

XmNbuttonFontList

Specifies the font list used for MenuShell's button descendants. If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNbuttonFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, **XmNbuttonFontList** is initialized by looking up the parent hierarchy of the widget for an ancestor that is a subclass of the **XmBulletinBoard**, VendorShell, or

XmMenuShell widget class. If such an ancestor is found, **XmNbuttonFontList** is initialized to the **XmNbuttonFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

XmNdefaultFontList

Specifies a default font list for MenuShell's descendants. This resource is obsolete and exists for compatibility with earlier releases. It has been replaced by **XmNbuttonFontList** and **XmNlabelFontList**.

XmNlabelFontList

Specifies the font list used for MenuShell's label descendants (Labels and LabelGadgets). If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNlabelFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or XmMenuShell widget class. If such an ancestor is found, **XmNlabelFontList** is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

Inherited Resources

MenuShell inherits behavior and resources from the following superclasses. For a complete description of each resource, refer to the man page for that superclass. The following tables define a set of widget resources used by the programmer to specify data. The programmer can set the resource values for these inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

Shell Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowShellResize XmCAllowShellResize	True Boolean	G
XmNcreatePopupChildProc XmCCreatePopupChildProc	NULL XtCreatePopupChildProc	CSG
XmNgeometry XmCGeometry	NULL String	CSG
XmNoverrideRedirect XmCOverrideRedirect	True Boolean	CSG
XmNpopdownCallback XmCCallback	NULL XtCallbackList	C
XmNpopupCallback XmCCallback	NULL XtCallbackList	C
XmNsaveUnder XmCSaveUnder	True Boolean	CSG
XmNvisual XmCVisual	CopyFromParent Visual *	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

The **XmMenuShell** translations are described in the following list. These translations may not directly correspond to a translation table.

BSelect Press: **ClearTraversal()**

BSelect Release: **MenuShellPopdownDone()**

Action Routines

The **XmMenuShell** action routines are

ClearTraversal():
Disables keyboard traversal for the menu, enables mouse traversal, and unposts any menus posted by this menu.

MenuShellPopdownDone():
Unposts the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores focus to the widget that had the focus before the menu system was entered.

MenuShellPopdownOne():
In a top-level Pulldown MenuPane from a MenuBar, this action unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, this action unposts the menu.

In a Popup MenuPane, this action unposts the menu, and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget from which the menu was posted.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Core(3X)**, **OverrideShell(3X)**, **Shell(3X)**, **XmCreateMenuShell(3X)**, **XmCreatePopupMenu(3X)**, **XmCreatePulldown(3X)**, and **XmRowColumn(3X)**.

XmMessageBox—The MessageBox widget class

Synopsis

#include <Xm/MessageB.h>

Description

MessageBox is a dialog class used for creating simple message dialogs. Convenience dialogs based on MessageBox are provided for several common interaction tasks, which include giving information, asking questions, and reporting errors.

A MessageBox dialog is typically transient in nature, displayed for the duration of a single interaction. MessageBox is a subclass of **XmBulletinBoard** and depends on it for much of its general dialog behavior.

The default value for **XmNinitialFocus** is the value of **XmNdefaultButton**.

A typical MessageBox contains a message symbol, a message, and up to three standard default PushButtons: **OK, Cancel**, and **Help**. It is laid out with the symbol and message on top and the PushButtons on the bottom. The help button is positioned to the side of the other push buttons. You can localize the default symbols and button labels for MessageBox convenience dialogs.

The user can specify resources in a resource file for the gadgets created automatically that contain the MessageBox symbol pixmap and separator. The gadget names are **Symbol** and **Separator**.

A MessageBox can also be customized by creating and managing new children that are added to the MessageBox children created automatically by the convenience dialogs. In the case of TemplateDialog, only the separator child is created by default. If the callback, string, or pixmap symbol resources are specified, the appropriate child will be created.

Additional children are laid out in the following manner:

- The first MenuBar child is placed at the top of the window.
- All **XmPushButton** widgets or gadgets, and their subclasses are placed after the **OK** button in the order of their creation.
- A child that is not in the above categories is placed above the row of buttons. If a message label exists, the child is placed below the label. If a message pixmap exists, but a message label is absent, the child is placed on the same row as the pixmap. The child behaves as a work area and grows or shrinks to fill the space above the row of buttons. The layout of multiple work area children is undefined.

At initialization, MessageBox looks for the following bitmap files:

- **xm_error**
- **xm_information**
- **xm_question**
- **xm_working**
- **xm_warning**

See **XmGetPixmap(3X)** for a list of the paths that are searched for these files.

Classes

MessageBox inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XmBulletinBoard**.

The class pointer is **xmMessageBoxWidgetClass**.

The class name is **XmMessageBox**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmMessageBox Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNcancelCallback XmCCallback	NULL XtCallbackList	C
XmNcancelLabelString XmCCancelLabelString	dynamic XmString	CSG
XmNdefaultButtonType XmCDefaultButtonType	XmDIALOG_OK_BUTTON unsigned char	CSG
XmNdialogType XmCDialogType	XmDIALOG_MESSAGE unsigned char	CSG
XmNhelpLabelString XmCHelpLabelString	dynamic XmString	CSG
XmNmessageAlignment XmCAlignment	XmALIGNMENT_BEGINNING unsigned char	CSG
XmNmessageString XmCMessageString	"" XmString	CSG
XmNminimizeButtons XmCMinimizeButtons	False Boolean	CSG
XmNokCallback XmCCallback	NULL XtCallbackList	C
XmNokLabelString XmCOkLabelString	dynamic XmString	CSG
XmNsymbolPixmap XmCPixmap	dynamic Pixmap	CSG

XmNcancelCallback

Specifies the list of callbacks that is called when the user clicks on the cancel button. The reason sent by the callback is **XmCR_CANCEL**.

XmNcancelLabelString

Specifies the string label for the cancel button. The default for this resource depends on the locale. In the C locale the default is **Cancel**.

XmNdefaultButtonType

Specifies the default PushButton. A value of **XmDIALOG_NONE** means that there should be no default PushButton. The following types are valid:

- **XmDIALOG_CANCEL_BUTTON**
- **XmDIALOG_OK_BUTTON**
- **XmDIALOG_HELP_BUTTON**
- **XmDIALOG_NONE**

XmNdialogType

Specifies the type of MessageBox dialog, which determines the default message symbol. The following are the possible values for this resource:

- Indicates an ErrorDialog.
- Indicates an InformationDialog.
- Indicates a MessageDialog. This is the default MessageBox dialog type. It does not have an associated message symbol.
- Indicates a QuestionDialog.
- Indicates a TemplateDialog. The TemplateDialog contains only a separator child. It does not have an associated message symbol.
- indicates a WarningDialog.
- Indicates a WorkingDialog.

If this resource is changed with **XtSetValues**, the symbol bitmap is modified to the new **XmNdialogType** bitmap unless **XmNsymbolPixmap** is also being set in the call to **XtSetValues**. If the dialog type does not have an associated message symbol, then no bitmap will be displayed.

XmNhelpLabelString
Specifies the string label for the help button. The default for this resource depends on the locale. In the C locale the default is **Help**.

XmNmessageAlignment
Controls the alignment of the message Label. Possible values include the following:

- **XmALIGNMENT_BEGINNING** (default)
- **XmALIGNMENT_CENTER**
- **XmALIGNMENT_END**

XmNmessageString
Specifies the string to be used as the message.

XmNminimizeButtons
Sets the buttons to the width of the widest button and height of the tallest button if False. If True, button width and height are set to the preferred size of each button.

XmNokCallback
Specifies the list of callbacks that is called when the user clicks on the OK button. The reason sent by the callback is **XmCR_OK**.

XmNokLabelString
Specifies the string label for the OK button. The default for this resource depends on the locale. In the C locale the default is **OK**.

XmNsymbolPixmap
Specifies the pixmap label to be used as the message symbol.

Inherited Resources

MessageBox inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmBulletinBoard Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowOverlap XmCAllowOverlap	True Boolean	CSG
XmNautoUnmanage XmCAutoUnmanage	True Boolean	CG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNcancelButton XmCWidget	Cancel button Widget	SG
XmNdefaultButton XmCWidget	dynamic Widget	SG
XmNdefaultPosition XmCDefaultPosition	True Boolean	CSG
XmNdialogStyle XmCDialogStyle	dynamic unsigned char	CSG
XmNdialogTitle XmCDialogTitle	NULL XmString	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	10 Dimension	CSG
XmNmarginWidth XmCMarginWidth	10 Dimension	CSG
XmNnoResize XmCNoResize	False Boolean	CSG
XmNresizePolicy XmCResizePolicy	XmRESIZE_ANY unsigned char	CSG

Name Class	Default Type	Access
XmNshadowType XmCShadowType	XmSHADOW_OUT unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNtextTranslations XmCTranslations	NULL XtTranslations	C
XmNunmapCallback XmCCallback	NULL XtCallbackList	C

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	dynamic Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	N/A
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

Translations

XmMessageBox includes the translations from **XmManager**.

Additional Behavior

The **XmMessageBox** widget has the following additional behavior:

MAny KCancel:
Calls the activate callbacks for the cancel button if it is sensitive.

KActivate: Calls the activate callbacks for the button with the keyboard focus. If no button has the keyboard focus, calls the activate callbacks for the default button if it is sensitive.

<Ok Button Activated>:
Calls the callbacks for **XmNokCallback**.

<Cancel Button Activated>:
Calls the callbacks for **XmNcancelCallback**.

<Help Button Activated>:
Calls the callbacks for **XmNhelpCallback**.

<FocusIn>: Calls the callbacks for **XmNfocusCallback**.

<Map>: Calls the callbacks for **XmNmapCallback** if the parent is a DialogShell.

<Unmap>: Calls the callbacks for **XmNunmapCallback** if the parent is a DialogShell.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmBulletinBoard(3X)**, **XmCreateErrorDialog(3X)**, **XmCreateInformationDialog(3X)**, **XmCreateMessageBox(3X)**, **XmCreateMessageDialog(3X)**, **XmCreateQuestionDialog(3X)**, **XmCreateTemplateDialog(3X)**, **XmCreateWarningDialog(3X)**, **XmCreateWorkingDialog(3X)**, **XmManager(3X)**, and **XmMessageBoxGetChild(3X)**.

XmMessageBoxGetChild—A MessageBox function that is used to access a component

Synopsis

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

Widget XmMessageBoxGetChild (widget, child)
        Widget          widget;
        unsigned char   child;
```

Description

XmMessageBoxGetChild is used to access a component within a MessageBox. The parameters given to the function are the MessageBox widget and a value indicating which component to access.

widget Specifies the MessageBox widget ID.

child Specifies a component within the MessageBox. The following are legal values for this parameter:

- **XmDIALOG_CANCEL_BUTTON**
- **XmDIALOG_DEFAULT_BUTTON**
- **XmDIALOG_HELP_BUTTON**
- **XmDIALOG_MESSAGE_LABEL**
- **XmDIALOG_OK_BUTTON**
- **XmDIALOG_SEPARATOR**
- **XmDIALOG_SYMBOL_LABEL**

For a complete definition of MessageBox and its associated resources, see **XmMessageBox(3X)**.

Return Value

Returns the widget ID of the specified MessageBox component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmMessageBox(3X).

XmOptionButtonGadget—A RowColumn function that obtains the widget ID for the CascadeButtonGadget in an OptionMenu

Synopsis

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

Widget XmOptionButtonGadget (option_menu)
        Widget          option_menu;
```

Description

XmOptionButtonGadget provides the application with the means for obtaining the widget ID for the internally created CascadeButtonGadget. Once the application has obtained the widget ID, it can adjust the visuals for the CascadeButtonGadget, if desired.

option_menu Specifies the OptionMenu widget ID

When an application creates an instance of the OptionMenu widget, the widget creates two internal gadgets. One is a LabelGadget that is used to display RowColumn's **XmNlabelString** resource. The other is a CascadeButtonGadget that displays the current selection and provides the means for posting the OptionMenu's submenu.

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. The following list identifies the names of these widgets (or gadgets) and the associated OptionMenu areas.

Option Menu Label Gadget	**OptionLabel**
Option Menu Cascade Button	**OptionButton**

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the widget ID for the internal button.

Related Information

XmCreateOptionMenu(3X), **XmCascadeButtonGadget(3X)**, **XmOptionLabelGadget(3X)**, and **XmRowColumn(3X)**.

XmOptionLabelGadget—A RowColumn function that obtains the widget ID for the LabelGadget in an OptionMenu

Synopsis

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

Widget XmOptionLabelGadget (option_menu)
        Widget          option_menu;
```

Description

XmOptionLabelGadget provides the application with the means for obtaining the widget ID for the internally created LabelGadget. Once the application has obtained the widget ID, it can adjust the visuals for the LabelGadget, if desired.

option_menu Specifies the OptionMenu widget ID

When an application creates an instance of the OptionMenu widget, the widget creates two internal gadgets. One is a LabelGadget that is used to display RowColumn's **XmNlabelString** resource. The other is a CascadeButtonGadget that displays the current selection and provides the means for posting the OptionMenu's submenu.

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. The following list identifies the names of these widgets (or gadgets) and the associated OptionMenu areas.

Option Menu Label Gadget	**OptionLabel**
Option Menu Cascade Button	**OptionButton**

For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the widget ID for the internal label.

Related Information

XmCreateOptionMenu(3X), **XmLabelGadget(3X)**, **XmOptionButtonGadget(3X)**, and **XmRowColumn(3X)**.

XmPanedWindow—The PanedWindow widget class

Synopsis

#include <Xm/PanedW.h>

Description

PanedWindow is a composite widget that lays out children in a vertically tiled format. Children appear in top-to-bottom fashion, with the first child inserted appearing at the top of the PanedWindow and the last child inserted appearing at the bottom. The PanedWindow grows to match the width of its widest child and all other children are forced to this width. The height of the PanedWindow is equal to the sum of the heights of all its children, the spacing between them, and the size of the top and bottom margins.

The user can also adjust the size of the panes. To facilitate this adjustment, a pane control sash is created for most children. The sash appears as a square box positioned on the bottom of the pane that it controls. The user can adjust the size of a pane by using the mouse or keyboard.

The PanedWindow is also a constraint widget, which means that it creates and manages a set of constraints for each child. You can specify a minimum and maximum size for each pane. The PanedWindow does not allow a pane to be resized below its minimum size or beyond its maximum size. Also, when the minimum size of a pane is equal to its maximum size, no control sash is presented for that pane or for the lowest pane.

The default **XmNinsertPosition** procedure for PanedWindow causes sashes to be inserted at the end of the list of children and causes nonsash widgets to be inserted after other nonsash children but before any sashes.

All panes and sashes in a PanedWindow must be tab groups. When a pane is inserted as a child of the PanedWindow, if the pane's **XmNnavigationType** is not **XmEXCLUSIVE_TAB_GROUP**, PanedWindow sets it to **XmSTICKY_TAB_GROUP**.

Classes

PanedWindow inherits behavior and resources from the **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is **xmPanedWindowWidgetClass**.

The class name is **XmPanedWindow**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmPanedWindow Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNmarginHeight XmCMarginHeight	3 Dimension	CSG
XmNmarginWidth XmCMarginWidth	3 Dimension	CSG
XmNrefigureMode XmCBoolean	True Boolean	CSG
XmNsashHeight XmCSashHeight	10 Dimension	CSG
XmNsashIndent XmCSashIndent	-10 Position	CSG
XmNsashShadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNsashWidth XmCSashWidth	10 Dimension	CSG
XmNseparatorOn XmCSeparatorOn	True Boolean	CSG
XmNspacing XmCSpacing	8 Dimension	CSG

XmNmarginHeight

Specifies the distance between the top and bottom edges of the PanedWindow and its children.

XmNmarginWidth

Specifies the distance between the left and right edges of the PanedWindow and its children.

XmNrefigureMode
Determines whether the panes' positions are recomputed and repositioned when programmatic changes are being made to the PanedWindow. Setting this resource to True resets the children to their appropriate positions.

XmNsashHeight
Specifies the height of the sash.

XmNsashIndent
Specifies the horizontal placement of the sash along each pane. A positive value causes the sash to be offset from the near (left) side of the PanedWindow, and a negative value causes the sash to be offset from the far (right) side of the PanedWindow. If the offset is greater than the width of the PanedWindow minus the width of the sash, the sash is placed flush against the near side of the PanedWindow.

Whether the placement actually corresponds to the left or right side of the PanedWindow may depend on the value of the **XmNstringDirection** resource.

XmNsashShadowThickness
Specifies the thickness of the shadows of the sashes.

XmNsashWidth
Specifies the width of the sash.

XmNseparatorOn
Determines whether a Separator is created between each of the panes. Setting this resource to True creates a Separator at the midpoint between each of the panes.

XmNspacing Specifies the distance between each child pane.

XmPanedWindow Constraint Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowResize XmCBoolean	False Boolean	CSG
XmNpaneMaximum XmCPaneMaximum	1000 Dimension	CSG
XmNpaneMinimum XmCPaneMinimum	1 Dimension	CSG
XmNpositionIndex XmCPositionIndex	XmLAST_POSITION short	CSG
XmNskipAdjust XmCBoolean	False Boolean	CSG

XmNallowResize

Allows an application to specify whether the PanedWindow should allow a pane to request to be resized. This flag has an effect only after the PanedWindow and its children have been realized. If this flag is set to True, the PanedWindow tries to honor requests to alter the height of the pane. If False, it always denies pane requests to resize.

XmNpaneMaximum

Allows an application to specify the maximum size to which a pane may be resized. This value must be greater than the specified minimum.

XmNpaneMinimum

Allows an application to specify the minimum size to which a pane may be resized. This value must be greater than 0 (zero).

XmNpositionIndex

Specifies the position of the widget in its parent's list of children (the list of pane children, not including sashes). The value is an integer that is no less than 0 (zero) and no greater than the number of children in the list at the time the value is specified. A value of 0 means that the child is placed at the beginning of the list. The value can also be specified as **XmLAST_POSITION** (the default), which means that the child is placed at the end of the list. Any other value is ignored. **XtGetValues** returns the position of the widget in its parent's child list at the time of the call to **XtGetValues**.

When a widget is inserted into its parent's child list, the positions of any existing children that are greater than or equal to the specified widget's **XmNpositionIndex** are increased by 1. The effect of a call to **XtSetValues** for **XmNpositionIndex** is to remove the specified widget from its parent's child list, decrease by one the positions of any existing children that are greater than the specified widget's former position in the list, and then insert the specified widget into its parent's child list as described in the preceding sentence.

XmNskipAdjust

When set to True, this Boolean resource allows an application to specify that the PanedWindow should not automatically resize this pane.

Inherited Resources

PanedWindow inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	default procedure XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Translations

XmPanedWindow inherits translations from **XmManager**.

The translations for sashes within the PanedWindow are described in the following table. These translations may not directly correspond to a translation table.

BSelect Press: **SashAction(Start)**

BSelect Motion: **SashAction(Move)**

BSelect Release: **SashAction(Commit)**

BTransfer Press: **SashAction(Start)**

BTransfer Motion: **SashAction(Move)**

BTransfer Release: **SashAction(Commit)**

KUp: **SashAction(Key,DefaultIncr,Up)**

MCtrl KUp: **SashAction(Key,LargeIncr,Up)**

KDown:	**SashAction(Key,DefaultIncr,Down)**
MCtrl KDown:	**SashAction(Key,LargeIncr,Down)**
KNextField:	**NextTabGroup()**
KPrevField:	**PrevTabGroup()**
KHelp:	**Help()**

Action Routines

The **XmPanedWindow** action routines are

Help(): Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

NextTabGroup():
Moves the keyboard focus to the next tab group. By default, each pane and sash is a tab group.

PrevTabGroup():
Moves the keyboard focus to the previous tab group. By default, each pane and sash is a tab group.

SashAction(*action***)** or **SashAction(Key,***increment***,***direction***)**:
The **Start** action activates the interactive placement of the pane's borders. The **Move** action causes the sash to track the position of the pointer. If one of the panes reaches its minimum or maximum size, adjustment continues with the next adjustable pane. The **Commit** action ends sash motion.

When sash action is caused by a keyboard event, the sash with the keyboard focus is moved according to the *increment* and *direction* specified. **DefaultIncr** adjusts the sash by one line. **LargeIncr** adjusts the sash by one view region. The *direction* is specified as either **Up** or **Down**.

Note that the SashAction action routine is not a direct action routine of the **XmPanedWindow,** but rather an action of the Sash control created by the **XmPanedWindow**.

Additional Behavior

This widget has the following additional behavior:

<FocusIn>: Moves the keyboard focus to the sash and highlights it

<FocusOut>: Unsets the keyboard focus in the sash and unhighlights it

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreatePanedWindow(3X)**, and **XmManager(3X)**.

XmPrimitive—The Primitive widget class

Synopsis

#include <Xm/Xm.h>

Description

Primitive is a widget class used as a supporting superclass for other widget classes. It handles border drawing and highlighting, traversal activation and deactivation, and various callback lists needed by Primitive widgets.

Classes

Primitive inherits behavior and resources from **Core**.

The class pointer is **xmPrimitiveWidgetClass**.

The class name is **XmPrimitive**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

XmNbottomShadowColor

Specifies the color to use to draw the bottom and right sides of the border shadow. This color is used if the **XmNtopShadowPixmap** resource is unspecified.

XmNbottomShadowPixmap
Specifies the pixmap to use to draw the bottom and right sides of the border shadow.

XmNforeground
Specifies the foreground drawing color used by Primitive widgets.

XmNhelpCallback
Specifies the list of callbacks that is called when the help key is pressed. The reason sent by the callback is **XmCR_HELP**.

XmNhighlightColor
Specifies the color of the highlighting rectangle. This color is used if the highlight pixmap resource is **XmUNSPECIFIED_PIXMAP**.

XmNhighlightOnEnter
Specifies if the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is **XmEXPLICIT**, this resource is ignored, and the widget is highlighted when it has the focus. If the shell's focus policy is **XmPOINTER** and if this resource is True, the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is **XmPOINTER** and if this resource is False, the highlighting rectangle is not drawn when the the cursor moves into the widget. The default is False.

XmNhighlightPixmap
Specifies the pixmap used to draw the highlighting rectangle.

XmNhighlightThickness
Specifies the thickness of the highlighting rectangle.

XmNnavigationType
Determines whether the widget is a tab group.

XmNONE Indicates that the widget is not a tab group.

XmTAB_GROUP
Indicates that the widget is a tab group, unless the **XmNnavigationType** of another widget in the hierarchy is **XmEXCLUSIVE_TAB_GROUP**.

XmSTICKY_TAB_GROUP
Indicates that the widget is a tab group, even if the **XmNnavigationType** of another widget in the hierarchy is **XmEXCLUSIVE_TAB_GROUP**.

XmEXCLUSIVE_TAB_GROUP

Indicates that the widget is a tab group and that widgets in the hierarchy whose **XmNnavigationType** is **XmTAB_GROUP** are not tab groups.

When a parent widget has an **XmNnavigationType** of **XmEXCLUSIVE_TAB_GROUP**, traversal of non-tab-group widgets within the group is based on the order of those widgets in their parent's **XmNchildren** list.

When the **XmNnavigationType** of any widget in a hierarchy is **XmEXCLUSIVE_TAB_GROUP**, traversal of tab groups in the hierarchy proceeds to widgets in the order in which their **XmNnavigationType** resources were specified as either **XmEXCLUSIVE_TAB_GROUP** or **XmSTICKY_TAB_GROUP**, whether by creating the widgets with that value, by calling **XtSetValues**, or by calling **XmAddTabGroup**.

XmNshadowThickness

Specifies the size of the drawn border shadow.

XmNtopShadowColor

Specifies the color to use to draw the top and left sides of the border shadow. This color is used if the **XmNtopShadowPixmap** resource is unspecified.

XmNtopShadowPixmap

Specifies the pixmap to use to draw the top and left sides of the border shadow.

XmNtraversalOn

Specifies if traversal is activated for this widget. In CascadeButton and CascadeButtonGadget, this resource is forced to True unless the parent is an OptionMenu.

XmNunitType

Provides the basic support for resolution independence. It defines the type of units a widget uses with sizing and positioning resources. If the widget's parent is a subclass of **XmManager** and if the **XmNunitType** resource is not explicitly set, it defaults to the unit type of the parent widget. If the widget's parent is not a subclass of **XmManager**, the resource has a default unit type of **XmPIXELS**.

XmNunitType can have the following values:

XmPIXELS All values provided to the widget are treated as normal pixel values.

Xm100TH_MILLIMETERS
All values provided to the widget are treated as 1/100 of a millimeter.

Xm1000TH_INCHES
All values provided to the widget are treated as 1/1000 of an inch.

Xm100TH_POINTS
All values provided to the widget are treated as 1/100 of a point. A point is a unit used in text processing applications and is defined as 1/72 of an inch.

Xm100TH_FONT_UNITS
All values provided to the widget are treated as 1/100 of a font unit. A font unit has horizontal and vertical components. These are the values of the **XmScreen** resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

XmNuserData

Allows the application to attach any necessary specific data to the widget. It is an internally unused resource.

Dynamic Color Defaults

The foreground, background, top shadow, bottom shadow, and highlight color resources are dynamically defaulted. If no color data is specified, the colors are automatically generated. On a single-plane system, a black and white color scheme is generated. Otherwise, four colors are generated, which display the correct shading for the 3-D visuals. If the background is the only color specified for a widget, the top shadow and bottom shadow colors are generated to give the 3-D appearance. Foreground and highlight colors are generated to provide sufficient contrast with the background color.

Colors are generated only at creation. Resetting the background through **XtSetValues** does not regenerate the other colors. **XmChangeColor** can be used to recalculate all associated colors based on a new background color.

Inherited Resources

Primitive inherits behavior and resources from the superclass described in the following table. For a complete description of each resource, refer to the reference page for that superclass.

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          *event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked. For this callback, *reason* is set to **XmCR_HELP**.

event Points to the **XEvent** that triggered the callback.

Translations

The **XmPrimitive** translations are listed below. These translations may not directly correspond to a translation table.

Note that for buttons in menus, altering translations in **#override** or **#augment** mode is undefined.

KUp: **PrimitiveTraverseUp()**

KDown: **PrimitiveTraverseDown()**

KLeft: **PrimitiveTraverseLeft()**

KRight: **PrimitiveTraverseRight()**

KBeginLine: **PrimitiveTraverseHome()**

KNextField: **PrimitiveNextTabGroup()**

KPrevField: **PrimitivePrevTabGroup()**

KActivate:	**PrimitiveParentActivate()**
KCancel:	**PrimitiveParentCancel()**
KHelp:	**PrimitiveHelp()**

Action Routines

The **XmPrimitive** action routines are

PrimitiveHelp():
Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

PrimitiveNextTabGroup():
Traverses to the first item in the next tab group. If the current tab group is the last entry in the tab group list, it wraps to the beginning of the tab group list.

PrimitiveParentActivate():
If the parent is a manager, passes the **KActivate** event received by the widget to the parent.

PrimitiveParentCancel():
If the parent is a manager, passes the **KCancel** event received by the widget to the parent.

PrimitivePrevTabGroup():
Traverses to the first item in the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

PrimitiveTraverseDown():
Traverses to the next item below the current widget in the current tab group, wrapping if necessary.

PrimitiveTraverseHome():
Traverses to the first widget or gadget in the current tab group.

PrimitiveTraverseLeft():
Traverses to the next item to the left of the current widget in the current tab group, wrapping if necessary.

PrimitiveTraverseNext():
Traverses to the next item in the current tab group, wrapping if necessary.

PrimitiveTraversePrev():
Traverses to the previous item in the current tab group, wrapping if necessary.

PrimitiveTraverseRight():
Traverses to the next item to the right of the current gadget in the current tab group, wrapping if necessary.

PrimitiveTraverseUp():
Traverses to the next item above the current gadget in the current tab group, wrapping if necessary.

Additional Behavior

This widget has the following additional behavior:

<FocusIn>: If the shell's keyboard focus policy is **XmEXPLICIT**, highlights the widget and gives it the focus

<FocusOut>: If the shell's keyboard focus policy is **XmEXPLICIT**, unhighlights the widget and removes the focus

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmChangeColor(3X)**, and **XmScreen(3X)**.

XmProcessTraversal—A function that determines which component receives keyboard events when a widget has the focus

Synopsis

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

Boolean XmProcessTraversal (widget, direction)
        Widget              widget;
        XmTraversalDirection direction;
```

Description

XmProcessTraversal determines which component of a hierarchy receives keyboard events when the hierarchy that contains the given widget has keyboard focus. Using **XmProcessTraversal** to traverse to MenuBars, Pulldown MenuPanes, or Popup MenuPanes is not supported.

widget Specifies the widget ID of the widget whose hierarchy is to be traversed. The hierarchy is only traversed up to the top of the shell. If that shell does not currently have the focus, any changes to the element with focus within that shell will not occur until the next time the shell recieves focus.

direction Specifies the direction of traversal.

The *direction* parameter can have the following values, which cause the routine to take the corresponding actions:

XmTRAVERSE_CURRENT
Finds the hierarchy and the tab group that contain *widget*. If this tab group is not the active tab group, this action makes it the active tab group. If *widget* is an item in the active tab group, this action makes it the active item. If *widget* is the active tab group, this action makes the first traversable item in the tab group the active item.

XmTRAVERSE_DOWN
Finds the hierarchy that contains *widget*, finds the active item in the active tab group, and makes the item below it the active item. If there is no item below, it wraps.

XmTRAVERSE_HOME
Finds the hierarchy that contains *widget* and finds the active item in the active tab group and makes the first traversable item in the tab group the active item.

XmTRAVERSE_LEFT
Finds the hierarchy that contains *widget*, finds the active item in the active tab group, and makes the item to the left the active item. If there is no item to the left, this action wraps.

XmTRAVERSE_NEXT
Finds the hierarchy that contains *widget*, finds the active item in the active tab group, and makes the next item in child order the active item.

XmTRAVERSE_NEXT_TAB_GROUP
Finds the hierarchy that contains *widget*, finds the active tab group (if any), and makes the next tab group the active tab group in the hierarchy.

XmTRAVERSE_PREV
Finds the hierarchy that contains *widget*, finds the active item in the active tab group, and makes the previous item in child order the active item.

XmTRAVERSE_PREV_TAB_GROUP
Finds the hierarchy that contains *widget*, finds the active tab group (if any), and makes the previous tab group the active tab group in the hierarchy.

XmTRAVERSE_RIGHT
Finds the hierarchy that contains *widget*, finds the active item in the active tab group, and makes the item to the right the active item. If there is no item to the right, this action wraps.

XmTRAVERSE_UP
Finds the hierarchy that contains *widget*, finds the active item in the active tab group, and makes the item above it the active item. If there is no item above, this action wraps.

Cautions

- **XmProcessTraversal** will not allow traversal to a widget in a different shell.
- **XmProcessTraversal** will only allow traversal to widgets that are currently mapped.
- You cannot call **XmProcessTraversal** from inside a focusCallback routine (or you will get a segmentation fault).

Return Value

Returns True if the setting succeeded. Returns False if the keyboard focus policy is not **XmEXPLICIT**, if there are no traversable items, or if the call to the routine has invalid parameters.

Related Information

XmGetVisibility(3X) and **XmIsTraversable(3X)**.

XmPushButton—The PushButton widget class

Synopsis

#include <Xm/PushB.h>

Description

PushButton issues commands within an application. It consists of a text label or pixmap surrounded by a border shadow. When a PushButton is selected, the shadow changes to give the appearance that it has been pressed in. When a PushButton is unselected, the shadow changes to give the appearance that it is out.

The default behavior associated with a PushButton in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the PushButton. In addition, **BMenu** controls the behavior of the PushButton if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Thickness for a second shadow, used when the PushButton is the default button, may be specified with the **XmNshowAsDefault** resource. If it has a nonzero value, the Label's resources **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** may be modified to accommodate the second shadow.

If an initial value is specified for **XmNarmPixmap** but not for **XmNlabelPixmap**, the **XmNarmPixmap** value is used for **XmNlabelPixmap**.

Classes

PushButton inherits behavior and resources from **Core**, **XmPrimitive**, and **XmLabel**.

The class pointer is **xmPushButtonWidgetClass**.

The class name is **XmPushButton**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmPushButton Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNarmCallback XmCCallback	NULL XtCallbackList	C
XmNarmColor XmCArmColor	dynamic Pixel	CSG
XmNarmPixmap XmCArmPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNdefaultButtonShadowThickness XmCDefaultButtonShadowThickness	dynamic Dimension	CSG
XmNdisarmCallback XmCCallback	NULL XtCallbackList	C
XmNfillOnArm XmCFillOnArm	True Boolean	CSG
XmNmultiClick XmCMultiClick	dynamic unsigned char	CSG
XmNshowAsDefault XmCShowAsDefault	0 Dimension	CSG

XmNactivateCallback

Specifies the list of callbacks that is called when PushButton is activated. PushButton is activated when the user presses and releases the active mouse button while the pointer is inside that widget. Activating the PushButton also disarms it. For this callback, the reason is **XmCR_ACTIVATE**.

XmNarmCallback

Specifies the list of callbacks that is called when PushButton is armed. PushButton is armed when the user presses the active mouse button while the pointer is inside that widget. For this callback, the reason is **XmCR_ARM**.

XmNarmColor

Specifies the color with which to fill the armed button. **XmNfillOnArm** must be set to True for this resource to have an effect. The default for a color display is a color between the background and the bottom shadow color. For a monochrome display, the default is set to the foreground color, and any text in the label appears in the background color when the button is armed.

XmNarmPixmap

Specifies the pixmap to be used as the button face if **XmNlabelType** is **XmPIXMAP** and PushButton is armed. This resource is disabled when the PushButton is in a menu.

XmNdefaultButtonShadowThickness

This resource specifies the width of the default button indicator shadow. If this resource is 0 (zero), the width of the shadow comes from the value of the **XmNshowAsDefault** resource. If this resource is greater than 0, the **XmNshowAsDefault** resource is only used to specify whether this button is the default. The default value is the initial value of **XmNshowAsDefault**.

XmNdisarmCallback

Specifies the list of callbacks that is called when PushButton is disarmed. PushButton is disarmed when the user presses and releases the active mouse button while the pointer is inside that widget. For this callback, the reason is **XmCR_DISARM**.

XmNfillOnArm

Forces the PushButton to fill the background of the button with the color specified by **XmNarmColor** when the button is armed and when this resource is set to True. If False, only the top and bottom shadow colors are switched. When the PushButton is in a menu, this resource is ignored and assumed to be False.

XmNmultiClick

If a button click is followed by another button click within the time span specified by the display's multiclick time, and this resource is set to **XmMULTICLICK_DISCARD**, do not process the second click. If this resource is set to **XmMULTICLICK_KEEP**, process the event and increment *click_count* in the callback structure. When the button is in a menu, the default is **XmMULTICLICK_DISCARD**; otherwise, for a button not in a menu, **XmMULTICLICK_KEEP** is the default value.

XmNshowAsDefault

If **XmNdefaultButtonShadowThickness** is greater than 0 (zero), a value greater than 0 in this resource specifies to mark this button as the default button. If **XmNdefaultButtonShadowThickness** is 0, a value greater than 0 in this resource specifies to mark this button as the default button with the shadow thickness specified by this resource. The space between the shadow and the default shadow is equal to the sum of both shadows. The default value is 0. When this value is not 0, the Label resources **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** may be modified to accommodate the second shadow. This resource is disabled when the PushButton is in a menu.

Inherited Resources

PushButton inherits behavior and resources from the superclasses described the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmLabel Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	CSG
XmNacceleratorText XmCAcceleratorText	NULL XmString	CSG
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	dynamic Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	dynamic Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	dynamic Dimension	CSG
XmNmarginRight XmCMarginRight	dynamic Dimension	CSG
XmNmarginTop XmCMarginTop	dynamic Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG
XmNmnemonic XmCMnemonic	NULL KeySym	CSG

Name Class	Default Type	Access
XmNmnemonicCharSet XmCMnemonicCharSet	XmFONTLIST_DEFAULT_TAG String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    int         click_count;
} XmPushButtonCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

click_count This value is valid only when the reason is **XmCR_ACTIVATE**. It contains the number of clicks in the last multiclick sequence if the **XmNmultiClick** resource is set to **XmMULTICLICK_KEEP**, otherwise it contains 1. The activate callback is invoked for each click if **XmNmultiClick** is set to **XmMULTICLICK_KEEP**.

Translations

XmPushButton includes translations from **XmPrimitive**.

Note that altering translations in **#override** or **#augment** mode is undefined.

Additional **XmPushButton** translations for **XmPushButtons** not in a menu system are described in the following list. These translations may not directly correspond to a translation table.

BTransfer Press: **ProcessDrag()**

BSelect Press: **Arm()**

BSelect Click:	**Activate()**
Disarm()	
BSelect Release:	**Activate()** **Disarm()**
BSelect Press 2+:	**MultiArm()**
BSelect Release 2+:	**MultiActivate()** **Disarm()**
KSelect:	**ArmAndActivate()**
KHelp:	**Help()**

XmPushButton inherits menu traversal translations from **XmLabel**. Additional XmPushButton translations for PushButtons in a menu system are described in the following list. In a Popup menu system, **BMenu** also performs the **BSelect** actions. These translations may not directly correspond to a translation table.

BSelect Press:	**BtnDown()**
BSelect Release:	**BtnUp()**
KActivate:	**ArmAndActivate()**
KSelect:	**ArmAndActivate()**
MAny KCancel:	**MenuShellPopdownOne()**

Action Routines

The **XmPushButton** action routines are

Activate(): This action draws the shadow in the unarmed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, **XmNlabelPixmap** is used for the button face. If the pointer is still within the button, this action calls the callbacks for **XmNactivateCallback**.

Arm(): This action arms the PushButton. It draws the shadow in the armed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, it fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, the **XmNarmPixmap** is used for the button face. It calls the **XmNarmCallback** callbacks.

ArmAndActivate():

In a menu, unposts all menus in the menu hierarchy and, unless the button is already armed, calls the **XmNarmCallback** callbacks. This action calls the **XmNactivateCallback** and **XmNdisarmCallback** callbacks.

Outside a menu, draws the shadow in the armed state and, if **XmNfillOnArm** is set to True, fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, **XmNarmPixmap** is used for the button face. This action calls the **XmNarmCallback** callbacks.

Outside a menu, this action also arranges for the following to happen, either immediately or at a later time: the shadow is drawn in the unarmed state and, if **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, **XmNlabelPixmap** is used for the button face. The **XmNactivateCallback** and **XmNdisarmCallback** callbacks are called.

BtnDown(): This action unposts any menus posted by the PushButton's parent menu, disables keyboard traversal for the menu, and enables mouse traversal for the menu. It draws the shadow in the armed state and, unless the button is already armed, calls the **XmNarmCallback** callbacks.

BtnUp(): This action unposts all menus in the menu hierarchy and activates the PushButton. It calls the **XmNactivateCallback** callbacks and then the **XmNdisarmCallback** callbacks.

Disarm(): Calls the callbacks for **XmNdisarmCallback**.

Help(): In a Pulldown or Popup MenuPane, unposts all menus in the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the menu system was entered. This action calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MenuShellPopdownOne():

In a top-level Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar; and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, it unposts the menu.

In a Popup MenuPane, this action unposts the menu and restores keyboard focus to the widget from which the menu was posted.

MultiActivate():

If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action increments *click_count* in the callback structure and draws the shadow in the unarmed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used for the button face. If the pointer is within the PushButton, calls the callbacks for **XmNactivateCallback** and **XmNdisarmCallback**.

MultiArm(): If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action draws the shadow in the armed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, this action fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, the **XmNarmPixmap** is used for the button face. This action calls the **XmNarmCallback** callbacks.

ProcessDrag():

Drags the contents of a PushButton label, identified when **BTransfer** is pressed. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT** for a label type of **XmSTRING**; otherwise, it is set to **PIXMAP** if the label type is **XmPIXMAP**. This action is undefined for PushButtons used in a menu system.

Additional Behavior

This widget has the following additional behavior:

<EnterWindow>:

In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the armed state and calls the **XmNarmCallback** callbacks.

If the PushButton is not in a menu and the cursor leaves and then reenters the PushButton's window while the button is pressed, this action draws the shadow in the armed state. If **XmNfillOnArm** is set to True, it also fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, **XmNarmPixmap** is used for the button face.

<LeaveWindow>:

In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the unarmed state and calls the **XmNdisarmCallback** callbacks.

If the PushButton is not in a menu and the cursor leaves the PushButton's window while the button is pressed, this action draws the shadow in the unarmed state. If **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used for the button face.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreatePushButton(3X)**, **XmLabel(3X)**, **XmPrimitive(3X)**, and **XmRowColumn(3X)**.

XmPushButtonGadget—The PushButtonGadget widget class

Synopsis

#include <Xm/PushBG.h>

Description

PushButtonGadget issues commands within an application. It consists of a text label or pixmap surrounded by a border shadow. When PushButtonGadget is selected, the shadow changes to give the appearance that the PushButtonGadget has been pressed in. When PushButtonGadget is unselected, the shadow changes to give the appearance that the PushButtonGadget is out.

The default behavior associated with a PushButtonGadget in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the PushButtonGadget. In addition, **BMenu** controls the behavior of the PushButtonGadget if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Thickness for a second shadow may be specified with the **XmNshowAsDefault** resource. If it has a nonzero value, the Label's **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** resources may be modified to accommodate the second shadow.

If an initial value is specified for **XmNarmPixmap** but not for **XmNlabelPixmap**, the **XmNarmPixmap** value is used for **XmNlabelPixmap**.

Classes

PushButtonGadget inherits behavior and resources from **Object**, **RectObj**, **XmGadget** and **XmLabelGadget**.

The class pointer is **xmPushButtonGadgetClass**.

The class name is **XmPushButtonGadget**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmPushButtonGadget		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNarmCallback XmCCallback	NULL XtCallbackList	C
XmNarmColor XmCArmColor	dynamic Pixel	CSG
XmNarmPixmap XmCArmPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNdefaultButtonShadowThickness XmCdefaultButtonShadowThickness	dynamic Dimension	CSG
XmNdisarmCallback XmCCallback	NULL XtCallbackList	C
XmNfillOnArm XmCFillOnArm	True Boolean	CSG
XmNmultiClick XmCMultiClick	dynamic unsigned char	CSG
XmNshowAsDefault XmCShowAsDefault	0 Dimension	CSG

XmNactivateCallback

Specifies the list of callbacks that is called when the PushButtonGadget is activated. It is activated when the user presses and releases the active mouse button while the pointer is inside the PushButtonGadget. Activating PushButtonGadget also disarms it. For this callback, the reason is **XmCR_ACTIVATE**.

XmNarmCallback

Specifies the list of callbacks that is called when PushButtonGadget is armed. It is armed when the user presses the active mouse button while the pointer is inside the PushButtonGadget. For this callback, the reason is **XmCR_ARM**.

XmNarmColor

Specifies the color with which to fill the armed button. **XmNfillOnArm** must be set to True for this resource to have an effect. The default for a color display is a color between the background and the bottom shadow color. For a monochrome display, the default is set to the foreground color, and any text in the label appears in the background color when the button is armed.

XmNarmPixmap

Specifies the pixmap to be used as the button face if **XmNlabeltype** is **XmPIXMAP** and PushButtonGadget is armed. This resource is disabled when the PushButtonGadget is in a menu.

XmNdefaultButtonShadowThickness

This resource specifies the width of the default button indicator shadow. If this resource is zero, the width of the shadow comes from the value of the **XmNshowAsDefault** resource. If this resource is greater than zero, the **XmNshowAsDefault** resource is only used to specify whether this button is the default. The default value is the initial value of **XmNshowAsDefault**.

XmNdisarmCallback

Specifies the list of callbacks that is called when the PushButtonGadget is disarmed. PushButtonGadget is disarmed when the user presses and releases the active mouse button while the pointer is inside that gadget. For this callback, the reason is **XmCR_DISARM**.

XmNfillOnArm

Forces the PushButtonGadget to fill the background of the button with the color specified by **XmNarmColor** when the button is armed and when this resource is set to True. If it is False, only the top and bottom shadow colors are switched. When the PushButtonGadget is in a menu, this resource is ignored and assumed to be False.

XmNmultiClick

If a button click is followed by another button click within the time span specified by the display's multiclick time, and this resource is set to **XmMULTICLICK_DISCARD**, the second click is not processed. If this resource is set to **XmMULTICLICK_KEEP**, the event is processed and *click_count* is incremented in the callback structure. When the button is in a menu, the default is **XmMULTICLICK_DISCARD**; otherwise, for a button not in a menu, the default value is **XmMULTICLICK_KEEP**.

XmNshowAsDefault

If **XmNdefaultButtonShadowThickness** is greater than 0 (zero), a value greater than zero in this resource specifies to mark this button as the default button. If **XmNdefaultButtonShadowThickness** is 0, a value greater than 0 in this resource specifies to mark this button as the default button with the shadow thickness specified by this resource. The space between the shadow and the default shadow is equal to the sum of both shadows. The default value is 0. When this value is not 0, the Label **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** resources may be modified to accommodate the second shadow. This resource is disabled when the PushButton is in a menu.

Inherited Resources

PushButtonGadget inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmLabelGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	CSG
XmNacceleratorText XmCAcceleratorText	NULL XmString	CSG
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	dynamic Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	dynamic Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	dynamic Dimension	CSG
XmNmarginRight XmCMarginRight	dynamic Dimension	CSG
XmNmarginTop XmCMarginTop	dynamic Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG

Name Class	Default Type	Access
XmNmnemonic XmCMnemonic	NULL KeySym	CSG
XmNmnemonicCharSet XmCMnemonicCharSet	dynamic String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	G
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	G
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	G
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

RectObj Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	0 Dimension	N/A
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    int         click_count;
} XmPushButtonCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

click_count Valid only when the reason is **XmCR_ACTIVATE**. It contains the number of clicks in the last multiclick sequence if the **XmNmultiClick** resource is set to **XmMULTICLICK_KEEP**; otherwise it contains 1. The activate callback is invoked for each click if **XmNmultiClick** is set to **XmMULTICLICK_KEEP**.

Behavior

XmPushButtonGadget includes behavior from **XmGadget**. **XmPushButtonGadget** includes menu traversal behavior from **XmLabelGadget**. Additional behavior for **XmPushButtonGadget** is described in the following list.

BTransfer Press:

Drags the contents of a PushButtonGadget label, identified when **BTransfer** is pressed. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT** for a label type of **XmSTRING**; otherwise, it is set to **PIXMAP** if the label type is **XmPIXMAP**. This action is undefined for PushButtonGadgets used in a menu system.

BSelect Press:

This action arms the PushButtonGadget.

In a menu, this action unposts any menus posted by the PushButtonGadget's parent menu, disables keyboard traversal for the menu, and enables mouse traversal for the menu. It draws the shadow in the armed state. Unless the button is already armed, it calls the **XmNarmCallback** callbacks.

If the button is not in a menu, this action draws the shadow in the armed state. If **XmNfillOnArm** is set to True, it fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, the **XmNarmPixmap** is used for the button face. It calls the **XmNarmCallback** callbacks.

BSelect Press 2+:

If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action draws the shadow in the armed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, it fills the button with the color

specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, the **XmNarmPixmap** is used for the button face. This action calls the **XmNarmCallback** callbacks.

BSelect Click or BSelect Release:
In a menu, this action unposts all menus in the menu hierarchy and activates the PushButtonGadget. It calls the **XmNactivateCallback** callbacks and then the **XmNdisarmCallback** callbacks.

If the PushButtonGadget is not in a menu, this action draws the shadow in the unarmed state. If **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used for the button face. If the pointer is still within the button, this action calls the callbacks for **XmNactivateCallback** and **XmNdisarmCallback**.

BSelect Release 2+:
If **XmNmultiClick** is **XmMULTICLICK_DISCARD**, this action does nothing.

If **XmNmultiClick** is **XmMULTICLICK_KEEP**, this action increments *click_count* in the callback structure and draws the shadow in the unarmed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, **XmNlabelPixmap** is used for the button face. If the pointer is within the PushButtonGadget, this action calls the callbacks for **XmNactivateCallback** and **XmNdisarmCallback**.

KActivate: In a menu, this action unposts all menus in the menu hierarchy, unless the button is already armed, and calls the **XmNarmCallback** callbacks, the **XmNactivateCallback** and the **XmNdisarmCallback** callbacks. Outside a menu, **KActivate** has no effect. For PushButtonGadgets outside of a menu, if the parent is a manager, this action passes the event to the parent.

KSelect: In a menu, this action unposts all menus in the menu hierarchy, unless the button is already armed, and calls the **XmNarmCallback** callbacks. This acton calls the **XmNactivateCallback** and **XmNdisarmCallback** callbacks.

Outside a menu, this action draws the shadow in the armed state and, if **XmNfillOnArm** is set to True, fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, **XmNarmPixmap** is used for the button face. This action calls the **XmNarmCallback** callbacks.

Outside a menu, this action also arranges for the following to happen, either immediately or at a later time: the shadow is drawn in the unarmed state and, if **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used for the button face. The **XmNactivateCallback** and **XmNdisarmCallback** callbacks are called.

KHelp: In a Pulldown or Popup MenuPane, unposts all menus in the menu hierarchy and restores keyboard focus to the widget that had the focus before the menu system was entered. This action calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MAny KCancel:
In a toplevel Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu.

In a Popup MenuPane, unposts the menu and restores keyboard focus to the widget from which the menu was posted. For a PushButtonGadget outside of a menu, if the parent is a manger, this action passes the event to the parent.

<Enter>: In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the armed state and calls the **XmNarmCallback** callbacks.

If the PushButtonGadget is not in a menu and the cursor leaves and then reenters the PushButtonGadget while the button is pressed, this action draws the shadow in the armed state. If **XmNfillOnArm** is

set to True, it also fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is **XmPIXMAP**, the **XmNarmPixmap** is used for the button face.

<Leave>: In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the unarmed state and calls the **XmNdisarmCallback** callbacks.

If the PushButtonGadget is not in a menu and the cursor leaves the PushButtonGadget while the button is pressed, this action draws the shadow in the unarmed state. If **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used for the button face.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Object(3X), **RectObj(3X)**, **XmCreatePushButtonGadget(3X)**, **XmGadget(3X)**, **XmLabelGadget(3X)**, and **XmRowColumn(3X)**.

XmRegisterSegmentEncoding—A compound string function that registers a compound text encoding format for a specified font list element tag

Synopsis

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

char * XmRegisterSegmentEncoding (fontlist_tag, ct_encoding)
        char        *fontlist_tag;
        char        *ct_encoding;
```

Description

XmRegisterSegmentEncoding registers a compound text encoding format with the specified font list element tag. The **XmCvtXmStringToCT** function uses this registry to map the font list tags of compound string segments to compound text encoding formats. Registering a font list tag that already exists in the registry overwrites the original entry. You can unregister a font list tag by passing a NULL value for the *ct_encoding* parameter.

fontlist_tag Specifies the font list element tag to be registered. The tag must be a NULL-terminated ISO8859-1 string.

ct_encoding Specifies the compound text character set to be used for segments with the font list tag. The value must be a NULL-terminated ISO8859-1 string. A value of **XmFONTLIST_DEFAULT_TAG** maps the specified font list tag to the code set of the locale.

Return Value

Returns NULL for a new font list tag or the old *ct_encoding* value for an already registered font list tag. The application is responsible for freeing the storage associated with the returned data (if any) by calling **XtFree**.

Related Information

XmCvtXmStringToCT(3X), **XmFontList(3X)**, **XmMapSegmentEncoding(3X)**, and **XmString(3X)**.

XmRemoveProtocolCallback—A VendorShell function that removes a callback from the internal list

Synopsis

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

void XmRemoveProtocolCallback (shell, property, protocol, callback, closure)
        Widget          shell;
        Atom            property;
        Atom            protocol;
        XtCallbackProc  callback;
        XtPointer       closure;

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

Description

XmRemoveProtocolCallback removes a callback from the internal list.

XmRemoveWMProtocolCallback is a convenience interface. It calls **XmRemoveProtocolCallback** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

property Specifies the protocol property

protocol Specifies the protocol atom (or an **int** cast to **Atom**)

callback Specifies the procedure to call when a protocol message is received

closure Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmInternAtom(3X)**, and **XmRemoveWMProtocolCallback(3X)**.

XmRemoveProtocols—A VendorShell function that removes the protocols from the protocol manager and deallocates the internal tables

Synopsis

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

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

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

Description

XmRemoveProtocols removes the protocols from the protocol manager and deallocates the internal tables. If any of the protocols are active, it will update the handlers and update the property if *shell* is realized.

XmRemoveWMProtocols is a convenience interface. It calls **XmRemoveProtocols** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

property Specifies the protocol property

protocols Specifies the protocol atoms (or **ints** cast to **Atom**)

num_protocols
Specifies the number of elements in protocols

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmInternAtom(3X)**, and **XmRemoveWMProtocols(3X)**.

XmRemoveTabGroup—A function that removes a tab group

Synopsis

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

void XmRemoveTabGroup (tab_group)
        Widget          tab_group;
```

Description

This function is obsolete and its behavior is replaced by setting **XmNnavigationType** to **XmNONE**. **XmRemoveTabGroup** removes a widget from the list of tab groups associated with a particular widget hierarchy and sets the widget's **XmNnavigationType** to **XmNONE**.

tab_group Specifies the widget ID

Related Information

XmAddTabGroup(3X), **XmManager(3X)**, and **XmPrimitive(3X)**.

XmRemoveWMProtocolCallback—A VendorShell convenience interface that removes a callback from the internal list

Synopsis

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

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

Description

XmRemoveWMProtocolCallback is a convenience interface. It calls **XmRemoveProtocolCallback** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

protocol Specifies the protocol atom (or an **int** type cast to **Atom**)

callback Specifies the procedure to call when a protocol message is received

closure Specifies the client data to be passed to the callback when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmInternAtom(3X)**, and **XmRemoveProtocolCallback(3X)**.

XmRemoveWMProtocols—A VendorShell convenience interface that removes the protocols from the protocol manager and deallocates the internal tables

Synopsis

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

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

Description

XmRemoveWMProtocols is a convenience interface. It calls **XmRemoveProtocols** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

protocols Specifies the protocol atoms (or **ints** cast to **Atom**)

num_protocols
Specifies the number of elements in protocols

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmInternAtom(3X)**, and **XmRemoveProtocols(3X)**.

XmRepTypeAddReverse—A representation type manager function that installs the reverse converter for a previously registered representation type

Synopsis

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

void XmRepTypeAddReverse (rep_type_id)
        XmRepTypeId rep_type_id;
```

Description

XmRepTypeAddReverse installs the reverse converter for a previously registered representation type. The reverse converter takes a numerical representation type value and returns its corresponding string value. Certain applications may require this capability to obtain a string value to display on a screen or to build a resource file.

The *values* argument of the **XmRepTypeRegister** function can be used to register representation types with nonconsecutive values or with duplicate names for the same value. If the list of numerical values for a representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value. For example, if a *value_names* array has **cancel**, **proceed**, and **abort**, and the corresponding *values* array contains 0, 1, and 0, the reverse converter will return **cancel** instead of **abort** for an input value of 0.

rep_type_id Specifies the identification number of the representation type

Related Information

XmRepTypeGetId(3X) and **XmRepTypeRegister(3X)**.

XmRepTypeGetId—A representation type manager function that retrieves the identification number of a representation type

Synopsis

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

XmRepTypeId XmRepTypeGetId (rep_type)
        String          rep_type;
```

Description

XmRepTypeGetId searches the registration list for the specified representation type and returns the associated identification number.

rep_type Specifies the representation type for which an identification number is requested

Return Value

Returns the identification number of the specified representation type. If the representation type is not registered, the function returns **XmREP_TYPE_INVALID**.

Related Information

XmRepTypeGetRegistered(3X) and **XmRepTypeRegister(3X)**.

XmRepTypeGetNameList—A representation type manager function that generates a list of values for a representation type

Synopsis

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

String * XmRepTypeGetNameList (rep_type_id, use_uppercase_format)
        XmRepTypeId     rep_type_id;
        Boolean         use_uppercase_format;
```

Description

XmRepTypeGetNameList generates a NULL-terminated list of the value names associated with the specified representation type. Each value name is a NULL-terminated string. This routine allocates memory for the returned data. The application must free this memory using **XtFree**.

rep_type_id Specifies the identification number of the representation type.

use_uppercase_format
Specifies a Boolean value that controls the format of the name list. If the value is True, each value name is in uppercase characters prefixed by **Xm**; if it is False, the names are in lowercase characters.

Return Value

Returns a pointer to an array of the value names.

Related Information

XmRepTypeGetId(3X), **XmRepTypeGetRegistered(3X)**, and **XmRepTypeRegister(3X)**.

XmRepTypeGetRecord—A representation type manager function that returns information about a representation type

Synopsis

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

XmRepTypeEntry XmRepTypeGetRecord (rep_type_id)
        XmRepTypeId rep_type_id;
```

Description

XmRepTypeGetRecord retrieves information about a particular representation type that is registered with the representation type manager. This routine allocates memory for the returned data. The application must free this memory using **XtFree**.

rep_type_id The identification number of the representation type

The representation type entry structure contains the following information:

```
typedef struct
{
    String          rep_type_name;
    String          *value_names;
    unsigned char   *values;
    unsigned char   num_values;
    Boolean         reverse_installed;
    XmRepTypeId     rep_type_id;
} XmRepTypeEntryRec, *XmRepTypeEntry ;
```

rep_type_name
The name of the representation type

value_names An array of representation type value names

values An array of representation type numerical values

num_values The number of values associated with the representation type

reverse_installed
A flag that indicates whether or not the reverse converter is installed

rep_type_id The identification number of the representation type

Return Value

Returns a pointer to the representation type entry structure that describes the representation type.

Related Information

XmRepTypeGetId(3X), **XmRepTypeGetRegistered(3X)**, and **XmRepTypeRegister(3X)**.

XmRepTypeGetRegistered—A representation type manager function that returns a copy of the registration list

Synopsis

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

XmRepTypeList XmRepTypeGetRegistered ()
```

Description

XmRepTypeGetRegistered retrieves information about all representation types that are registered with the representation type manager. The registration list is an array of structures, each of which contains information for a representation type entry. The end of the registration list is marked with a representation type entry whose *rep_type_name* field has a NULL pointer. This routine allocates memory for the returned data. The application must free this memory using **XtFree**.

The representation type entry structure contains the following information:

```
typedef struct
{
    String          rep_type_name;
    String          *value_names;
    unsigned char   *values;
    unsigned char   num_values;
    Boolean         reverse_installed;
    XmRepTypeId     rep_type_id;
} XmRepTypeEntryRec, *XmRepTypeList ;
```

rep_type_name
: The name of the representation type

value_names
: An array of representation type value names

values
: An array of representation type numerical values

num_values
: The number of values associated with the representation type

reverse_installed
: A flag that indicates whether or not the reverse converter is installed

rep_type_id
: The identification number of the representation type

Return Value

Returns a pointer to the registration list of representation types.

Related Information

XmRepTypeRegister(3X) and **XmRepTypeGetRecord(3X)**.

XmRepTypeInstallTearOffModelConverter—A representation type manager function that installs the resource converter for **XmNtearOffModel**.

Synopsis

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

void XmRepTypeInstallTearOffModelConverter ()
```

Description

XmRepTypeInstallTearOffModelConverter installs the resource converter that allows values for the **XmNtearOffModel** resource to be specified in resource default files.

Related Information

XmRowColumn(3X).

XmRepTypeRegister—A representation type manager function that registers a representation type resource

Synopsis

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

XmRepTypeId XmRepTypeRegister (rep_type, value_names, values, num_values)
        String          rep_type;
        String          *value_names;
        unsigned char   *values;
        unsigned char   num_values;
```

Description

XmRepTypeRegister registers a representation type resource with the representation type manager. All features of the representation type management facility become available for the specified representation type. The function installs a forward type converter to convert string values to numerical representation type values.

When the *values* argument is NULL, consecutive numerical values are assumed. The order of the strings in the *value_names* array determines the numerical values for the resource. For example, the first value name is 0 (zero); the second value name is 1; and so on.

If it is non-NULL, the *values* argument can be used to assign values to representation types that have nonconsecutive values or have duplicate names for the same value. Representation types registered in this manner will consume additional storage and will be slightly slower than representation types with consecutive values.

A representation type can only be registered once; if the same representation type name is registered more than once, the behavior is undefined.

The function **XmRepTypeAddReverse** installs a reverse converter for a registered representation type. The reverse converter takes a representation type numerical value and returns the corresponding string value. If the list of numerical values for a representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value. For example, if a *value_names* array has **cancel**, **proceed**, and **abort**, and the corresponding *values* array contains 0, 1, and 0, the reverse converter will return **cancel** instead of **abort** for an input value of 0.

rep_type Specifies the representation type name.

value_names Specifies a pointer to an array of value names associated with the representation type. A value name is specified in lowercase characters without an **Xm** prefix. Words within a name are separated with underscores.

values Specifies a pointer to an array of values associated with the representation type. A value in this array is associated with the value name in the corresponding position of the *value_names* array.

num_values Specifies the number of entries in the *value_names* and *values* arrays.

Return Value

Returns the identification number for the specified representation type.

Related Information

XmRepTypeAddReverse(3X), **XmRepTypeGetId(3X)**, **XmRepTypeGetNameList(3X)**, **XmRepTypeGetRecord(3X)**, **XmRepTypeGetRegistered(3X)**, and **XmRepTypeValidValue(3X)**.

XmRepTypeValidValue—A representation type manager function that tests the validity of a numerical value of a representation type resource

Synopsis

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

Boolean XmRepTypeValidValue (rep_type_id, test_value, enable_default_warning)
        XmRepTypeId     rep_type_id;
        unsigned char   test_value;
        Widget          enable_default_warning;
```

Description

XmRepTypeValidValue tests the validity of a numerical value for a given representation type resource. The function generates a default warning message if the value is invalid and the *enable_default_warning* argument is non-NULL.

rep_type_id Specifies the identification number of the representation type.

test_value Specifies the numerical value to test.

enable_default_warning
Specifies the ID of the widget that contains a default warning message. If this parameter is NULL, no default warning message is generated and the application must provide its own error handling.

Return Value

Returns True if the specified value is valid; otherwise, returns False.

Related Information

XmRepTypeGetId(3X) and **XmRepTypeRegister(3X)**.

XmResolveAllPartOffsets—A function that allows writing of upward-compatible applications and widgets

Synopsis

#include <Xm/XmP.h>

void XmResolveAllPartOffsets (*widget_class, offset, constraint_offset*)
WidgetClass *widget_class*;
XmOffsetPtr * *offset*;
XmOffsetPtr * *constraint_offset*;

Description

The use of offset records requires two extra global variables per widget class. The variables consist of pointers to arrays of offsets into the widget record and constraint record for each part of the widget structure. The **XmResolveAllPartOffsets** function allocates the offset records needed by an application to guarantee upward-compatible access to widget instance and constraint records by applications and widgets. These offset records are used by the widget to access all of the widget's variables. A widget needs to take the steps described in the following paragraphs.

Instead of creating a resource list, the widget creates an offset resource list. To accomplish this, use the **XmPartResource** structure and the **XmPartOffset** macro. The **XmPartResource** data structure looks just like a resource list but, instead of having one integer for its offset, it has two shorts. This structure is put into the class record as if it were a normal resource list. Instead of using **XtOffset** for the offset, the widget uses **XmPartOffset**.

If the widget is a subclass of the Constraint class and it defines additional constraint resources, create an offset resource list for the constraint part as well. Instead of using **XtOffset** for the offset, the widget uses **XmConstraintPartOffset** in the constraint resource list.

```
XmPartResource resources[] = {
   {  BarNxyz, BarCXyz, XmRBoolean, sizeof(Boolean),
      XmPartOffset(Bar,xyz), XmRImmediate, (XtPointer)False } };

XmPartResource constraints[] = {
   {  BarNmaxWidth, BarNMaxWidth,
         XmRDimension, sizeof(Dimension),
         XmConstraintPartOffset(Bar,max_width),
         XmRImmediate, (XtPointer)100 } };
```

Instead of putting the widget size in the class record, the widget puts the widget part size in the same field. If the widget is a subclass of the Constraint class, instead of putting the widget constraint record size in the class record, the widget puts the widget constraint part size in the same field.

Instead of putting **XtVersion** in the class record, the widget puts **XtVersionDontCheck** in the class record.

Define a variable, of type **XmOffsetPtr**, to point to the offset record. If the widget is a subclass of the Constraint class, define a variable of type **XmOffsetPtr** to point to the constraint offset record. These can be part of the widget's class record or separate global variables.

In class initialization, the widget calls **XmResolveAllPartOffsets**, passing it pointers to the class record, the address of the offset record, and the address of the constraint offset record. If the widget not is a subclass of the Constraint class, it should pass NULL as the address of the constraint offset record. This does several things:

- Adds the superclass (which, by definition, has already been initialized) size field to the part size field
- If the widget is a subclass of the Constraint class, adds the superclass constraint size field to the constraint size field
- Allocates an array based upon the number of superclasses
- If the widget is a subclass of the constraint class, allocates an array for the constraint offset record
- Fills in the offsets of all the widget parts and constraint parts with the appropriate values, determined by examining the size fields of all superclass records
- Uses the part offset array to modify the offset entries in the resource list to be real offsets, in place

The widget defines a constant that will be the index to its part structure in the offsets array. The value should be 1 greater than the index of the widget's superclass. Constants defined for all **Xm** widgets can be found in **XmP.h**.

```
#define BarIndex (XmBulletinBIndex + 1)
```

Instead of accessing fields directly, the widget must always go through the offset table. The **XmField** and **XmConstraintField** macros help you access these fields. Because the **XmPartOffset**, **XmConstraintPartOffset**, **XmField**, and **XmConstraintField** macros concatenate things, you must ensure that there is no space after the part argument. For example, the following macros do not work because of the space after the part (Label) argument:

```
XmField(w, offset, Label , text, char *)
XmPartOffset(Label , text).
```

Therefore, you must not have any spaces after the part (Label) argument, as illustrated here:

```
XmField(w, offset, Label, text, char *)
```

You can define macros for each field to make this easier. Assume an integer field *xyz*:

```
#define BarXyz(w) (*(int *)(((char *) w) + \
   offset[BarIndex] + XtOffset(BarPart,xyz)))
```

For constraint field *max_width*:

```
#define BarMaxWidth(w) \
   XmConstraintField(w,constraint_offsets,Bar,max_width,Dimension)
```

The parameters for **XmResolveAllPartOffsets** are

widget_class Specifies the widget class pointer for the created widget

offset Returns the offset record

constraint_offset
Returns the constraint offset record

Related Information

XmResolvePartOffsets(3X).

XmResolvePartOffsets—A function that allows writing of upward-compatible applications and widgets

Synopsis

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

void XmResolvePartOffsets (widget_class, offset)
        WidgetClass    widget_class;
        XmOffsetPtr    * offset;
```

Description

The use of offset records requires one extra global variable per widget class. The variable consists of a pointer to an array of offsets into the widget record for each part of the widget structure. The **XmResolvePartOffsets** function allocates the offset records needed by an application to guarantee upward-compatible access to widget instance records by applications and widgets. These offset records are used by the widget to access all of the widget's variables. A widget needs to take the steps described in the following paragraphs.

Instead of creating a resource list, the widget creates an offset resource list. To accomplish this, use the **XmPartResource** structure and the **XmPartOffset** macro. The **XmPartResource** data structure looks just like a resource list but, instead of having one integer for its offset, it has two shorts. This structure is put into the class record as if it were a normal resource list. Instead of using **XtOffset** for the offset, the widget uses **XmPartOffset**.

```
XmPartResource resources[] = {
  { BarNxyz, BarCXyz, XmRBoolean,
    sizeof(Boolean), XmPartOffset(Bar,xyz),
    XmRImmediate, (XtPointer)False }
};
```

Instead of putting the widget size in the class record, the widget puts the widget part size in the same field.

Instead of putting **XtVersion** in the class record, the widget puts **XtVersionDontCheck** in the class record.

The widget defines a variable, of type **XmOffsetPtr**, to point to the offset record. This can be part of the widget's class record or a separate global variable.

In class initialization, the widget calls **XmResolvePartOffsets**, passing it a pointer to contain the address of the offset record and the class record. This does several things:

- Adds the superclass (which, by definition, has already been initialized) size field to the part size field
- Allocates an array based upon the number of superclasses
- Fills in the offsets of all the widget parts with the appropriate values, determined by examining the size fields of all superclass records
- Uses the part offset array to modify the offset entries in the resource list to be real offsets, in place

The widget defines a constant that will be the index to its part structure in the offsets array. The value should be 1 greater than the index of the widget's superclass. Constants defined for all **Xm** widgets can be found in **XmP.h**.

```
#define BarIndex (XmBulletinBIndex + 1)
```

Instead of accessing fields directly, the widget must always go through the offset table. The **XmField** macro helps you access these fields. Because the **XmPartOffset** and **XmField** macros concatenate things together, you must ensure that there is no space after the part argument. For example, the following macros do not work because of the space after the part (Label) argument:

```
XmField(w, offset, Label , text, char *)
XmPartOffset(Label , text)
```

Therefore, you must not have any spaces after the part (Label) argument, as illustrated here:

```
XmField(w, offset, Label, text, char *)
```

You can define macros for each field to make this easier. Assume an integer field *xyz*:

```
#define BarXyz(w) (*(int *)(((char *) w) + \
   offset[BarIndex] + XtOffset(BarPart,xyz)))
```

The parameters for **XmResolvePartOffsets** are

widget_class	Specifies the widget class pointer for the created widget
offset	Returns the offset record

Related Information

XmResolveAllPartOffsets(3X).

XmRowColumn—The RowColumn widget class

Synopsis

#include <Xm/RowColumn.h>

Description

The RowColumn widget is a general-purpose RowColumn manager capable of containing any widget type as a child. In general, it requires no special knowledge about how its children function and provides nothing beyond support for several different layout styles. However, it can be configured as a menu, in which case, it expects only certain children, and it configures to a particular layout. The menus supported are MenuBar, Pulldown or Popup MenuPanes, and OptionMenu.

The type of layout performed is controlled by how the application has set the various layout resources. It can be configured to lay out its children in either rows or columns. In addition, the application can specify how the children are laid out, as follows:

- The children are packed tightly together into either rows or columns.
- Each child is placed in an identically sized box (producing a symmetrical look).
- A particular layout is specified (the current *x* and *y* positions of the children control their location).

In addition, the application has control over both the spacing that occurs between each row and column and the margin spacing present between the edges of the RowColumn widget and any children that are placed against it.

In a MenuBar, Pulldown MenuPane, or Popup MenuPane the default for the **XmNshadowThickness** resource is 2. In an OptionMenu or a WorkArea, (such as a RadioBox or CheckBox) this resource is not applicable and its use is undefined. If an application wishes to place a 3-D shadow around an OptionMenu or WorkArea, it can create the RowColumn as a child of a Frame widget.

In a MenuBar, Pulldown MenuPane, or Popup MenuPane the **XmNnavigationType** resource is not applicable and its use is undefined. In a WorkArea, the default for **XmNnavigationType** is **XmTAB_GROUP**. In an OptionMenu the default for **XmNnavigationType** is **XmNONE**.

In a MenuBar, Pulldown MenuPane, or Popup MenuPane the **XmNtraversalOn** resource is not applicable and its use is undefined. In an OptionMenu or WorkArea, the default for **XmNtraversalOn** is True.

If the parent of the RowColumn is a MenuShell, the **XmNmappedWhenManaged** resource is forced to False when the widget is realized.

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. The following list identifies the names of these widgets (or gadgets) and the associated OptionMenu areas.

Option Menu Label Gadget	**OptionLabel**
Option Menu Cascade Button	**OptionButton**

Tear-off Menus

Pulldown and Popup MenuPanes support tear-off menus, which enable the user to retain a MenuPane on the display to facilitate subsequent menu selections. A MenuPane that can be torn off is identified by a tear-off button that spans the width of the MenuPane and displays a dashed line. A torn-off MenuPane contains a window manager system menu icon and a title bar. The window title displays the label of the cascade button that initiated the action when the label type is **XmSTRING**. If the label contains a pixmap the window title is empty. A tear-off menu from a Popup MenuPane also displays an empty title.

The user can tear off a MenuPane using the mouse or keyboard. Clicking **BSelect** or pressing **KActivate** (or **KSelect**) on the tear-off button, tears off the MenuPane at the current position. Pressing **BTransfer** on the tear-off button tears off the MenuPane and allows the user to drag the torn-off menu to a new position designated by releasing the mouse button. Tearing off a MenuPane unposts the current active menu. Only one tear-off copy for each MenuPane is allowed. Subsequent tear-off actions of a torn MenuPane unpost the existing copy first.

The name of the tear-off button of a torn-off menu pane is **TearOffControl**. The name can be used to set resources in a resource file. An application can also obtain the tear-off button itself using **XmGetTearOffControl** and then set resource values by calling **XtSetValues**.

The tear-off button has Separator-like behavior. Its appearance can be specified with the following tear-off button resources: **XmNbackground**, **XmNbackgroundPixmap**, **XmNbottomShadowColor**, **XmNforeground**, **XmNheight**, **XmNmargin**, **XmNseparatorType**, **XmNshadowThickness**, and **XmNtopShadowColor**. Refer to the **XmSeparator** reference page for a complete description of each of these resources.

The **XmNtearOffModel**, **XmNtearOffMenuActivateCallback**, and **XmNtearOffMenuDeactivateCallback** are RowColumn resources that affect tear-off menu behavior.

By default, menus do not tear off. Setting the **XmNtearOffModel** resource to **XmTEAR_OFF_ENABLED** enables tear-off functionality. There is no resource converter preregistered for **XmNtearOffModel**. To allow tear-off functionality to be enabled through the resource database, call the function **XmRepTypeInstallTearOffModelConverter**.

Tear-off menu focus policy follows standard window manager policy. It is recommended that the **startupKeyFocus** and **autoKeyFocus mwm** resources be set to True.

Classes

RowColumn inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is **xmRowColumnWidgetClass**.

The class name is **XmRowColumn**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmRowColumn Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNadjustLast XmCAdjustLast	True Boolean	CSG
XmNadjustMargin XmCAdjustMargin	True Boolean	CSG
XmNentryAlignment XmCAlignment	XmALIGNMENT_BEGINNING unsigned char	CSG
XmNentryBorder XmCEntryBorder	0 Dimension	CSG
XmNentryCallback XmCCallback	NULL XtCallbackList	C
XmNentryClass XmCEntryClass	dynamic WidgetClass	CSG
XmNentryVerticalAlignment XmCVerticalAlignment	XmALIGNMENT_CENTER unsigned char	CSG
XmNisAligned XmCIsAligned	True Boolean	CSG
XmNisHomogeneous XmCIsHomogeneous	dynamic Boolean	CG
XmNlabelString XmCXmString	NULL XmString	C
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	dynamic Dimension	CSG
XmNmarginWidth XmCMarginWidth	dynamic Dimension	CSG
XmNmenuAccelerator XmCAccelerators	dynamic String	CSG
XmNmenuHelpWidget XmCMenuWidget	NULL Widget	CSG

Name Class	Default Type	Access
XmNmenuHistory XmCMenuWidget	NULL Widget	CSG
XmNmenuPost XmCMenuPost	NULL String	CSG
XmNmnemonic XmCMnemonic	NULL KeySym	CSG
XmNmnemonicCharSet XmCMnemonicCharSet	XmFONTLIST_DEFAULT_TAG String	CSG
XmNnumColumns XmCNumColumns	1 short	CSG
XmNorientation XmCOrientation	dynamic unsigned char	CSG
XmNpacking XmCPacking	dynamic unsigned char	CSG
XmNpopupEnabled XmCPopupEnabled	True Boolean	CSG
XmNradioAlwaysOne XmCRadioAlwaysOne	True Boolean	CSG
XmNradioBehavior XmCRadioBehavior	False Boolean	CSG
XmNresizeHeight XmCResizeHeight	True Boolean	CSG
XmNresizeWidth XmCResizeWidth	True Boolean	CSG
XmNrowColumnType XmCRowColumnType	XmWORK_AREA unsigned char	CG
XmNspacing XmCSpacing	dynamic Dimension	CSG
XmNsubMenuId XmCMenuWidget	NULL Widget	CSG
XmNtearOffMenuActivateCallback XmCCallback	NULL XtCallbackList	C

Name Class	Default Type	Access
XmNtearOffMenuDeactivateCallback XmCCallback	NULL XtCallbackList	C
XmNtearOffModel XmCTearOffModel	XmTEAR_OFF_DISABLED unsigned char	CSG
XmNunmapCallback XmCCallback	NULL XtCallbackList	C
XmNwhichButton XmCWhichButton	dynamic unsigned int	CSG

XmNadjustLast

Extends the last row of children to the bottom edge of RowColumn (when **XmNorientation** is **XmHORIZONTAL**) or extends the last column to the right edge of RowColumn (when **XmNorientation** is **XmVERTICAL**). Setting **XmNadjustLast** to False disables this feature.

XmNadjustMargin

Specifies whether the inner minor margins of all items contained within the RowColumn widget are forced to the same value. The inner minor margin corresponds to the **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** resources supported by **XmLabel** and **XmLabelGadget**.

A horizontal orientation causes **XmNmarginTop** and **XmNmarginBottom** for all items in a particular row to be forced to the same value; the value is the largest margin specified for one of the Label items.

A vertical orientation causes **XmNmarginLeft** and **XmNmarginRight** for all items in a particular column to be forced to the same value; the value is the largest margin specified for one of the Label items.

This keeps all text within each row or column lined up with all other text in its row or column. If **XmNrowColumnType** is either **XmMENU_POPUP** or **XmMENU_PULLDOWN** and this resource is True, only button children have their margins adjusted.

XmNentryAlignment

Specifies the alignment type for children that are subclasses of **XmLabel** or **XmLabelGadget** when **XmNisAligned** is enabled. The following are textual alignment types:

- **XmALIGNMENT_BEGINNING** (default)
- **XmALIGNMENT_CENTER**
- **XmALIGNMENT_END**

See the description of **XmNalignment** in the **XmLabel(3X)** reference page for an explanation of these actions.

XmNentryBorder

Imposes a uniform border width upon all RowColumn's children. The default value is 0 (zero), which disables the feature.

XmNentryCallback

Disables the **XmNactivateCallback** and **XmNvalueChangedCallback** callbacks for all CascadeButton, DrawnButton, PushButton, and ToggleButton widgets and gadgets contained within the RowColumn widget. If the application supplies this resource, the **XmNactivateCallback** and **XmNvalueChangedCallback** callbacks are then revectored to the **XmNentryCallback** callbacks. This allows an application to supply a single callback routine for handling all items contained in a RowColumn widget. The callback reason is **XmCR_ACTIVATE**. If the application does not supply this resource, the **XmNactivateCallback** and **XmNvalueChangedCallback** callbacks for each item in the RowColumn widget work as normal.

The application must supply this resource when this widget is created. Changing this resource using the **XtSetValues** is not supported.

XmNentryClass

Specifies the only widget class that can be added to the RowColumn widget; this resource is meaningful only when the **XmNisHomogeneous** resource is set to True. Both widget and gadget variants of the specified class may be added to the widget.

When **XmCreateRadioBox** is called or when **XmNrowColumnType** is set to **XmWORK_AREA** and **XmNradioBehavior** is True, the default value of **XmNentryClass**

is **xmToggleButtonGadgetClass**. When **XmNrowColumnType** is set to **XmMENU_BAR**, the value of **XmNentryClass** is forced to **xmCascadeButtonWidgetClass**.

XmNentryVerticalAlignment

Specifies the type of vertical alignment for children that are subclasses of **XmLabel**, **XmLabelGadget, and XmText**. This resource is invalid if **XmNorientation** is **XmVERTICAL** and **XmNpacking** is **XmPACK_TIGHT**, because this layout preserves variable heights among the children. The vertical alignment types include:

XmALIGNMENT_BASELINE_BOTTOM

Causes the bottom baseline of all children in a row to be aligned. This resource is applicable only when all children in a row contain textual data.

XmALIGNMENT_BASELINE_TOP

Causes the top baseline of all children in a row to be aligned. This resource is applicable only when all children in a row contain textual data.

XmALIGNMENT_CONTENTS_BOTTOM

Causes the bottom of the contents (text or pixmap) of all children in a row to be aligned.

XmALIGNMENT_CENTER

Causes the center of all children in a row to be aligned.

XmALIGNMENT_CONTENTS_TOP

Causes the top of the contents (text or pixmap) of all children in a row to be aligned.

XmNisAligned

Specifies text alignment for each item within the RowColumn widget; this applies only to items that are subclasses of **XmLabel** or **XmLabelGadget**. However, if the item is a Label widget or gadget and its parent is either a Popup MenuPane or a Pulldown MenuPane, alignment is not performed; the Label is treated as the title within the MenuPane, and the alignment set by the application is not overridden. **XmNentryAlignment** controls the type of textual alignment.

XmNisHomogeneous

Indicates whether the RowColumn widget should enforce exact homogeneity among the items it contains; if this resource is set to True, only the widgets that are of the class indicated by **XmNentryClass** are allowed as children of the RowColumn widget. This is most often used when creating a MenuBar. Attempting to insert a child that is not a member of the specified class generates a warning message.

In a MenuBar the value of **XmNisHomogeneous** is forced to True. In an OptionMenu the value is forced to False. When **XmCreateRadioBox** is called the default value is True. Otherwise, the default value is False.

XmNlabelString

Points to a text string that displays the label to one side of the selection area when **XmNrowColumnType** is set to **XmMENU_OPTION**. This resource is not meaningful for all other RowColumn types. If the application wishes to change the label after creation, it must get the LabelGadget ID (**XmOptionLabelGadget**) and call **XtSetValues** on the LabelGadget directly. The default value is no label.

XmNmapCallback

Specifies a widget-specific callback function that is invoked when the window associated with the RowColumn widget is about to be mapped. The callback reason is **XmCR_MAP**.

XmNmarginHeight

Specifies the amount of blank space between the top edge of the RowColumn widget and the first item in each column, and the bottom edge of the RowColumn widget and the last item in each column. The default value is 0 (zero) for Pulldown and Popup MenuPanes, and 3 pixels for other RowColumn types.

XmNmarginWidth

Specifies the amount of blank space between the left edge of the RowColumn widget and the first item in each row, and the right edge of the RowColumn widget and the last item in each row. The default value is 0 (zero) for Pulldown and Popup MenuPanes, and 3 pixels for other RowColumn types.

XmNmenuAccelerator

This resource is useful only when the RowColumn widget has been configured to operate as a Popup MenuPane or a MenuBar. The format of this resource is similar to the left side specification of a

translation string, with the limitation that it must specify a key event. For a Popup MenuPane, when the accelerator is typed by the user, the Popup MenuPane is posted. For a MenuBar, when the accelerator is typed by the user, the first item in the MenuBar is highlighted, and traversal is enabled in the MenuBar. The default for a Popup MenuPane is **KMenu**. The default for a MenuBar is **KMenuBar**. Setting the **XmNpopupEnabled** resource to False disables the accelerator.

XmNmenuHelpWidget

Specifies the widget ID for the CascadeButton, which is treated as the Help widget if **XmNrowColumnType** is set to **XmMENU_BAR**. The MenuBar always places the Help widget at the bottom right corner (in a left to right environment) of the MenuBar. If the RowColumn widget is any type other than **XmMENU_BAR**, this resource is not meaningful.

XmNmenuHistory

Specifies the widget ID of the last menu entry to be activated. It is also useful for specifying the current selection for an OptionMenu. If **XmNrowColumnType** is set to **XmMENU_OPTION**, the specified menu item is positioned under the cursor when the menu is displayed.

If the RowColumn widget has the **XmNradioBehavior** resource set to True, the widget field associated with this resource contains the widget ID of the last ToggleButton or ToggleButtonGadget to change from unselected to selected. The default value is the widget ID of the first child in the widget.

XmNmenuPost

Specifies an X event description indicating a button event that posts a menu system. The default for **XmMENU_POPUP** is **BMenu Press**. The default for **XmMENU_OPTION**, **XmMENU_BAR**, and **XmWORK_AREA** is **BSelect Press**. The **XmNmenuPost** resource for pulldowns should be consistent with that of the top-level parent menu (although the event type is ignored). Setting this resource to **BTransfer Press** will conflict with drag and drop operations, which use **BTransfer Press** as a default button binding.

XmNmnemonic

This resource is useful only when **XmNrowColumnType** is set to **XmMENU_OPTION**. It specifies a keysym for a key that, when pressed by the user along with the **MAlt** modifier, posts the associated Pulldown MenuPane. The first character in the OptionMenu label string that exactly matches the mnemonic in the

character set specified in **XmNmnemonicCharSet** is underlined. The user can post the menu by pressing either the shifted or the unshifted mnemonic key. The default is no mnemonic.

XmNmnemonicCharSet
Specifies the character set of the mnemonic for an OptionMenu. The default is **XmFONTLIST_DEFAULT_TAG**. If the RowColumn widget is any type other than **XmMENU_OPTION**, this resource is not meaningful.

XmNnumColumns
Specifies the number of minor dimension extensions that are made to accommodate the entries; this attribute is meaningful only when **XmNpacking** is set to **XmPACK_COLUMN**.

For vertically oriented RowColumn widgets, this attribute indicates how many columns are built; the number of entries per column is adjusted to maintain this number of columns, if possible.

For horizontally oriented RowColumn widgets, this attribute indicates how many rows are built.

The default value is 1. In an OptionMenu the value is forced to 1. The value must be greater than 0 (zero).

XmNorientation
Determines whether RowColumn layouts are row-major or column-major. In a column-major layout, the children of the RowColumn are laid out in columns top to bottom within the widget. In a row-major layout the children of the RowColumn are laid out in rows. The **XmVERTICAL** resource value selects a column-major layout. **XmHORIZONTAL** selects a row-major layout.

When creating a MenuBar or an OptionMenu, the default is **XmHORIZONTAL**. Otherwise, the default value is **XmVERTICAL**. The results of specifying a value of **XmVERTICAL** for a MenuBar are undefined.

XmNpacking Specifies how to pack the items contained within a RowColumn widget. This can be set to **XmPACK_TIGHT, XmPACK_COLUMN** or **XmPACK_NONE**. When a RowColumn widget packs the items it contains, it determines its major dimension using the value of the **XmNorientation** resource.

XmPACK_TIGHT indicates that given the current major dimension (for example, vertical if **XmNorientation** is **XmVERTICAL**), entries are placed one after the other until the RowColumn widget must wrap. RowColumn wraps when there is no room left for a complete child in that dimension. Wrapping occurs by beginning a new row or column in the next available space. Wrapping continues, as often as necessary, until all of the children are laid out. In the vertical dimension (columns), boxes are set to the same width; in the horizontal dimension (rows), boxes are set to the same depth. Each entry's position in the major dimension is left unaltered (for example, **XmNy** is left unchanged when **XmNorientation** is **XmVERTICAL**); its position in the minor dimension is set to the same value as the greatest entry in that particular row or column. The position in the minor dimension of any particular row or column is independent of all other rows or columns.

XmPACK_COLUMN indicates that all entries are placed in identically sized boxes. The boxes are based on the largest height and width values of all the children widgets. The value of the **XmNnumColumns** resource determines how many boxes are placed in the major dimension, before extending in the minor dimension.

XmPACK_NONE indicates that no packing is performed. The *x* and *y* attributes of each entry are left alone, and the RowColumn widget attempts to become large enough to enclose all entries.

When **XmCreateRadioBox** is called or when **XmNrowColumnType** is set to **XmWORK_AREA** and **XmNradioBehavior** is True, the default value of **XmNpacking** is **XmPACK_COLUMN**. In an OptionMenu the value is initialized to **XmPACK_TIGHT**. Otherwise, the value defaults to **XmPACK_TIGHT**.

XmNpopupEnabled

Allows the menu system to enable keyboard input (accelerators and mnemonics) defined for the Popup MenuPane and any of its submenus. The Popup MenuPane needs to be informed whenever its accessibility to the user changes because posting of the Popup MenuPane is controlled by the application. The default value of this resource is True (keyboard input—accelerators and mnemonics—defined for the Popup MenuPane and any of its submenus is enabled).

XmNradioAlwaysOne

If True, forces the active ToggleButton or ToggleButtonGadget to be automatically selected after having been unselected (if no other toggle was activated). If False, the active toggle may be unselected. The default value is True. This resource is important only when **XmNradioBehavior** is True.

The application can always add and subtract toggles from RowColumn regardless of the selected/unselected state of the toggle. The application can also manage and unmanage toggle children of RowColumn at any time regardless of state. Therefore, the application can sometimes create a RowColumn that has **XmNradioAlwaysOne** set to True and none of the toggle children selected. The result is undefined if the value of this resource is True and the application sets more than one ToggleButton at a time.

XmNradioBehavior

Specifies a Boolean value that when True, indicates that the RowColumn widget should enforce a RadioBox-type behavior on all of its children that are ToggleButtons or ToggleButtonGadgets.

When the value of this resource is True, **XmNindicatorType** defaults to **XmONE_OF_MANY** for ToggleButton and ToggleButtonGadget children.

RadioBox behavior dictates that when one toggle is selected and the user selects another toggle, the first toggle is unselected automatically. The RowColumn usually does not enforce this behavior if the application, rather than the user, changes the state of a toggle. The RowColumn does enforce this behavior if a toggle child is selected with **XmToggleButtonSetState** or **XmToggleButtonGadgetSetState** with a *notify* argument of True.

When **XmCreateRadioBox** is called, the default value of **XmNradioBehavior** is True. Otherwise, the default value is False.

XmNresizeHeight
Requests a new height if necessary, when set to True. When this resource is set to False, the widget does not request a new height regardless of any changes to the widget or its children.

XmNresizeWidth
Requests a new width if necessary, when set to True. When set to False, the widget does not request a new width regardless of any changes to the widget or its children.

XmNrowColumnType
Specifies the type of RowColumn widget to be created. It is a nonstandard resource that cannot be changed after it is set. If an application uses any of the convenience routines, except **XmCreateRowColumn**, this resource is automatically forced to the appropriate value by the convenience routine. If an application uses the Xt Intrinsics API to create its RowColumn widgets, it must specify this resource itself. The set of possible settings for this resource are

- **XmWORK_AREA** (default)
- **XmMENU_BAR**
- **XmMENU_PULLDOWN**
- **XmMENU_POPUP**
- **XmMENU_OPTION**

This resource cannot be changed after the RowColumn widget is created. Any changes attempted through **XtSetValues** are ignored.

The value of this resource is used to determine the value of a number of other resources. The descriptions of RowColumn resources explain this when it is the case. The resource **XmNnavigationType**, inherited from **XmManager**, is changed to **XmNONE** if **XmNrowColumnType** is **XmMENU_OPTION**.

XmNspacing Specifies the horizontal and vertical spacing between items contained within the RowColumn widget. The default value is 3 pixels for **XmOPTION_MENU** and **XmWORK_AREA** and 0 (zero) for other RowColumn types.

XmNsubMenuId

Specifies the widget ID for the Pulldown MenuPane to be associated with an OptionMenu. This resource is useful only when **XmNrowColumnType** is set to **XmMENU_OPTION**. The default value is NULL.

XmNtearOffMenuActivateCallback

Specifies the callback list that notifies the application when the tear-off MenuPane is about to be activated. It precedes the tear-off's map callback.

Use this resource when your application has shared MenuPanes and when the torn-off menu can have two or more parents that can have opposing sensitivity states for the same menu item. This resource enables the application to track whether a menu item is sensitive or insensitive and to set the state to the original parent's menu item state when the torn-off menu is reposted. The application can use **XmGetPostedFromWidget** to determine from which parent the menu was torn. The callback reason is **XmCR_TEAR_OFF_ACTIVATE**. The default is NULL.

XmNtearOffMenuDeactivateCallback

Specifies the callback list that notifies the application when the tear-off MenuPane is about to be deactivated. It follows the tear-off's unmap callback.

Use this resource when your application has shared MenuPanes and when the torn-off menu can have two or more parents that can have opposing sensitivity states for the same menu item. This resource enables the application to track whether a menu item is sensitive or insensitive and to set the state to the original parent's menu item state when the torn-off menu is reposted. The application can use **XmGetPostedFromWidget** to determine from which parent the menu was torn. The callback reason is **XmCR_TEAR_OFF_DEACTIVATE**. The default is NULL.

XmNtearOffModel

Indicates whether tear-off functionality is enabled or disabled when **XmNrowColumnType** is set to **XmMENU_PULLDOWN** or **XmMENU_POPUP**. The values are **XmTEAR_OFF_ENABLED** or **XmTEAR_OFF_DISABLED** (default value). This resource is invalid for type **XmMENU_OPTION**; however, it does affect any

pulldown submenus within an OptionMenu. The function **XmRepTypeInstallTearOffModelConverter** installs a resource converter for this resource.

XmNunmapCallback

Specifies a list of callbacks that is called after the window associated with the RowColumn widget has been unmapped. The callback reason is **XmCR_UNMAP**. The default value is NULL.

XmNwhichButton

This resource is obsolete; it has been replaced by **XmNmenuPost** and is present for compatibility with older releases of OSF/Motif.

XmRowColumn Constraint Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNpositionIndex XmCPositionIndex	XmLAST_POSITION short	CSG

XmNpositionIndex

Specifies the position of the widget in its parent's list of children (the value of the **XmNchildren** resource). The value is an integer that is no less than 0 (zero) and no greater than the number of children in the list at the time the value is specified. A value of 0 (zero) means that the child is placed at the beginning of the list. The value can also be specified as **XmLAST_POSITION** (the default), which means that the child is placed at the end of the list. Any other value is ignored. **XtGetValues** returns the position of the widget in its parent's child list at the time of the call to **XtGetValues**.

When a widget is inserted into its parent's child list, the positions of any existing children that are greater than or equal to the specified widget's **XmNpositionIndex** are increased by 1. The effect of a call to **XtSetValues** for **XmNpositionIndex** is to remove the specified widget from its parent's child list, decrease by 1 the positions of any existing children that are greater than the specified widget's former position in the list, and then insert the specified widget into its parent's child list as described in the preceding sentence.

Simple Menu Creation Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbuttonAccelerators XmCButtonAccelerators	NULL StringTable	C
XmNbuttonAcceleratorText XmCButtonAcceleratorText	NULL XmStringTable	C
XmNbuttonCount XmCButtonCount	0 int	C
XmNbuttonMnemonicCharSets XmCButtonMnemonicCharSets	NULL XmStringCharSetTable	C
XmNbuttonMnemonics XmCButtonMnemonics	NULL XmKeySymTable	C
XmNbuttons XmCButtons	NULL XmStringTable	C
XmNbuttonSet XmCButtonSet	-1 int	C
XmNbuttonType XmCButtonType	NULL XmButtonTypeTable	C
XmNoptionLabel XmCOptionLabel	NULL XmString	C
XmNoptionMnemonic XmCOptionMnemonic	NULL KeySym	C
XmNpostFromButton XmCPostFromButton	-1 int	C
XmNsimpleCallback XmCCallback	NULL XtCallbackProc	C

XmNbuttonAccelerators

This resource is for use with the simple menu creation routines. It specifies a list of accelerators for the buttons created. The list contains one element for each button, separator, and title created.

XmNbuttonAcceleratorText

This resource is for use with the simple menu creation routines. It specifies a list of compound strings to display for the accelerators for the buttons created. The list contains one element for each button, separator, and title created.

XmNbuttonCount
This resource is for use with the simple menu creation routines. It specifies the total number of menu buttons, separators, and titles to create. The value must not be negative.

XmNbuttonMnemonicCharSets
This resource is for use with the simple menu creation routines. It specifies a list of character sets with which button mnemonics are to be displayed. The list contains one element for each button, separator, and title created. The default is determined dynamically depending on the locale of the widget.

XmNbuttonMnemonics
This resource is for use with the simple menu creation routines. It specifies a list of mnemonics for the buttons created. The list contains one element for each button, separator, and title created.

XmNbuttons This resource is for use with the simple menu creation routines. It specifies a list of compound strings to use as labels for the buttons created. The list contains one element for each button, separator, and title created.

XmNbuttonSet
This resource is for use with the simple menu creation routines. It specifies which button of a RadioBox or OptionMenu Pulldown submenu is initially set. The value is an integer *n* indicating the *n*th ToggleButtonGadget specified for a RadioBox or the *n*th PushButtonGadget specified for an OptionMenu Pulldown submenu. The first button specified is number 0. The value must not be negative.

XmNbuttonType
This resource is for use with the simple menu creation routines. It specifies a list of button types associated with the buttons to be created. The list contains one element for each button, separator, and title created. If this resource is not specified, each button in a MenuBar is a CascadeButtonGadget, each button in a RadioBox or CheckBox is a ToggleButtonGadget, and each button in any other

type of RowColumn widget is a PushButtonGadget. Each button type is of type **XmButtonType**, an enumeration with the following possible values:

XmCASCADEBUTTON
Specifies a CascadeButtonGadget for a MenuBar, Popup MenuPane, or Pulldown MenuPane.

XmCHECKBUTTON
Specifies a ToggleButtonGadget for a CheckBox, Popup MenuPane, or Pulldown MenuPane.

XmDOUBLE_SEPARATOR
Specifies a SeparatorGadget for a Popup MenuPane, Pulldown MenuPane, or OptionMenu Pulldown submenu. The separator type is **XmDOUBLE_LINE**.

XmPUSHBUTTON
Specifies a PushButtonGadget for a Popup MenuPane, Pulldown MenuPane, or OptionMenu Pulldown submenu.

XmRADIOBUTTON
Specifies a ToggleButtonGadget for a RadioBox, Popup MenuPane, or Pulldown MenuPane.

XmSEPARATOR
Specifies a SeparatorGadget for a Popup MenuPane, Pulldown MenuPane, or OptionMenu Pulldown submenu.

XmTITLE Specifies a LabelGadget used as a title for a Popup MenuPane or Pulldown MenuPane.

XmNoptionLabel
This resource is for use with the simple menu creation routines. It specifies a compound string for the label string to be used on the left side of an OptionMenu.

XmNoptionMnemonic

This resource is for use with the simple menu creation routines. It specifies a keysym for a key that, when pressed by the user along with the **MAlt** modifier, posts the associated Pulldown MenuPane for an OptionMenu.

XmNpostFromButton

This resource is for use with the simple menu creation routines. For a Pulldown MenuPane, it specifies the button in the parent to which the submenu is attached. The menu is then posted from this button. The value is an integer *n* indicating the *n*th CascadeButton or CascadeButtonGadget specified for the parent of the Pulldown MenuPane. The first button specified is number 0. The value must not be negative.

XmNsimpleCallback

This resource is for use with the simple menu creation routines. It specifies a callback procedure to be called when a button is activated or when its value changes. This callback function is added to each button after creation. For a CascadeButtonGadget or a PushButtonGadget, the callback is added as the button's **XmNactivateCallback**, and it is called when the button is activated. For a ToggleButtonGadget, the callback is added as the button's **XmNvalueChangedCallback**, and it is called when the button's value changes. The button number is passed in the *client_data* field.

Inherited Resources

RowColumn inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	dynamic XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	dynamic Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	default procedure XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    Widget      widget;
    char        * data;
    char        * callbackstruct;
} XmRowColumnCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

The following fields apply only when the callback reason is **XmCR_ACTIVATE**; for all other callback reasons, these fields are set to NULL. The **XmCR_ACTIVATE** callback reason is generated only when the application has supplied an entry callback, which overrides any activation callbacks registered with the individual RowColumn items.

widget Is set to the widget ID of the RowColumn item that has been activated

data Contains the client-data value supplied by the application when the RowColumn item's activation callback was registered

callbackstruct
Points to the callback structure generated by the RowColumn item's activation callback

Translations

XmRowColumn translations depend on the value of the **XmNrowColumnType** resource.

If **XmNrowColumnType** is set to **XmWORK_AREA**, **XmRowColumn** inherits translations from **XmManager**.

If **XmNrowColumnType** is set to **XmMENU_OPTION**, **XmRowColumn** inherits traversal, KActivate, and KCancel translations from **XmManager** and has the following additional translations. These translations may not directly correspond to a translation table.

BSelect Press:	**MenuBtnDown()**
BSelect Release:	**MenuBtnUp()**
KSelect:	**ManagerGadgetSelect()**
KHelp:	**Help()**

The translations for **XmRowColumn** if **XmNrowColumnType** is set to **XmMENU_BAR** **XmMENU_PULLDOWN**, or **XmMENU_POPUP** are described in the following list. In a Popup menu system, **BMenu** also performs the **BSelect** actions. These translations may not directly correspond to a translation table.

BSelect Press:	**MenuBtnDown()**
BSelect Release:	**MenuBtnUp()**
KActivate:	**ManagerGadgetSelect()**
KSelect:	**ManagerGadgetSelect()**
MAny KCancel:	**MenuGadgetEscape()**
KHelp:	**Help()**
KLeft:	**MenuGadgetTraverseLeft()**
KRight:	**MenuGadgetTraverseRight()**
KUp:	**MenuGadgetTraverseUp()**
KDown:	**MenuGadgetTraverseDown()**

Action Routines

The **XmRowColumn** action routines are

Help():
Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

ManagerGadgetSelect():
When a gadget child of the menu has the focus, invokes the gadget child's behavior associated with **KSelect**. This generally has the effect of unposting the menu hierarchy and arming and activating the gadget, except that, for a CascadeButtonGadget with a submenu, it posts the submenu.

MenuBtnDown():
When a gadget child of the menu has focus, invokes the gadget child's behavior associated with **BSelect Press**. This generally has the effect of unposting any menus posted by the parent menu, enabling mouse traversal in the menu, and arming the gadget. For a CascadeButtonGadget with a submenu, it also posts the associated submenu.

MenuBtnUp():
When a gadget child of the menu has focus, invokes the gadget child's behavior associated with **BSelect Release**. This generally has the effect of unposting the menu hierarchy and activating the gadget, except that for a CascadeButtonGadget with a submenu, it posts the submenu and enables keyboard traversal in the menu.

MenuGadgetEscape():
In a top-level Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu.

In a Popup MenuPane, unposts the menu and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores keyboard focus to the widget from which the menu was posted. In a TearOff MenuPane that has no submenus posted, dismisses the menu; otherwise, if one or more submenus are posted, unposts the last menu pane.

MenuGadgetTraverseDown():

If the current menu item has a submenu and is in a MenuBar, then this action posts the submenu, disarms the current menu item, and arms the submenu's first traversable menu item.

If the current menu item is in a MenuPane, then this action disarms the current menu item and arms the item below it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's bottom edge, then this action wraps to the topmost menu item in the column to the right, if one exists. When the current menu item is at the bottom, rightmost corner of the MenuPane, then this action wraps to the tear-off control, if present, or to the top, leftmost menu item.

MenuGadgetTraverseLeft():

When the current menu item is in a MenuBar, this action disarms the current item and arms the MenuBar item to the left. This action wraps within the MenuBar.

In MenuPanes, if the current menu item is not at the left edge of a MenuPane, this action disarms the current item and arms the item to its left. If the current menu item is at the left edge of a submenu attached to a MenuBar item, then this action unposts the submenu and traverses to the MenuBar item to the left, wrapping if necessary. If that MenuBar item has a submenu, it posts the submenu and arms the first traversable item in the submenu. If the current menu item is at the left edge of a submenu not directly attached to a MenuBar item, then this action unposts the current submenu only.

In Popup or torn-off MenuPanes, when the current menu item is at the left edge, this action wraps within the MenuPane. If the current menu item is at the left edge of the MenuPane and not in the top row, this action wraps to the rightmost menu item in the row above. If the current menu item is in the upper, leftmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, rightmost menu item in the MenuPane.

MenuGadgetTraverseRight():

If the current menu item is in a MenuBar, then this action disarms the current item and arms the MenuBar item to the right. This action wraps within the MenuBar.

In MenuPanes, if the current menu item is a CascadeButton, then this action posts its associated submenu. If the current menu item is not a CascadeButton and is not at the right edge of a MenuPane, this action disarms the current item and arms the item to its right, wrapping if necessary. If the current menu item is not a

CascadeButton and is at the right edge of a submenu that is a descendent of a MenuBar, then this action unposts all submenus and traverses to the MenuBar item to the right. If that MenuBar item has a submenu, it posts the submenu and arms the submenu's first traversable item.

In Popup or torn-off menus, if the current menu item is not a CascadeButton and is at the right edge of a row (except the bottom row), this action wraps to the leftmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, rightmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, leftmost menu item of the MenuPane.

MenuGadgetTraverseUp():
When the current menu item is in a MenuPane, then this action disarms the current menu item and arms the item above it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's top edge, then this action wraps to the bottommost menu item in the column to the left, if one exists. When the current menu item is at the top, leftmost corner of the MenuPane, then this action wraps to the tear-off control, if present, or to the bottom, rightmost menu item.

Related Behavior

The following menu functions are available:

KMenuBar: In any non-popup descendant of a MenuBar's parent, excluding the MenuBar itself, this action enables keyboard traversal and moves keyboard focus to the first item in the MenuBar. In the MenuBar or any menu cascaded from it, this action unposts the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores focus to the widget that had the focus when the menu system was entered.

KMenu: Pops up the menu associated with the control that has the keyboard focus. Enables keyboard traversal in the menu. In the Popup menu system or any menu cascaded from it, this action unposts the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICIT**, restores focus to the widget that had the focus when the menu system was entered.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreateMenuBar(3X)**, **XmCreateOptionMenu(3X)**, **XmCreatePopupMenu(3X)**, **XmCreatePulldownMenu(3X)**, **XmCreateRadioBox(3X)**, **XmCreateRowColumn(3X)**, **XmCreateSimpleCheckBox(3X)**, **XmCreateSimpleMenuBar(3X)**, **XmCreateSimpleOptionMenu(3X)**, **XmCreateSimplePopupMenu(3X)**, **XmCreateSimplePulldownMenu(3X)**, **XmCreateSimpleRadioBox(3X)**, **XmCreateWorkArea(3X)**, **XmGetMenuCursor(3X)**, **XmGetPostedFromWidget(3X)**, **XmGetTearOffControl**, **XmLabel(3X)**, **XmManager(3X)**, **XmMenuPosition(3X)**, **XmOptionButtonGadget(3X)**, **XmOptionLabelGadget(3X)**, **XmRepTypeInstallTearOffModelConverter**, **XmSetMenuCursor(3X)**, **XmUpdateDisplay(3X)**, **XmVaCreateSimpleCheckBox(3X)**, **XmVaCreateSimpleMenuBar(3X)**, **XmVaCreateSimpleOptionMenu(3X)**, **XmVaCreateSimplePopupMenu(3X)**, **XmVaCreateSimplePulldownMenu(3X)**, and **XmVaCreateSimpleRadioBox(3X)**.

XmScale—The Scale widget class

Synopsis

#include <Xm/Scale.h>

Description

Scale is used by an application to indicate a value from within a range of values, and it allows the user to input or modify a value from the same range.

A Scale has an elongated rectangular region similar to a ScrollBar. A slider inside this region indicates the current value along the Scale. The user can also modify the Scale's value by moving the slider within the rectangular region of the Scale. A Scale can also include a label set located outside the Scale region. These can indicate the relative value at various positions along the scale.

A Scale can be either input/output or output only. An input/output Scale's value can be set by the application and also modified by the user with the slider. An output-only Scale is used strictly as an indicator of the current value of something and cannot be modified interactively by the user. The **Core** resource **XmNsensitive** specifies whether the user can interactively modify the Scale's value.

The user can specify resources in a resource file for the automatically created gadget that contains the title of the Scale widget. The name of the gadget is **Title**.

Classes

Scale inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is **xmScaleWidgetClass**.

The class name is **XmScale**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmScale Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdecimalPoints XmCDecimalPoints	0 short	CSG
XmNdragCallback XmCCallback	NULL XtCallbackList	C
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNmaximum XmCMaximum	100 int	CSG
XmNminimum XmCMinimum	0 int	CSG
XmNorientation XmCOrientation	XmVERTICAL unsigned char	CSG
XmNprocessingDirection XmCProcessingDirection	dynamic unsigned char	CSG
XmNscaleHeight XmCScaleHeight	0 Dimension	CSG
XmNscaleMultiple XmCScaleMultiple	dynamic int	CSG
XmNscaleWidth XmCScaleWidth	0 Dimension	CSG
XmNshowValue XmCShowValue	False Boolean	CSG

Name Class	Default Type	Access
XmNtitleString XmCTitleString	NULL XmString	CSG
XmNvalue XmCValue	dynamic int	CSG
XmNvalueChangedCallback XmCCallback	NULL XtCallbackList	C

XmNdecimalPoints
Specifies the number of decimal points to shift the slider value when displaying it. For example, a slider value of 2,350 and an **XmdecimalPoints** value of 2 results in a display value of 23.50. The value must not be negative.

XmNdragCallback
Specifies the list of callbacks that is called when the slider position changes as the slider is being dragged. The reason sent by the callback is **XmCR_DRAG.**

XmNfontList Specifies the font list to use for the title text string specified by **XmNtitleString**, and the label displayed when **XmNshowValue** is True. If this value is NULL at initialization, the parent hierarchy is searched for an ancestor that is a subclass of the BulletinBoard, VendorShell, or MenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on the creation and structure of a font list.

XmNhighlightOnEnter
Specifies whether the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is **XmEXPLICIT**, this resource is ignored, and the widget is highlighted when it has the focus. If the shell's focus policy is **XmPOINTER** and if this resource is True, the highlighting rectangle is drawn when the the cursor moves into the widget. If the shell's focus policy is **XmPOINTER** and if this resource is False, the highlighting rectangle is not drawn when the the cursor moves into the widget. The default is False.

XmNhighlightThickness
Specifies the size of the slider's border drawing rectangle used for enter window and traversal highlight drawing.

XmNmaximum
Specifies the slider's maximum value. **XmNmaximum** must be greater than **XmNminimum**.

XmNminimum
Specifies the slider's minimum value. **XmNmaximum** must be greater than **XmNminimum**.

XmNorientation
Displays Scale vertically or horizontally. This resource can have values of **XmVERTICAL and XmHORIZONTAL**.

XmNprocessingDirection
Specifies whether the value for **XmNmaximum** is on the right or left side of **XmNminimum** for horizontal Scales or above or below **XmNminimum** for vertical Scales. This resource can have values of **XmMAX_ON_TOP, XmMAX_ON_BOTTOM, XmMAX_ON_LEFT**, and **XmMAX_ON_RIGHT**. If the XmScale is oriented vertically, the default value is **XmMAX_ON_TOP**. If the XmScale is oriented horizontally, the default value may depend on the value of the **XmNstringDirection** resource.

XmNscaleHeight
Specifies the height of the slider area. The value should be in the specified unit type (the default is pixels). If no value is specified a default height is computed.

XmNscaleMultiple
Specifies the amount to move the slider when the user takes an action that moves the slider by a multiple increment. The default is (**XmNmaximum** - **XmNminimum**) divided by 10, with a minimum of 1.

XmNscaleWidth
Specifies the width of the slider area. The value should be in the specified unit type (the default is pixels). If no value is specified a default width is computed.

XmNshowValue
Specifies whether a label for the current slider value should be displayed next to the slider. If the value is True, the current slider value is displayed.

XmNtitleString
Specifies the title text string to appear in the Scale widget window.

XmNvalue Specifies the slider's current position along the scale, between **XmNminimum** and **XmNmaximum**. The value is constrained to be within these inclusive bounds. The initial value of this resource is the larger of 0 and **XmNminimum**.

XmNvalueChangedCallback
Specifies the list of callbacks that is called when the value of the slider has changed. The reason sent by the callback is **XmCR_VALUE_CHANGED**.

Inherited Resources

Scale inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    int         value;
} XmScaleCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

value Is the new slider value

Behavior

XmScale has the following behavior:

BSelect Press or **BTransfer Press**:

In the region between an end of the Scale and the slider:
Moves the slider by one multiple increment in the direction of the end of the Scale and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement toward the right or bottom increments the Scale value, and movement toward the left or top decrements the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement toward the right or bottom decrements the Scale value, and movement toward the left or top increments the Scale value. If the button is held down longer than a delay period, the slider is moved again by the same increment and the same callbacks are called.

In slider:
Activates the interactive dragging of the slider.

BSelect Motion or **BTransfer Motion**:
If the button press occurs within the slider, the subsequent motion events move the slider to the position of the pointer and call the callbacks for **XmNdragCallback**.

BSelect Release or **BTransfer Release**:
If the button press occurs within the slider and the slider position is changed, the callbacks for **XmNvalueChangedCallback** are called.

MCtrl BSelect Press:
In the region between an end of the Scale and the slider:
Moves the slider to that end of the Scale and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement toward the right or bottom increments the Scale value, and movement toward the left or top decrements the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement toward the right or bottom decrements the Scale value, and movement toward the left or top increments the Scale value.

KUp: For vertical Scales, moves the slider up one increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_TOP**, movement toward the top increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_BOTTOM**, movement toward the top decrements the Scale value.

KDown: For vertical Scales, moves the slider down one increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_BOTTOM**, movement toward the bottom increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_TOP**, movement toward the bottom decrements the Scale value.

KLeft: For horizontal Scales, moves the slider one increment to the left and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_LEFT**, movement toward the left increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT**, movement toward the left decrements the Scale value.

KRight: For horizontal Scales, moves the slider one increment to the right and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT**, movement toward the right increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_LEFT**, movement toward the right decrements the Scale value.

MCtrl KUp or **KPageUp**:
For vertical Scales, moves the slider up one multiple increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_TOP**, movement toward the top increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_BOTTOM**, movement toward the top decrements the Scale value.

MCtrl KDown or **KPageDown**:
For vertical Scales, moves the slider down one multiple increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_BOTTOM**, movement toward the bottom increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_TOP**, movement toward the bottom decrements the Scale value.

MCtrl KLeft or **KPageLeft**:
For horizontal Scales, moves the slider one multiple increment to the left and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_LEFT**, movement toward the left increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT**, movement toward the left decrements the Scale value.

MCtrl KRight or **KPageRight**:
For horizontal Scales, moves the slider one multiple increment to the right and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT**, movement toward the right increments the Scale value. If **XmNprocessingDirection** is **XmMAX_ON_LEFT**, movement toward the right decrements the Scale value.

KBeginLine or **KBeginData**:
Moves the slider to the minimum value and calls the **XmNvalueChangedCallback** callbacks.

KEndLine or **KEndData**:
Moves the slider to the maximum value and calls the **XmNvalueChangedCallback** callbacks.

KNextField: Traverses to the first item in the next tab group. If the current tab group is the last entry in the tab group list, it wraps to the beginning of the tab group list.

KPrevField: Traverses to the first item in the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

KHelp: Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreateScale(3X)**, **XmManager(3X)**, **XmScaleGetValue(3X)**, and **XmScaleSetValue(3X)**.

XmScaleGetValue—A Scale function that returns the current slider position

Synopsis

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

void XmScaleGetValue (widget, value_return)
        Widget          widget;
        int             * value_return;
```

Description

XmScaleGetValue returns the current slider position value displayed in the scale.

widget Specifies the Scale widget ID

value_return Returns the current slider position value

For a complete definition of Scale and its associated resources, see **XmScale(3X)**.

Related Information

XmScale(3X).

XmScaleSetValue—A Scale function that sets a slider value

Synopsis

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

void XmScaleSetValue (widget, value)
        Widget      widget;
        int         value;
```

Description

XmScaleSetValue sets the slider *value* within the Scale widget.

widget Specifies the Scale widget ID.

value Specifies the slider position along the scale. This sets the **XmNvalue** resource.

For a complete definition of Scale and its associated resources, see **XmScale(3X)**.

Related Information

XmScale(3X).

XmScreen—The Screen widget class

Synopsis

#include <Xm/Screen.h>

Description

The XmScreen object is used by Motif widgets to store information that is specific to a screen. It also allows the toolkit to store certain information on widget hierarchies that would otherwise be unavailable. Each client has one XmScreen object for each screen that it accesses.

An XmScreen object is automatically created when the application creates the first shell on a screen (usually accomplished by a call to **XtAppInitialize** or **XtAppCreateShell**). It is not necessary to create an XmScreen object by any other means. An application can use the function **XmGetXmScreen** to obtain the widget ID of the XmScreen object for a given screen.

An application cannot supply initial values for XmScreen resources as arguments to a call to any function that creates widgets. The application or user can supply initial values in a resource file. After creating the first shell on the screen, the application can use **XmGetXmScreen** to obtain the widget ID of the XmScreen object and then call **XtSetValues** to set the XmScreen resources.

Classes

Screen inherits behavior and resources from **Core**.

The class pointer is **xmScreenClass**.

The class name is **XmScreen**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in an **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in an **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmScreen Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdarkThreshold XmCDarkThreshold	dynamic int	C
XmNdefaultCopyCursorIcon XmCDefaultCopyCursorIcon	NULL Widget	CSG
XmNdefaultInvalidCursorIcon XmCDefaultInvalidCursorIcon	NULL Widget	CSG
XmNdefaultLinkCursorIcon XmCDefaultLinkCursorIcon	NULL Widget	CSG
XmNdefaultMoveCursorIcon XmCDefaultMoveCursorIcon	NULL Widget	CSG
XmNdefaultNoneCursorIcon XmCDefaultNoneCursorIcon	NULL Widget	CSG
XmNdefaultSourceCursorIcon XmCDefaultSourceCursorIcon	NULL Widget	CSG
XmNdefaultValidCursorIcon XmCDefaultValidCursorIcon	NULL Widget	CSG
XmNfont XmCFont	NULL XFontStruct *	CSG
XmNforegroundThreshold XmCForegroundThreshold	dynamic int	C
XmNhorizontalFontUnit XmCHorizontalFontUnit	dynamic int	CSG
XmNlightThreshold XmCLightThreshold	dynamic int	C
XmNmenuCursor XmCCursor	arrow String	C
XmNmoveOpaque XmCMoveOpaque	False Boolean	CSG

Name Class	Default Type	Access
XmNunpostBehavior XmCUnpostBehavior	XmUNPOST_AND_REPLAY unsigned char	CSG
XmNverticalFontUnit XmCVerticalFontUnit	dynamic int	CSG

XmNdarkThreshold

An integer between 0 (zero) and 100, inclusive, that specifies a level of perceived brightness for a color. If the perceived brightness of the background color is below this level, Motif treats the background as "dark" when computing default shadow and select colors. If this resource is specified for a particular screen, it applies to widgets created on that screen; otherwise it applies to widgets created on all screens. The default value is implementation specific.

XmNdefaultCopyCursorIcon

Specifies the DragIcon used during a drag operation when the operation is a copy and no other pixmap is specified by the application. If this resource is NULL, a system default icon is used.

XmNdefaultInvalidCursorIcon

Specifies the DragIcon used to indicate that the cursor is over an invalid drop site during a drag operation when no other pixmap symbol is specified by the application. If this resource is NULL, a system default icon is used.

XmNdefaultLinkCursorIcon

Specifies the DragIcon used during a drag operation when the operation is a link and no other pixmap is specified by the application. If this resource is NULL, a system default icon is used.

XmNdefaultMoveCursorIcon

Specifies the DragIcon used during a drag operation when the operation is a move and no other pixmap is specified by the application. If this resource is NULL, a system default icon is used.

XmNdefaultNoneCursorIcon

Specifies the DragIcon used to indicate that the cursor is not over a drop site during a drag operation when no other pixmap is specified by the application. If this resource is NULL, a system default icon is used.

XmNdefaultSourceCursorIcon
Specifies the depth-1 pixmap used as a cursor when an **XmNsourceCursorIcon** is not provided by the DragContext, or it is not usable. If this resource is NULL, a system default icon is used.

XmNdefaultValidCursorIcon
Specifies the DragIcon used to indicate that the cursor is over a valid drop site during a drag operation when no other pixmap is specified by the application. If this resource is NULL, a system default icon is used.

XmNfont Specifies a font for use in computing values for **XmNhorizontalFontUnit** and **XmNverticalFontUnit**. When an application is initialized, this resource can be supplied in a resource file or through the standard command line options **-fn**, **-font**, and **-xrm**.

XmNforegroundThreshold
An integer between 0 (zero) and 100, inclusive, that specifies a level of perceived brightness for a color. If the perceived brightness of the background color is equal to or below this level, Motif treats the background as "dark" when computing the default foreground and highlight colors. If the perceived brightness of the background color is above this level, Motif treats the background as "light" when computing the default foreground and highlight colors. When the background is "dark," the default foreground and highlight is white; when the background is "light," the default foreground and highlight is black. If this resource is specified for a particular screen, it applies to widgets created on that screen; otherwise, it applies to widgets created on all screens. The default value is implementation specific.

XmNhorizontalFontUnit
Specifies the horizontal component of the font units used by **XmConvertUnits**, and is used to interpret the values of geometry resources when the **XmNshellUnitType** resource of VendorShell or the **XmNunitType** resource of Gadget, Manager, or Primitive has the value **Xm100TH_FONT_UNITS**. If no initial value is supplied for this resource, the default is computed from the font specified in **XmNfont**. If no initial value is supplied for this resource or for **XmNfont**, the default is 10.

If a call to **XtSetValues** specifies a value for **XmNhorizontalFontUnit**, this resource is set to that value. If a call to **XtSetValues** specifies a value for **XmNfont** but not for **XmNhorizontalFontUnit**, this resource is set to a value computed from the new **XmNfont**.

A horizontal font unit is derived from a font as follows:

- If the font has an **AVERAGE_WIDTH** property, the horizontal font unit is the **AVERAGE_WIDTH** property divided by 10.
- If the font has no **AVERAGE_WIDTH** property but has a **QUAD_WIDTH** property, the horizontal font unit is the **QUAD_WIDTH** property.
- If the font has no **AVERAGE_WIDTH** or **QUAD_WIDTH** property, the horizontal font unit is the sum of the font structure's *min_bounds.width* and *max_bounds.width* divided by 2.3.

XmNlightThreshold

An integer between 0 (zero) and 100, inclusive, that specifies a level of perceived brightness for a color. If the perceived brightness of the background color is above this level, Motif treats the background as "light" when computing default shadow and select colors. If this resource is specified for a particular screen, it applies to widgets created on that screen; otherwise, it applies to widgets created on all screens. The default value is implementation specific.

XmNmenuCursor

Sets a variable that controls the cursor used whenever this application posts a menu. This resource can be specified only once at application startup time, either by placing it within a defaults file or by using the **-xrm** command line argument. For example:

```
myProg -xrm "*menuCursor: arrow"
```

The menu cursor can also be selected in the program through the function **XmSetMenuCursor**. The following list shows acceptable cursor names. If the application does not specify a cursor or if an invalid name is supplied, the default cursor (an arrow pointing up and to the right) is used.

X_cursor
arrow
based_arrow_down
based_arrow_up
boat
bogosity
bottom_left_corner
bottom_right_corner
bottom_side
bottom_tee
box_spiral
center_ptr
circle
clock
coffee_mug
cross
cross_reverse
crosshair
diamond_cross
dot
dotbox
double_arrow
draft_large
draft_small
draped_box
exchange
leftbutton
ll_angle
lr_angle
man
middlebutton
mouse
pencil
pirate
plus
question_arrow
right_ptr
right_side
right_tee
rightbutton
rtl_logo
sailboat
sb_down_arrow
sb_h_double_arrow
sb_left_arrow
sb_right_arrow
sb_up_arrow
sb_v_double_arrow
shuttle
sizing
spider
spraycan

fleur	**star**
gobbler	**target**
gumby	**tcross**
hand1	**top_left_arrow**
hand2	**top_left_corner**
heart	**top_right_corner**
icon	**top_side**
iron_cross	**left_ptr**
left_side	**top_tee**
left_tee	**trek**
ul_angle	**umbrella**
ur_angle	**watch**
xterm	

XmNmoveOpaque

Specifies whether an interactive operation that moves a window, such as tearing off and dragging a tear-off menu or moving a window in MWM, displays an outline of the window or a representation of the window itself during the move. If the value is True, the operation displays a representation of the window during the move. If the value is False, the operation displays an outline of the window.

XmNunpostBehavior

Specifies the behavior of an active menu posted in traversal mode when a subsequent menu button selection is made outside the posted menu. When the value is **XmUNPOST_AND_REPLAY**, the resource unposts the menu hierarchy and causes the server to replay the event to the window in which the pointer is located. When the value is **XmUNPOST**, the resource unposts the hierarchy without replaying the event.

XmNverticalFontUnit

Specifies the vertical component of the font units used by **XmConvertUnits** and used to interpret the values of geometry resources when the **XmNshellUnitType** resource of VendorShell or the **XmNunitType** resource of Gadget, Manager, or Primitive has

the value **Xm100TH_FONT_UNITS**. If no initial value is supplied for this resource, the default is computed from the font specified in **XmNfont**. If no initial value is supplied for this resource or for **XmNfont**, the default is 10.

If a call to **XtSetValues** specifies a value for **XmNverticalFontUnit**, this resource is set to that value. If a call to **XtSetValues** specifies a value for **XmNfont** but not for **XmNverticalFontUnit**, this resource is set to a value computed from the new **XmNfont**.

A vertical font unit is derived from a font as follows:

- If the font has a **PIXEL_SIZE** property, the vertical font unit is the **PIXEL_SIZE** property divided by 1.8.
- If the font has no **PIXEL_SIZE** property but has **POINT_SIZE** and **RESOLUTION_Y** properties, the vertical font unit is the product of the **POINT_SIZE** and **RESOLUTION_Y** properties divided by 1400.
- If the font has no **PIXEL_SIZE**, **POINT_SIZE**, or **RESOLUTION_Y** properties, the vertical font unit is the sum of the font structure's *max_bounds.ascent* and *max_bounds.descent* divided by 2.2.

Inherited Resources

All of the superclass resources inherited by **XmScreen** are designated N/A (not applicable).

Related Information

Core(3X), **XmDisplay(3X)**, and **XmGetXmScreen(3X)**.

XmScrollBar—The ScrollBar widget class

Synopsis

#include <Xm/ScrollBar.h>

Description

The ScrollBar widget allows the user to view data that is too large to be displayed all at once. ScrollBars are usually located inside a ScrolledWindow and adjacent to the widget that contains the data to be viewed. When the user interacts with the ScrollBar, the data within the other widget scrolls.

A ScrollBar consists of two arrows placed at each end of a rectangle. The rectangle is called the scroll region. A smaller rectangle, called the slider, is placed within the scroll region. The data is scrolled by clicking either arrow, selecting on the scroll region, or dragging the slider. When an arrow is selected, the slider within the scroll region is moved in the direction of the arrow by an amount supplied by the application. If the mouse button is held down, the slider continues to move at a constant rate.

The ratio of the slider size to the scroll region size typically corresponds to the relationship between the size of the visible data and the total size of the data. For example, if 10 percent of the data is visible, the slider typically occupies 10 percent of the scroll region. This provides the user with a visual clue to the size of the invisible data.

Classes

ScrollBar inherits behavior and resources from the **Core** and **XmPrimitive** classes.

The class pointer is **xmScrollBarWidgetClass**.

The class name is **XmScrollBar**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmScrollBar Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdecrementCallback XmCCallback	NULL XtCallbackList	C
XmNdragCallback XmCCallback	NULL XtCallbackList	C
XmNincrement XmCIncrement	1 int	CSG
XmNincrementCallback XmCCallback	NULL XtCallbackList	C
XmNinitialDelay XmCInitialDelay	250 ms int	CSG
XmNmaximum XmCMaximum	dynamic int	CSG
XmNminimum XmCMinimum	0 int	CSG
XmNorientation XmCOrientation	XmVERTICAL unsigned char	CSG
XmNpageDecrementCallback XmCCallback	NULL XtCallbackList	C
XmNpageIncrement XmCPageIncrement	10 int	CSG
XmNpageIncrementCallback XmCCallback	NULL XtCallbackList	C
XmNprocessingDirection XmCProcessingDirection	dynamic unsigned char	CSG
XmNrepeatDelay XmCRepeatDelay	50 ms int	CSG
XmNshowArrows XmCShowArrows	True Boolean	CSG
XmNsliderSize XmCSliderSize	dynamic int	CSG

Name Class	Default Type	Access
XmNtoBottomCallback XmCCallback	NULL XtCallbackList	C
XmNtoTopCallback XmCCallback	NULL XtCallbackList	C
XmNtroughColor XmCTroughColor	dynamic Pixel	CSG
XmNvalue XmCValue	dynamic int	CSG
XmNvalueChangedCallback XmCCallback	NULL XtCallbackList	C

XmNdecrementCallback

Specifies the list of callbacks that is called when the user takes an action that moves the ScrollBar by one increment and the value decreases. The reason passed to the callback is **XmCR_DECREMENT**.

XmNdragCallback

Specifies the list of callbacks that is called on each incremental change of position when the slider is being dragged. The reason sent by the callback is **XmCR_DRAG**.

XmNincrement

Specifies the amount by which the value increases or decreases when the user takes an action that moves the slider by one increment. The actual change in value is the lesser of **XmNincrement** and (previous **XmNvalue** - **XmNminimum**) when the slider moves to the end of the ScrollBar with the minimum value, and the lesser of**XmNincrement** and (**XmNmaximum**-**XmNsliderSize** - previous **XmNvalue**) when the slider moves to the end of the ScrollBar with the maximum value. The value of this resource must be greater than 0 (zero).

XmNincrementCallback

Specifies the list of callbacks that is called when the user takes an action that moves the ScrollBar by one increment and the value increases. The reason passed to the callback is **XmCR_INCREMENT**.

XmNinitialDelay
Specifies the amount of time in milliseconds to wait before starting continuous slider movement while a button is pressed in an arrow or the scroll region. The value of this resource must be greater than 0 (zero).

XmNmaximum
Specifies the slider's maximum value. ScrollBars contained within ScrolledWindows have a maximum equal to the size of ScrollBar (that is, the height if it is vertical, or the width if it is horizontal). **XmNmaximum** must be greater than **XmNminimum**.

XmNminimum
Specifies the slider's minimum value. **XmNmaximum** must be greater than **XmNminimum**.

XmNorientation
Specifies whether the ScrollBar is displayed vertically or horizontally. This resource can have values of **XmVERTICAL** and **XmHORIZONTAL**.

XmNpageDecrementCallback
Specifies the list of callbacks that is called when the user takes an action that moves the ScrollBar by one page increment and the value decreases. The reason passed to the callback is **XmCR_PAGE_DECREMENT**.

XmNpageIncrement
Specifies the amount by which the value increases or decreases when the user takes an action that moves the slider by one page increment. The actual change in value is the lesser of **XmNpageIncrement** and (previous **XmNvalue** - **XmNminimum**) when the slider moves to the end of the ScrollBar with the minimum value, and the lesser of **XmNpageIncrement** and (**XmNmaximum**-**XmNsliderSize** - previous **XmNvalue**) when the slider moves to the end of the ScrollBar with the maximum value. The value of this resource must be greater than 0 (zero).

XmNpageIncrementCallback
Specifies the list of callbacks that is called when the user takes an action that moves the ScrollBar by one page increment and the value increases. The reason passed to the callback is **XmCR_PAGE_INCREMENT**.

XmNprocessingDirection

Specifies whether the value for **XmNmaximum** should be on the right or left side of **XmNminimum** for horizontal ScrollBars or above or below **XmNminimum** for vertical ScrollBars. This resource can have values of **XmMAX_ON_TOP, XmMAX_ON_BOTTOM,XmMAX_ON_LEFT**, and **XmMAX_ON_RIGHT**. If the XmScrollBar is oriented vertically, the default value is **XmMAX_ON_BOTTOM**. If the XmScrollBar is oriented horizontally, the default value may depend on the value of the **XmNstringDirection** resource.

XmNrepeatDelay

Specifies the amount of time in milliseconds to wait between subsequent slider movements after the **XmNinitialDelay** has been processed. The value of this resource must be greater than 0 (zero).

XmNshowArrows

Specifies whether the arrows are displayed.

XmNsliderSize

Specifies the length of the slider between the values of 1 and (**XmNmaximum** - **XmNminimum**). The value is constrained to be within these inclusive bounds. The default value is (**XmNmaximum** - **XmNminimum**) divided by 10, with a minimum of 1.

XmNtoBottomCallback

Specifies the list of callbacks that is called when the user takes an action that moves the slider to the end of the ScrollBar with the maximum value. The reason passed to the callback is **XmCR_TO_BOTTOM**.

XmNtoTopCallback

Specifies the list of callbacks that is called when the user takes an action that moves the slider to the end of the ScrollBar with the minimum value. The reason passed to the callback is **XmCR_TO_TOP**.

XmNtroughColor

Specifies the color of the slider trough.

XmNvalue Specifies the slider's position, between **XmNminimum** and (**XmNmaximum** - **XmNsliderSize**). The value is constrained to be within these inclusive bounds. The initial value of this resource is the larger of 0 (zero) and **XmNminimum**.

XmNvalueChangedCallback
Specifies the list of callbacks that is called when the slider is released after being dragged. These callbacks are also called in place of **XmNincrementCallback**, **XmNdecrementCallback**, **XmNpageIncrementCallback**, **XmNpageDecrementCallback**, **XmNtoTopCallback**, or **XmNtoBottomCallback** when one of these callback lists would normally be called but the value of the corresponding resource is NULL. The reason passed to the callback is **XmCR_VALUE_CHANGED**.

Inherited Resources

ScrollBar inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	dynamic Dimension	CSG
XmNnavigationType XmCNavigationType	XmSTICKY_TAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	dynamic Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int         reason;
    XEvent      * event;
    int         value;
    int         pixel;
} XmScrollBarCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

value Contains the new slider location value.

pixel Is used only for **XmNtoTopCallback** and **XmNtoBottomCallback**. For horizontal ScrollBars, it contains the *x* coordinate of where the mouse button selection occurred. For vertical ScrollBars, it contains the *y* coordinate.

Translations

XmScrollBar includes translations from Primitive. The **XmScrollBar** translations are described in the following list. These translations may not directly correspond to a translation table.

BSelect Press: **Select()**

BSelect Release: **Release()**

BSelect Press Moved: **Moved()**

BTransfer Press:	**Select()**
BTransfer Release:	**Release()**
BTransfer Press Moved:	**Moved()**
MCtrl BSelect Press:	**TopOrBottom()**
MCtrl BSelect Release:	**Release()**
KUp:	**IncrementUpOrLeft(0)**
MCtrl KUp:	**PageUpOrLeft(0)**
KDown:	**IncrementDownOrRight(0)**
MCtrl KDown:	**PageDownOrRight(0)**
KLeft:	**IncrementUpOrLeft(1)**
MCtrl KLeft:	**PageUpOrLeft(1)**
KRight:	**IncrementDownOrRight(1)**
MCtrl KRight:	**PageDownOrRight(1)**
KPageUp:	**PageUpOrLeft(0)**
KPageDown:	**PageDownOrRight(0)**
KPageLeft:	**PageUpOrLeft(1)**
KPageRight:	**PageDownOrRight(1)**
KBeginLine:	**TopOrBottom()**
KEndLine:	**TopOrBottom()**
KBeginData:	**TopOrBottom()**
KEndData:	**TopOrBottom()**
KNextField:	**PrimitiveNextTabGroup()**
KPrevField:	**PrimitivePrevTabGroup()**
KActivate:	**PrimitiveParentActivate()**
KCancel:	**CancelDrag()**
KHelp:	**PrimitiveHelp()**

Action Routines

The ScrollBar action routines are

CancelDrag():

If the key press occurs during scrolling, cancels the scroll and returns the slider to its previous location in the scrollbar; otherwise, and if the parent is a manager, it passes the event to the parent.

IncrementDownOrRight(0|1):

With an argument of 0, moves the slider down by one increment. With an argument of 1, it moves the slider right by one increment. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement toward the right or bottom calls the callbacks for **XmNincrementCallback**. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement toward the right or bottom calls the callbacks for **XmNdecrementCallback**. The **XmNvalueChangedCallback** is called if the **XmNincrementCallback** or **XmNdecrementCallback** is NULL.

IncrementUpOrLeft(0|1):

With an argument of 0, moves the slider up by one increment. With an argument of 1, it moves the slider left by one increment. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement to the left or top calls the callbacks for **XmNdecrementCallback**. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement to the left or top calls the callbacks for **XmNincrementCallback**. The **XmNvalueChangedCallback** is called if the **XmNincrementCallback** or **XmNdecrementCallback** is NULL.

Moved(): If the button press occurs within the slider, the subsequent motion events move the slider to the position of the pointer and call the callbacks for **XmNdragCallback**.

PageDownOrRight(0|1):

With an argument of 0, moves the slider down by one page increment. With an argument of 1, moves the slider right by one page increment. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement toward the right or bottom calls the callbacks for **XmNpageIncrementCallback**. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement toward the right or bottom calls the **XmNpageDecrementCallback**

callbacks. The **XmNvalueChangedCallback** is called if the **XmNpageIncrementCallback** or **XmNpageDecrementCallback** is NULL.

PageUpOrLeft(0|1):
With an argument of 0, moves the slider up by one page increment. With an argument of 1, it fmoves the slider left by one page increment. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement to the left or top calls the callbacks for **XmNpageDecrementCallback**. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement to the left or top calls the **XmNpageIncrementCallback** callbacks. The **XmNvalueChangedCallback** is called if the **XmNpageIncrementCallback** or **XmNpageDecrementCallback** is NULL.

PrimitiveHelp():
Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

PrimitiveNextTabGroup():
Traverses to the first item in the next tab group. If the current tab group is the last entry in the tab group list, it wraps to the beginning of the tab group list.

PrimitiveParentActivate():
If the parent is a manager, passes the event to the parent.

PrimitivePrevTabGroup():
Traverses to the first item in the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

Release(): If the button press occurs within the slider and the slider position is changed, the callbacks for **XmNvalueChangedCallback** are called.

Select(): **In arrow**:
Moves the slider by one increment in the direction of the arrow. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement toward the right or bottom calls the callbacks for **XmNincrementCallback**, and movement to the left or top calls the callbacks for **XmNdecrementCallback**. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement toward the right or bottom calls the

callbacks for **XmNdecrementCallback**, and movement to the left or top calls the callbacks for **XmNincrementCallback**. The **XmNvalueChangedCallback** is called if the **XmNincrementCallback** or **XmNdecrementCallback** is NULL.

In scroll region between an arrow and the slider:
Moves the slider by one page increment in the direction of the arrow. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement toward the right or bottom calls the callbacks for **XmNpageIncrementCallback**, and movement to the left or top calls the callbacks for **XmNpageDecrementCallback**. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement toward the right or bottom calls the callbacks for **XmNpageDecrementCallback**, and movement to the left or top calls the callbacks for **XmNpageIncrementCallback**. The **XmNvalueChangedCallback** is called if the **XmNpageIncrementCallback** or **XmNpageDecrementCallback** is NULL.

In slider:
Activates the interactive dragging of the slider.

If the button is held down in either the arrows or the scroll region longer than the **XmNinitialDelay** resource, the slider is moved again by the same increment and the same callbacks are called. After the initial delay has been used, the time delay changes to the time defined by the resource **XmNrepeatDelay**.

TopOrBottom():
MCtrl BSelect Press in an arrow or in the scroll region between an arrow and the slider moves the slider as far as possible in the direction of the arrow. If **XmNprocessingDirection** is **XmMAX_ON_RIGHT** or **XmMAX_ON_BOTTOM**, movement toward the right or bottom calls the callbacks for **XmNtoBottomCallback**, and movement to the left or top calls the callbacks for **XmNtoTopCallback**. If **XmNprocessingDirection** is **XmMAX_ON_LEFT** or **XmMAX_ON_TOP**, movement toward the right or bottom calls the callbacks for **XmNtoTopCallback**, and movement to the left or top calls the callbacks for **XmNtoBottomCallback**. The **XmNvalueChangedCallback** is called if the **XmNtoTopCallback** or **XmNtoBottomCallback** is NULL. Pressing **KBeginLine** or **KBeginData** moves the slider to the minimum value and invokes the **XmNtoTopCallback**. Pressing **KEndLine** or **KEndData** moves the slider to the maximum value and invokes the **XmNtoBottomCallback**.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateScrollBar(3X)**, **XmPrimitive(3X)**, **XmScrollBarGetValues(3X)**, and **XmScrollBarSetValues(3X)**.

XmScrollBarGetValues—A ScrollBar function that returns the ScrollBar's increment values

Synopsis

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

void XmScrollBarGetValues (widget, value_return,
        slider_size_return, increment_return, page_increment_return)
    Widget      widget;
    int         * value_return;
    int         * slider_size_return;
    int         * increment_return;
    int         * page_increment_return;
```

Description

XmScrollBarGetValues returns the the ScrollBar's increment values. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main ScrollBar or set slider function attributes.

widget Specifies the ScrollBar widget ID.

value_return Returns the ScrollBar's slider position between the **XmNminimum** and **XmNmaximum** resources.

slider_size_return
Returns the size of the slider as a value between zero and the absolute value of **XmNmaximum** minus **XmNminimum**. The size of the slider varies, depending on how much of the slider scroll area it represents.

increment_return
Returns the amount of increment and decrement.

page_increment_return
Returns the amount of page increment and decrement.

For a complete definition of ScrollBar and its associated resources, see **XmScrollBar(3X)**.

Return Value

Returns the ScrollBar's increment values.

Related Information

XmScrollBar(3X).

XmScrollBarSetValues—A ScrollBar function that changes ScrollBar's increment values and the slider's size and position

Synopsis

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

void XmScrollBarSetValues (widget, value,
                           slider_size, increment, page_increment, notify)
    Widget      widget;
    int         value;
    int         slider_size;
    int         increment;
    int         page_increment;
    Boolean     notify;
```

Description

XmSetScrollBarValues changes the ScrollBar's increment values and the slider's size and position. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main ScrollBar or set slider function attributes.

widget Specifies the ScrollBar widget ID.

value Specifies the ScrollBar's slider position between **XmNminimum** and **XmNmaximum**. The resource name associated with this argument is **XmNvalue**.

slider_size Specifies the size of the slider as a value between 0 (zero) and the absolute value of **XmNmaximum** minus **XmNminimum**. The size of the slider varies, depending on how much of the slider scroll area it represents. This sets the **XmNsliderSize** resource associated with ScrollBar.

increment Specifies the amount of button increment and decrement. If this argument is not 0 (zero), the ScrollBar widget automatically adjusts the slider when an increment or decrement action occurs. This sets the **XmNincrement** resource associated with ScrollBar.

page_increment
Specifies the amount of page increment and decrement. If this argument is not 0 (zero), the ScrollBar widget automatically adjusts the slider when an increment or decrement action occurs. This sets the **XmNpageIncrement** resource associated with ScrollBar.

notify Specifies a Boolean value that, when True, indicates a change in the ScrollBar value and also specifies that the ScrollBar widget automatically activates the **XmNvalueChangedCallback** with the recent change. If it is set to False, it specifies any change that has occurred in the ScrollBar's value, but does not activate **XmNvalueChangedCallback**.

For a complete definition of ScrollBar and its associated resources, see **XmScrollBar(3X)**.

Related Information

XmScrollBar(3X).

XmScrollVisible—A ScrolledWindow function that makes an invisible descendant of a ScrolledWindow work area visible

Synopsis

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

void XmScrollVisible (scrollw_widget, widget, left_right_margin, top_bottom_margin)
        Widget      scrollw_widget;
        Widget      widget;
        Dimension   left_right_margin;
        Dimension   top_bottom_margin;
```

Description

XmScrollVisible makes an obscured or partially obscured widget or gadget descendant of a ScrolledWindow work area visible. The function repositions the work area and sets the specified margins between the widget and the nearest viewport boundary. The widget's location relative to the viewport determines whether one or both of the margins must be adjusted. This function requires that the **XmNscrollingPolicy** of the ScrolledWindow widget be set to **XmAUTOMATIC**.

scrollw_widget
: Specifies the ID of the ScrolledWindow widget whose work area window contains an obscured descendant.

widget
: Specifies the ID of the widget to be made visible.

left_right_margin
: Specifies the margin to establish between the left or right edge of the widget and the associated edge of the viewport. This margin is established only if the widget must be moved horizontally to make it visible.

top_bottom_margin
: Specifies the margin to establish between the top or bottom edge of the widget and the associated edge of the viewport. This margin is established only if the widget must be moved vertically to make it visible.

For a complete definition of ScrolledWindow and its associated resources, see **XmScrolledWindow(3X)**

Related Information

XmScrolledWindow(3X).

XmScrolledWindow—The ScrolledWindow widget class

Synopsis

#include <Xm/ScrolledW.h>

Description

The ScrolledWindow widget combines one or two ScrollBar widgets and a viewing area to implement a visible window onto some other (usually larger) data display. The visible part of the window can be scrolled through the larger display by the use of ScrollBars.

To use ScrolledWindow, an application first creates a ScrolledWindow widget, any needed ScrollBar widgets, and a widget capable of displaying any desired data as the work area of ScrolledWindow. ScrolledWindow positions the work area widget and displays the ScrollBars if so requested. When the user performs some action on the ScrollBar, the application is notified through the normal ScrollBar callback interface.

ScrolledWindow can be configured to operate automatically so that it performs all scrolling and display actions with no need for application program involvement. It can also be configured to provide a minimal support framework in which the application is responsible for processing all user input and making all visual changes to the displayed data in response to that input.

When ScrolledWindow is performing automatic scrolling it creates a clipping window and automatically creates the scroll bars. Conceptually, this window becomes the viewport through which the user examines the larger underlying data area. The application simply creates the desired data, then makes that data the work area of the ScrolledWindow. When the user moves the slider to change the displayed data, the workspace is moved under the viewing area so that a new portion of the data becomes visible.

Sometimes it is impractical for an application to create a large data space and simply display it through a small clipping window. For example, in a text editor, creating a single data area that consisted of a large file would involve an undesirable amount of overhead. The application needs to use a ScrolledWindow (a small viewport onto some larger data), but needs to be notified when the user scrolls the viewport so it can bring in more data from storage and update the display area. For these cases, the ScrolledWindow can be configured so that it provides only visual layout support. No clipping window is created, and the application must maintain the data displayed in the work area, as well as respond to user input on the ScrollBars.

The user can specify resources in a resource file for the automatically created widgets that contain the horizontal and vertical scrollbars of the ScrolledWindow widget. The names of these widgets are **HorScrollBar** and **VertScrollBar**, and remain consistent whether created by **XmCreateScrolledList**, **XmCreateScrolledText** or **XmCreateScrolledWindow**.

Classes

ScrolledWindow inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

The class pointer is **xmScrolledWindowWidgetClass**.

The class name is **XmScrolledWindow**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmScrolledWindow Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNclipWindow XmCClipWindow	dynamic Widget	G
XmNhorizontalScrollBar XmCHorizontalScrollBar	dynamic Widget	CSG
XmNscrollBarDisplayPolicy XmCScrollBarDisplayPolicy	dynamic unsigned char	CSG
XmNscrollBarPlacement XmCScrollBarPlacement	XmBOTTOM_RIGHT unsigned char	CSG
XmNscrolledWindowMarginHeight XmCScrolledWindowMarginHeight	0 Dimension	CSG
XmNscrolledWindowMarginWidth XmCScrolledWindowMarginWidth	0 Dimension	CSG
XmNscrollingPolicy XmCScrollingPolicy	XmAPPLICATION_DEFINED unsigned char	CG
XmNspacing XmCSpacing	4 Dimension	CSG
XmNtraverseObscuredCallback XmCCallback	NULL XtCallbackList	CSG
XmNverticalScrollBar XmCVerticalScrollBar	dynamic Widget	CSG
XmNvisualPolicy XmCVisualPolicy	dynamic unsigned char	G
XmNworkWindow XmCWorkWindow	NULL Widget	CSG

XmNclipWindow

Specifies the widget ID of the clipping area. This is automatically created by ScrolledWindow when the **XmNvisualPolicy** resource is set to **XmCONSTANT** and can only be read by the application. Any attempt to set this resource to a new value causes a warning message to be printed by the scrolled window. If the **XmNvisualPolicy** resource is set to **XmVARIABLE**, this resource is set to NULL, and no clipping window is created.

XmNhorizontalScrollBar

Specifies the widget ID of the horizontal ScrollBar. This is automatically created by ScrolledWindow when the **XmNscrollingPolicy** is initialized to **XmAUTOMATIC**; otherwise, the default is NULL.

XmNscrollBarDisplayPolicy

Controls the automatic placement of the ScrollBars. If it is set to **XmAS_NEEDED** and if **XmNscrollingPolicy** is set to **XmAUTOMATIC**, ScrollBars are displayed only if the workspace exceeds the clip area in one or both dimensions. A resource value of **XmSTATIC** causes the ScrolledWindow to display the ScrollBars whenever they are managed, regardless of the relationship between the clip window and the work area. This resource must be **XmSTATIC** when **XmNscrollingPolicy** is **XmAPPLICATION_DEFINED**. The default is **XmAS_NEEDED** when **XmNscrollingPolicy** is **XmAUTOMATIC**, and **XmSTATIC** otherwise.

XmNscrollBarPlacement

Specifies the positioning of the ScrollBars in relation to the work window. The values are

XmTOP_LEFT

The horizontal ScrollBar is placed above the work window; the vertical ScrollBar to is placed the left.

XmBOTTOM_LEFT

The horizontal ScrollBar is placed below the work window; the vertical ScrollBar to is placed the left.

XmTOP_RIGHT

The horizontal ScrollBar is placed above the work window; the vertical ScrollBar to is placed the right.

XmBOTTOM_RIGHT

The horizontal ScrollBar is placed below the work window; the vertical ScrollBar to is placed the right.

The default value may depend on the value of the **XmNstringDirection** resource.

XmNscrolledWindowMarginHeight

Specifies the margin height on the top and bottom of the ScrolledWindow.

XmNscrolledWindowMarginWidth
Specifies the margin width on the right and left sides of the ScrolledWindow.

XmNscrollingPolicy
Performs automatic scrolling of the work area with no application interaction. If the value of this resource is **XmAUTOMATIC**, ScrolledWindow automatically creates the ScrollBars; attaches callbacks to the ScrollBars; sets the visual policy to **XmCONSTANT**; and automatically moves the work area through the clip window in response to any user interaction with the ScrollBars. An application can also add its own callbacks to the ScrollBars. This allows the application to be notified of a scroll event without having to perform any layout procedures.

NOTE: Since the ScrolledWindow adds callbacks to the ScrollBars, an application should not perform an **XtRemoveAllCallbacks** on any of the ScrollBar widgets.

When **XmNscrollingPolicy** is set to **XmAPPLICATION_DEFINED**, the application is responsible for all aspects of scrolling. The ScrollBars must be created by the application, and it is responsible for performing any visual changes in the work area in response to user input.

This resource must be set to the desired policy at the time the ScrolledWindow is created. It cannot be changed through **SetValues**.

XmNspacing Specifies the distance that separates the ScrollBars from the work window.

XmNtraverseObscuredCallback
Specifies a list of callbacks that is called when traversing to a widget or gadget that is obscured due to its position in the work area relative to the location of the ScrolledWindow viewport. This resource is valid only when **XmNscrollingPolicy** is **XmAUTOMATIC**. If this resource is NULL, an obscured widget cannot be traversed to. The callback reason is **XmCR_OBSCURED_TRAVERSAL**.

XmNverticalScrollBar
Specifies the widget ID of the vertical ScrollBar. This is automatically created by ScrolledWindow when the **XmNscrollingPolicy** is initialized to **XmAUTOMATIC**; otherwise, the default is NULL.

XmNvisualPolicy

Enlarges the ScrolledWindow to match the size of the work area. It can also be used as a static viewport onto a larger data space. If the visual policy is **XmVARIABLE**, the ScrolledWindow forces the ScrollBar display policy to **XmSTATIC** and allows the work area to grow or shrink at any time and adjusts its layout to accommodate the new size. When the policy is **XmCONSTANT**, the work area grows or shrinks as requested, but a clipping window forces the size of the visible portion to remain constant. The only time the viewing area can grow is in response to a resize from the ScrolledWindow's parent. The default is **XmCONSTANT** when **XmNscrollingPolicy** is **XmAUTOMATIC**, and **XmVARIABLE** otherwise.

NOTE: This resource must be set to the desired policy at the time the ScrolledWindow is created. It cannot be changed through **SetValues**.

XmNworkWindow

Specifies the widget ID of the viewing area.

Inherited Resources

ScrolledWindow inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

The application must use the ScrollBar callbacks to be notified of user input.

ScrolledWindow defines a callback structure for use with **XmNtraverseObscuredCallback** callbacks. The **XmNtraverseObscuredCallback** resource provides a mechanism for traversal to obscured widgets (or gadgets) due to their position in the work area of a ScrolledWindow. The **XmNtraverseObscuredCallback** routine has responsibility for adjusting the position of the work area such that the specified traversal destination widget is positioned within the viewport of the ScrolledWindow. A NULL **XmNtraverseObscuredCallback** resource causes obscured widgets within the ScrolledWindow to be nontraversable.

Traversal to an obscured widget or gadget requires that the following conditions are met: the widget or gadget can be obscured only due to its position in the work area of a ScrolledWindow relative to the viewport; the viewport of the associated ScrolledWindow is fully visible, or can be made visible with the **XmNtraverseObscuredCallback** routines of its ancestors; and the **XmNtraverseObscuredCallback** resource must be non-NULL.

When ScrolledWindow widgets are nested, the **XmNtraverseObscuredCallback** routine for each ScrolledWindow that obscures the traversal destination is called in ascending order within the given hierarchy.

A pointer to the following structure is passed to callbacks for **XmNtraverseObscuredCallback**.

```
typedef struct
{
    int                 reason;
    XEvent              *event:
     Widget             traversal_destination;
     XmTraversalDirectiondirection;
} XmTraverseObscuredCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

traversal_destination
Specifies the widget or gadget to traverse to, which will be a descendant of the work window.

direction Specifies the direction of traversal. See the description of the *direction* parameter in the **XmProcessTraversal** reference page for an explanation of the valid values.

Translations

XmScrolledWindow includes the translations from **XmManager**.

Additional Behavior

This widget has the following additional behavior:

KPageUp: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window up the height of the viewport. The distance scrolled may be reduced to provide some overlap. The actual distance scrolled depends on the **XmNpageIncrement** resource of the vertical ScrollBar.

KPageDown: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window down the height of the viewport. The distance scrolled may be reduced to provide some overlap. The actual distance scrolled depends on the **XmNpageIncrement** resource of the vertical ScrollBar.

KPageLeft: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window left the width of the viewport. The distance scrolled may be reduced to provide some overlap. The actual distance scrolled depends on the **XmNpageIncrement** resource of the horizontal ScrollBar.

KPageRight: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window right the width of the viewport. The distance scrolled may be reduced to provide some overlap. The actual distance scrolled depends on the **XmNpageIncrement** resource of the horizontal ScrollBar.

KBeginLine: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window horizontally to the edge corresponding to the horizontal ScrollBar's minimum value.

KEndLine: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window horizontally to the edge corresponding to the horizontal ScrollBar's maximum value.

KBeginData: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window vertically to the edge corresponding to the vertical ScrollBar's minimum value.

KEndData: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window vertically to the edge corresponding to the vertical ScrollBar's maximum value.

Certain applications will want to replace the page bindings with ones that are specific to the content of the scrolled area.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmCreateScrolledWindow(3X)**, **XmManager(3X)**, **XmProcessTraversal(3X)**, **XmScrollBar(3X)**, **XmScrollVisible(3X)**, and **XmScrolledWindowSetAreas(3X)**.

XmScrolledWindowSetAreas—A ScrolledWindow function that adds or changes a window work region and a horizontal or vertical ScrollBar widget to the ScrolledWindow widget

Synopsis

#include <Xm/ScrolledW.h>

void XmScrolledWindowSetAreas (*widget, horizontal_scrollbar, vertical_scrollbar, work_region*)

```
Widget      widget;
Widget      horizontal_scrollbar;
Widget      vertical_scrollbar;
Widget      work_region;
```

Description

XmScrolledWindowSetAreas adds or changes a window work region and a horizontal or vertical ScrollBar widget to the ScrolledWindow widget for the application. Each widget is optional and may be passed as NULL.

widget Specifies the ScrolledWindow widget ID.

horizontal_scrollbar
Specifies the ScrollBar widget ID for the horizontal ScrollBar to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The resource name associated with this argument is **XmNhorizontalScrollBar**.

vertical_scrollbar
Specifies the ScrollBar widget ID for the vertical ScrollBar to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The resource name associated with this argument is **XmNverticalScrollBar**.

work_region Specifies the widget ID for the work window to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The attribute name associated with this argument is **XmNworkWindow**.

For a complete definition of ScrolledWindow and its associated resources, see **XmScrolledWindow(3X)**.

Related Information

XmScrolledWindow(3X).

XmSelectionBox—The SelectionBox widget class

Synopsis

#include <Xm/SelectioB.h>

Description

SelectionBox is a general dialog widget that allows the user to select one item from a list. By default, a SelectionBox includes the following:

- A scrolling list of alternatives
- An editable text field for the selected alternative
- Labels for the list and text field
- Three or four buttons

The default button labels are **OK**, **Cancel**, and **Help**. By default an **Apply** button is also created; if the parent of the SelectionBox is a DialogShell, it is managed; otherwise it is unmanaged. Additional children may be added to the SelectionBox after creation. The first child is used as a work area. The value of **XmNchildPlacement** determines if the work area is placed above or below the Text area, or above or below the List area. Additional children are laid out in the following manner:

MenuBar	The first menu bar child is placed at the top of the window.
Buttons	All **XmPushButton** widgets or gadgets, and their subclasses are placed after the **OK** button in the order of their creation.

The layout of additional children that are not in the above categories is undefined.

The user can select an item in two ways: by scrolling through the list and selecting the desired item or by entering the item name directly into the text edit area. Selecting an item from the list causes that item name to appear in the selection text edit area.

The user may select a new item as many times as desired. The item is not actually selected until the user presses the **OK** PushButton.

The default value for the **XmBulletinBoard** resource **XmNcancelButton** is the Cancel button, unless **XmNdialogType** is **XmDIALOG_COMMAND**, when the default is NULL. The default value for the **XmBulletinBoard** **XmNdefaultButton** resource is the OK button, unless **XmNdialogType** is **XmDIALOG_COMMAND**, when the default is NULL.

For SelectionBox and its subclasses, the default value for **XmNinitialFocus** is the text edit area.

The user can specify resources in a resource file for the automatically created widgets and gadgets of SelectionBox. The following list identifies the names of these widgets (or gadgets) and the associated SelectionBox areas:

List Items Label	**Items**
List Items	**ItemsList**
Selection Label	**Selection**
Selection Text	**Text**
Selection Separator	**Separator**

Classes

SelectionBox inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XmBulletinBoard**.

The class pointer is **xmSelectionBoxWidgetClass**.

The class name is **XmSelectionBox**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmSelectionBox Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNapplyCallback XmCCallback	NULL XtCallbackList	C
XmNapplyLabelString XmCApplyLabelString	dynamic XmString	CSG
XmNcancelCallback XmCCallback	NULL XtCallbackList	C
XmNcancelLabelString XmCCancelLabelString	dynamic XmString	CSG
XmNchildPlacement XmCChildPlacement	XmPLACE_ABOVE_SELECTION unsigned char	CSG
XmNdialogType XmCDialogType	dynamic unsigned char	CG
XmNhelpLabelString XmCHelpLabelString	dynamic XmString	CSG
XmNlistItemCount XmCItemCount	0 int	CSG
XmNlistItems XmCItems	NULL XmStringTable	CSG
XmNlistLabelString XmCListLabelString	dynamic XmString	CSG
XmNlistVisibleItemCount XmCVisibleItemCount	dynamic int	CSG
XmNminimizeButtons XmCMinimizeButtons	False Boolean	CSG
XmNmustMatch XmCMustMatch	False Boolean	CSG
XmNnoMatchCallback XmCCallback	NULL XtCallbackList	C
XmNokCallback XmCCallback	NULL XtCallbackList	C

Name Class	Default Type	Access
XmNokLabelString XmCOkLabelString	dynamic XmString	CSG
XmNselectionLabelString XmCSelectionLabelString	dynamic XmString	CSG
XmNtextAccelerators XmCTextAccelerators	default XtAccelerators	C
XmNtextColumns XmCColumns	dynamic short	CSG
XmNtextString XmCTextString	"" XmString	CSG

XmNapplyCallback
Specifies the list of callbacks called when the user activates the **Apply** button. The callback reason is **XmCR_APPLY**.

XmNapplyLabelString
Specifies the string label for the **Apply** button. The default for this resource depends on the locale. In the C locale the default is **Apply**.

XmNcancelCallback
Specifies the list of callbacks called when the user activates the **Cancel** button. The callback reason is **XmCR_CANCEL**.

XmNcancelLabelString
Specifies the string label for the **Cancel** button. The default for this resource depends on the locale. In the C locale the default is **Cancel**.

XmNchildPlacement
Specifies the placement of the work area child. The possible values are

XmPLACE_ABOVE_SELECTION
Places the work area child above the Text area

XmPLACE_BELOW_SELECTION
Places the work area child below the Text area

XmPLACE_TOP
Places the work area child above the List area, and below a MenuBar, if one is present

XmNdialogType

Determines the set of SelectionBox children widgets that are created and managed at initialization. The possible values are

XmDIALOG_PROMPT

All standard children except the list and list label are created, and all except the **Apply** button are managed.

XmDIALOG_COMMAND

Only the list, the selection label, and the text field are created and managed.

XmDIALOG_SELECTION

All standard children are created and managed.

XmDIALOG_FILE_SELECTION

All standard children are created and managed.

XmDIALOG_WORK_AREA

All standard children are created, and all except the **Apply** button are managed.

If the parent of the SelectionBox is a DialogShell, the default is **XmDIALOG_SELECTION**; otherwise, the default is **XmDIALOG_WORK_AREA**. **XmCreatePromptDialog** and **XmCreateSelectionDialog** set and append this resource to the creation *arglist* supplied by the application. This resource cannot be modified after creation.

XmNhelpLabelString

Specifies the string label for the **Help** button. The default for this resource depends on the locale. In the C locale the default is **Help**.

XmNlistItems

Specifies the items in the SelectionBox list. **XtGetValues** for this resource returns the list items themselves, not a copy of the list items. The application must not free the returned items.

XmNlistItemCount

Specifies the number of items in the SelectionBox list. The value must not be negative.

XmNlistLabelString
Specifies the string label to appear above the SelectionBox list containing the selection items. The default for this resource depends on the locale. In the C locale the default is **Items** unless **XmNdialogType** is **XmDIALOG_PROMPT**; in this case the default is NULL.

XmNlistVisibleItemCount
Specifies the number of items displayed in the SelectionBox list. The value must be greater than 0 (zero) unless **XmNdialogType** is **XmDIALOG_PROMPT**; in this case, the value is always 0. The default is dynamic based on the height of the list.

XmNminimizeButtons
Sets the buttons to the width of the widest button and height of the tallest button if False. If True, button width and height are not modified.

XmNmustMatch
Specifies whether the selection widget should check if the user's selection in the text edit field has an exact match in the SelectionBox list when the **OK** button is activated. If the selection does not have an exact match, and **XmNmustMatch** is True, the **XmNnoMatchCallback** callbacks are called. If the selection does have an exact match or if **XmNmustMatch** is False, **XmNokCallback** callbacks are called.

XmNnoMatchCallback
Specifies the list of callbacks called when the user makes a selection from the text edit field that does not have an exact match with any of the items in the list box. The callback reason is **XmCR_NO_MATCH**. Callbacks in this list are called only if **XmNmustMatch** is true.

XmNokCallback
Specifies the list of callbacks called when the user activates the **OK** button. The callback reason is **XmCR_OK**. If the selection text does not match a list item, and **XmNmustMatch** is True, the **XmNnoMatchCallback** callbacks are called instead.

XmNokLabelString
Specifies the string label for the **OK** button. The default for this resource depends on the locale. In the C locale the default is **OK**.

XmNselectionLabelString
Specifies the string label for the selection text edit field. The default for this resource depends on the locale. In the C locale the default is **Selection**.

XmNtextAccelerators
Specifies translations added to the Text widget child of the SelectionBox. The default includes bindings for the up and down keys for auto selection of list items. This resource is ignored if **XmNaccelerators** is initialized to a nondefault value.

XmNtextColumns
Specifies the number of columns in the Text widget. The value must be greater than 0 (zero).

XmNtextString
Specifies the text in the text edit selection field.

Inherited Resources

SelectionBox inherits behavior and resources from the superclasses in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmBulletinBoard Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNallowOverlap XmCAllowOverlap	True Boolean	CSG
XmNautoUnmanage XmCAutoUnmanage	True Boolean	CG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNcancelButton XmCWidget	dynamic Widget	SG
XmNdefaultButton XmCWidget	dynamic Widget	SG
XmNdefaultPosition XmCDefaultPosition	True Boolean	CSG
XmNdialogStyle XmCDialogStyle	dynamic unsigned char	CSG
XmNdialogTitle XmCDialogTitle	NULL XmString	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	10 Dimension	CSG
XmNmarginWidth XmCMarginWidth	10 Dimension	CSG
XmNnoResize XmCNoResize	False Boolean	CSG
XmNresizePolicy XmCResizePolicy	XmRESIZE_ANY unsigned char	CSG

Name Class	Default Type	Access
XmNshadowType XmCShadowType	XmSHADOW_OUT unsigned char	CSG
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNtextTranslations XmCTranslations	NULL XtTranslations	C
XmNunmapCallback XmCCallback	NULL XtCallbackList	C

XmManager Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	dynamic Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Composite Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	N/A
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
    XmString        value;
    int             length;
} XmSelectionBoxCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

value Indicates the **XmString** value selected by the user from the SelectionBox list or entered into the SelectionBox text field

length Indicates the size in bytes of the **XmString** value

Translations

XmSelectionBox inherits translations from **XmBulletinBoard**.

Accelerators

The **XmNtextAccelerators** are added to the Text descendant of **XmSelectionBox**. The default accelerators are described in the following list. These accelerators may not directly correspond to a translation table.

KUp:	**SelectionBoxUpOrDown(0)**
KDown:	**SelectionBoxUpOrDown(1)**
KBeginData:	**SelectionBoxUpOrDown(2)**
KEndData:	**SelectionBoxUpOrDown(3)**
KRestore:	**SelectionBoxRestore()**

Action Routines

The **XmSelectionBox** action routines are

SelectionBoxUpOrDown(0|1|2|3):

When called with an argument of 0 (zero), selects the previous item in the list and replaces the text with that item.

When called with an argument of 1, selects the next item in the list and replaces the text with that item.

When called with an argument of 2, selects the first item in the list and replaces the text with that item.

When called with an argument of 3, selects the last item in the list and replaces the text with that item.

SelectionBoxRestore():

Replaces the text value with the list selection. If no item in the list is selected, clears the text.

Additional Behavior

The SelectionBox widget has the following additional behavior:

MAny KCancel:

Calls the activate callbacks for the cancel button if it is sensitive. If no cancel button exists and the parent of the SelectionBox is a manager, passes the event to the parent.

KActivate: Calls the activate callbacks for the button with the keyboard focus. If no button has the keyboard focus, calls the activate callbacks for the default button if it is sensitive. In a List widget or single-line Text widget, the List or Text action associated with **KActivate** is called before the SelectionBox actions associated with **KActivate**.

In a multiline Text widget, any **KActivate** event except **KEnter** calls the Text action associated with **KActivate**, then the SelectionBox actions associated with **KActivate**. If no button has the focus, no default button exists, and the parent of the SelectionBox is a manager, passes the event to the parent.

<OK Button Activated>:
If **XmNmustMatch** is True and the text does not match an item in the file list, calls the **XmNnoMatchCallback** callbacks with reason **XmCR_NO_MATCH**. Otherwise, calls the **XmNokCallback** callbacks with reason **XmCR_OK**.

<Apply Button Activated>:
Calls the **XmNapplyCallback** callbacks with reason **XmCR_APPLY**.

<Cancel Button Activated>:
Calls the **XmNcancelCallback** callbacks with reason **XmCR_CANCEL**.

<Help Button Activated>:
Calls the **XmNhelpCallback** callbacks with reason **XmCR_HELP**.

<MapWindow>:
Calls the callbacks for **XmNmapCallback** if the SelectionBox is a child of a Dialog shell.

<UnmapWindow>:
Calls the callbacks for **XmNunmapCallback** if the SelectionBox is the child of a DialogShell.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Composite(3X), **Constraint(3X)**, **Core(3X)**, **XmBulletinBoard(3X)**, **XmCreateSelectionBox(3X)**, **XmCreateSelectionDialog(3X)**, **XmCreatePromptDialog(3X)**, **XmManager(3X)**, and **XmSelectionBoxGetChild(3X)**.

XmSelectionBoxGetChild—A SelectionBox function that is used to access a component

Synopsis

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

Widget XmSelectionBoxGetChild (widget, child)
        Widget          widget;
        unsigned char   child;
```

Description

XmSelectionBoxGetChild is used to access a component within a SelectionBox. The parameters given to the function are the SelectionBox widget and a value indicating which component to access.

widget Specifies the SelectionBox widget ID.

child Specifies a component within the SelectionBox. The following values are legal for this parameter:

- **XmDIALOG_APPLY_BUTTON**
- **XmDIALOG_CANCEL_BUTTON**
- **XmDIALOG_DEFAULT_BUTTON**
- **XmDIALOG_HELP_BUTTON**
- **XmDIALOG_LIST**
- **XmDIALOG_LIST_LABEL**
- **XmDIALOG_OK_BUTTON**
- **XmDIALOG_SELECTION_LABEL**
- **XmDIALOG_SEPARATOR**
- **XmDIALOG_TEXT**
- **XmDIALOG_WORK_AREA**

For a complete definition of SelectionBox and its associated resources, see **XmSelectionBox(3X)**.

Return Value

Returns the widget ID of the specified SelectionBox component. An application should not assume that the returned widget will be of any particular class.

Related Information

XmSelectionBox(3X).

XmSeparator—The Separator widget class

Synopsis

#include <Xm/Separator.h>

Description

Separator is a primitive widget that separates items in a display. Several different line drawing styles are provided, as well as horizontal or vertical orientation.

The Separator line drawing is automatically centered within the height of the widget for a horizontal orientation and centered within the width of the widget for a vertical orientation. An **XtSetValues** with a new **XmNseparatorType** resizes the widget to its minimal height (for horizontal orientation) or its minimal width (for vertical orientation) unless height or width is explicitly set in the **XtSetValues** call.

Separator does not draw shadows around the separator. The Primitive resource **XmNshadowThickness** is used for the Separator's thickness when the **XmNseparatorType** resource is **XmSHADOW_ETCHED_IN**, **XmSHADOW_ETCHED_IN_DASH**, **XmSHADOW_ETCHED_OUT**, or **XmSHADOW_ETCHED_OUT_DASH**.

Separator does not highlight and allows no traversing. The primitive resource **XmNtraversalOn** is forced to False.

The **XmNseparatorType** of **XmNO_LINE** provides an escape to the application programmer who needs a different style of drawing. A pixmap the height of the widget can be created and used as the background pixmap by building an argument list using the **XmNbackgroundPixmap** argument type as defined by **Core**. Whenever the widget is redrawn, its background is displayed containing the desired separator drawing.

Classes

Separator inherits behavior and resources from **Core** and **XmPrimitive**.

The class pointer is **xmSeparatorWidgetClass**.

The class name is **XmSeparator**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix anduse the remaining letters (in either lowercase or

uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmSeparator Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNmargin XmCMargin	0 Dimension	CSG
XmNorientation XmCOrientation	XmHORIZONTAL unsigned char	CSG
XmNseparatorType XmCSeparatorType	XmSHADOW_ETCHED_IN unsigned char	CSG

XmNmargin For horizontal orientation, specifies the space on the left and right sides between the border of the Separator and the line drawn. For vertical orientation, specifies the space on the top and bottom between the border of the Separator and the line drawn.

XmNorientation
Displays Separator vertically or horizontally. This resource can have values of **XmVERTICAL** and **XmHORIZONTAL**.

XmNseparatorType
Specifies the type of line drawing to be done in the Separator widget.

XmSINGLE_LINE
Single line

XmDOUBLE_LINE
Double line

XmSINGLE_DASHED_LINE
Single-dashed line

XmDOUBLE_DASHED_LINE
Double-dashed line

XmNO_LINE
No line

XmSHADOW_ETCHED_IN
A line whose shadows give the effect of a line etched into the window. The thickness of the line is equal to

the value of **XmNshadowThickness**. For horizontal orientation, the top shadow is drawn in **XmNtopShadowColor** and the bottom shadow is drawn in **XmNbottomShadowColor**. For vertical orientation, the left edge is drawn in **XmNtopShadowColor** and the right edge is drawn in **XmNbottomShadowColor**.

XmSHADOW_ETCHED_OUT
A line whose shadows give the effect of an etched line coming out of the window. The thickness of the line is equal to the value of **XmNshadowThickness**. For horizontal orientation, the top shadow is drawn in **XmNbottomShadowColor** and the bottom shadow is drawn in **XmNtopShadowColor**. For vertical orientation, the left edge is drawn in **XmNbottomShadowColor** and the right edge is drawn in **XmNtopShadowColor**.

XmSHADOW_ETCHED_IN_DASH
Identical to **XmSHADOW_ETCHED_IN** except a series of lines creates a dashed line.

XmSHADOW_ETCHED_OUT_DASH
Identical to **XmSHADOW_ETCHED_OUT** except a series of lines creates a dashed line.

Inherited Resources

Separator inherits behavior and resources from the superclasses in the following table. For a complete description of each resource, refer to the reference page for that superclass.

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	0 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	False Boolean	G
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Translations

There are no translations for **XmSeparator**.

Related Information

Core(3X), **XmCreateSeparator(3X)**, and **XmPrimitive(3X)**.

XmSeparatorGadget—The SeparatorGadget widget class

Synopsis

#include <Xm/SeparatoG.h>

Description

SeparatorGadget separates items in a display. Several line drawing styles are provided, as well as horizontal or vertical orientation.

Lines drawn within the SeparatorGadget are automatically centered within the height of the gadget for a horizontal orientation and centered within the width of the gadget for a vertical orientation. An **XtSetValues** with a new **XmNseparatorType** resizes the widget to its minimal height (for horizontal orientation) or its minimal width (for vertical orientation) unless height or width is explicitly set in the **XtSetValues** call.

SeparatorGadget does not draw shadows around the separator. The Gadget resource **XmNshadowThickness** is used for the SeparatorGadget's thickness when the **XmNseparatorType** resource is **XmSHADOW_ETCHED_IN**, **XmSHADOW_ETCHED_IN_DASH**, **XmSHADOW_ETCHED_OUT**, or **XmSHADOW_ETCHED_OUT_DASH**.

SeparatorGadget does not highlight and allows no traversing. The Gadget resource **XmNtraversalOn** is forced to False.

Classes

SeparatorGadget inherits behavior and resources from **Object**, **RectObj**, and **XmGadget**.

The class pointer is **xmSeparatorGadgetClass**.

The class name is **XmSeparatorGadget**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmSeparatorGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNmargin XmCMargin	0 Dimension	CSG
XmNorientation XmCOrientation	XmHORIZONTAL unsigned char	CSG
XmNseparatorType XmCSeparatorType	XmSHADOW_ETCHED_IN unsigned char	CSG

XmNmargin For horizontal orientation, specifies the space on the left and right sides between the border of SeparatorGadget and the line drawn. For vertical orientation, specifies the space on the top and bottom between the border of SeparatorGadget and the line drawn.

XmNorientation

Specifies whether SeparatorGadget is displayed vertically or horizontally. This resource can have values of **XmVERTICAL** and **XmHORIZONTAL**.

XmNseparatorType

Specifies the type of line drawing to be done in the Separator widget.

XmSINGLE_LINE

Single line.

XmDOUBLE_LINE

Double line.

XmSINGLE_DASHED_LINE

Single-dashed line.

XmDOUBLE_DASHED_LINE

Double-dashed line.

XmNO_LINE

No line.

XmSHADOW_ETCHED_IN

A line whose shadows give the effect of a line etched into the window. The thickness of the line is equal to the value of **XmNshadowThickness**. For horizontal orientation, the top shadow is drawn in

XmNtopShadowColor and the bottom shadow is drawn in **XmNbottomShadowColor**. For vertical orientation, the left edge is drawn in **XmNtopShadowColor** and the right edge is drawn in **XmNbottomShadowColor**.

XmSHADOW_ETCHED_OUT
A line whose shadows give the effect of an etched line coming out of the window. The thickness of the line is equal to the value of **XmNshadowThickness**. For horizontal orientation, the top shadow is drawn in **XmNbottomShadowColor** and the bottom shadow is drawn in **XmNtopShadowColor**. For vertical orientation, the left edge is drawn in **XmNbottomShadowColor** and the right edge is drawn in **XmNtopShadowColor**.

XmSHADOW_ETCHED_IN_DASH
Identical to **XmSHADOW_ETCHED_IN** except a series of lines creates a dashed line.

XmSHADOW_ETCHED_OUT_DASH
Identical to **XmSHADOW_ETCHED_OUT** except a series of lines creates a dashed line.

Inherited Resources

SeparatorGadget inherits behavior and resources from the superclasses in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	G
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	G
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	0 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	G
XmNtraversalOn XmCTraversalOn	False Boolean	G
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

RectObj Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	0 Dimension	N/A
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Behavior

XmSeparatorGadget has no behavior.

Related Information

Object(3X), **RectObject(3X)**, **XmCreateSeparatorGadget(3X)**, and **XmGadget(3X)**.

XmSetColorCalculation—A function to set the procedure used for default color calculation

Synopsis

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

XmColorProc XmSetColorCalculation (color_proc)
        XmColorProc color_proc;
```

Description

XmSetColorCalculation sets the procedure to calculate default colors. This procedure is used to calculate the foreground, top shadow, bottom shadow, and select colors on the basis of a given background color. If called with an argument of NULL, it restores the default procedure used to calculate colors.

color_proc Specifies the procedure to use for color calculation.

Following is a description of the **XmColorProc** type used by **XmSetColorCalculation**:

void (**color_proc*) (*background_color, foreground_color, select_color, top_shadow_color, bottom_shadow_color*)

```
XColor      *background_color;
XColor      *foreground_color;
XColor      *select_color;
XColor      *top_shadow_color;
XColor      *bottom_shadow_color;
```

Specifies the procedure used to calculate default colors. The procedure is passed a pointer to an **XColor** structure representing the background color. The *pixel*, *red*, *green*, and *blue* members of this structure are filled in with values that are valid for the current colormap.

The procedure is passed pointers to **XColor** structures representing the foreground, select, top shadow, and bottom shadow colors to be calculated. The procedure calculates and fills in the *red*, *green*, and *blue* members of these structures. The procedure should not allocate color cells for any of these colors.

background_color
Specifies the background color

foreground_color
Specifies the foreground color to be calculated

select_color Specifies the select color to be calculated

top_shadow_color
Specifies the top shadow color to be calculated

bottom_shadow_color
Specifies the bottom shadow color to be calculated

Return Value

Returns the color calculation procedure that was used at the time this routine was called.

Related Information

XmChangeColor(3X), **XmGetColors(3X)**, and **XmGetColorCalculation(3X)**.

XmSetFontUnit—A function that sets the font unit value for a display

Synopsis

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

void XmSetFontUnit (display, font_unit_value)
        Display         * display;
        int             font_unit_value;
```

Description

XmSetFontUnit provides an external function to initialize font unit values. Applications may want to specify resolution-independent data based on a global font size. See the **XmNunitType** resource description in the reference pages for **XmGadget**, **XmManager**, and **XmPrimitive** for more information on resolution independence.

This function sets the font units for all screens on the display.

NOTE: XmSetFontUnit is obsolete and exists for compatibility with previous releases. Instead of using this function, provide initial values or call **XtSetValues** for the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

display Defines the display for which this font unit value is to be applied

font_unit_value
Specifies the value to be used for both horizontal and vertical font units in the conversion calculations

Related Information

XmConvertUnits(3X), **XmSetFontUnits(3X)**, **XmGadget(3X)**, **XmManager(3X)**, **XmPrimitive(3X)**, and **XmScreen(3X)**.

XmSetFontUnits—A function that sets the font unit value for a display

Synopsis

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

void XmSetFontUnits (display, h_value, v_value)
        Display     * display;
        int         h_value;
        int         v_value;
```

Description

XmSetFontUnits provides an external function to initialize font unit values. Applications may want to specify resolution-independent data based on a global font size. This function must be called before any widgets with resolution-independent data are created. See the **XmNunitType** resource description in the reference pages for **XmGadget**, **XmManager**, and **XmPrimitive** for more information on resolution independence.

This function sets the font units for all screens on the display.

NOTE: XmSetFontUnits is obsolete and exists for compatibility with previous releases. Instead of using this function, provide initial values or call **XtSetValues** for the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

display Defines the display for which this font unit value is to be applied

h_value Specifies the value to be used for horizontal units in the conversion calculations

h_value Specifies the value to be used for vertical units in the conversion calculations

Related Information

XmConvertUnits(3X), **XmSetFontUnit(3X)**, **XmGadget(3X)**, **XmManager(3X)**, **XmPrimitive(3X)**, and **XmScreen(3X)**.

XmSetMenuCursor—A function that modifies the menu cursor for a client

Synopsis

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

void XmSetMenuCursor (display, cursorId)
        Display     * display;
        Cursor      cursorId;
```

Description

XmSetMenuCursor programmatically modifies the menu cursor for a client; after the cursor has been created by the client, this function registers the cursor with the menu system. After calling this function, the specified cursor is displayed whenever this client displays a Motif menu on the indicated display. The client can then specify different cursors on different displays.

This function sets the menu cursor for all screens on the display. **XmSetMenuCursor** is obsolete and exists for compatibility with previous releases. Instead of using this function, provide initial values or call **XtSetValues** for the XmScreen resource **XmNmenuCursor**.

display Specifies the display to which the cursor is to be associated

cursorId Specifies the **X** cursor ID

Related Information

XmScreen(3X).

XmSetProtocolHooks—A VendorShell function that allows pre and post actions to be executed when a protocol message is received from MWM

Synopsis

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

void XmSetProtocolHooks (*shell, property, protocol, prehook, pre_closure, posthook, post_closure*)

Widget	*shell*;
Atom	*property*;
Atom	*protocol*;
XtCallbackProc	*prehook*;
XtPointer	*pre_closure*;
XtCallbackProc	*posthook*;
XtPointer	*post_closure*;

void XmSetWMProtocolHooks (*shell, protocol, prehook, pre_closure, posthook, post_closure*)

Widget	*shell*;
Atom	*protocol*;
XtCallbackProc	*prehook*;
XtPointer	*pre_closure*;
XtCallbackProc	*posthook*;
XtPointer	*post_closure*;

Description

XmSetProtocolHooks is used by shells that want to have pre and post actions executed when a protocol message is received from MWM. Since there is no guaranteed ordering in execution of event handlers or callback lists, this allows the shell to control the flow while leaving the protocol manager structures opaque.

XmSetWMProtocolHooks is a convenience interface. It calls **XmSetProtocolHooks** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

property Specifies the protocol property

protocol Specifies the protocol atom (or an **int** cast to **Atom**)

prehook Specifies the procedure to call before calling entries on the client callback list

pre_closure Specifies the client data to be passed to the prehook when it is invoked

posthook Specifies the procedure to call after calling entries on the client callback list

post_closure Specifies the client data to be passed to the posthook when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmInternAtom(3X)**, and **XmSetWMProtocolHooks(3X)**.

XmSetWMProtocolHooks—A VendorShell convenience interface that allows pre and post actions to be executed when a protocol message is received from the window manager

Synopsis

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

void XmSetWMProtocolHooks (shell, protocol, prehook, pre_closure, posthook,
post_closure)
        Widget          shell;
        Atom            protocol;
        XtCallbackProc  prehook;
        XtPointer       pre_closure;
        XtCallbackProc  posthook;
        XtPointer       post_closure;
```

Description

XmSetWMProtocolHooks is a convenience interface. It calls **XmSetProtocolHooks** with the property value set to the atom returned by interning **WM_PROTOCOLS**.

shell Specifies the widget with which the protocol property is associated

protocol Specifies the protocol atom (or an **int** cast to **Atom**)

prehook Specifies the procedure to call before calling entries on the client callback list

pre_closure Specifies the client data to be passed to the prehook when it is invoked

posthook Specifies the procedure to call after calling entries on the client callback list

post_closure Specifies the client data to be passed to the posthook when it is invoked

For a complete definition of VendorShell and its associated resources, see **VendorShell(3X)**.

Related Information

VendorShell(3X), **XmInternAtom(3X)**, and **XmSetProtocolHooks(3X)**.

XmString—Data type for a compound string

Synopsis

#include <Xm/Xm.h>

Description

XmString is the data type for a compound string. Compound strings include one or more segments, each of which may contain a font list element tag, string direction, and text component. When a compound string is displayed, the font list element tag and the direction are used to determine how to display the text. Whenever a font list element tag is set to **XmFONTLIST_DEFAULT_TAG** the text is handled as a locale text segment.

Calling **XtGetValues** for a resource whose type is XmString yields a copy of the compound string resource value. The application is responsible for using **XmStringFree** to free the memory allocated for the copy.

Refer to the **XmFontList** reference page for a description of the algorithm that associates the font list element tag of a compound string segment with a font list entry in a font list.

The compound string interface consists of the routines listed in **Related Information**.

Related Information

XmStringBaseline(3X), **XmStringByteCompare(3X)**, **XmStringCompare(3X)**, **XmStringConcat(3X)**, **XmStringCopy(3X)**, **XmStringCreate(3X)**, **XmStringCreateLtoR(3X)**, **XmStringCreateLocalized(3X)**, **XmStringCreateSimple(3X)**, **XmStringDirection(3X)**, **XmStringDirectionCreate(3X)**, **XmStringDraw(3X)**, **XmStringDrawImage(3X)**, **XmStringDrawUnderline(3X)**, **XmStringEmpty(3X)**, **XmStringExtent(3X)**, **XmStringFree(3X)**, **XmStringFreeContext(3X)**, **XmStringGetLtoR(3X)**, **XmStringGetNextComponent(3X)**, **XmStringGetNextSegment(3X)**, **XmStringHasSubstring(3X)**, **XmStringHeight(3X)**, **XmStringInitContext(3X)**, **XmStringLength(3X)**, **XmStringLineCount(3X)**, **XmStringNConcat(3X)**, **XmStringNCopy(3X)**, **XmStringPeekNextComponent(3X)**, **XmStringSegmentCreate(3X)**, **XmStringSeparatorCreate(3X)**, **XmStringTable(3X)**, and **XmStringWidth(3X)**.

XmStringBaseline—A compound string function that returns the number of pixels between the top of the character box and the baseline of the first line of text

Synopsis

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

Dimension XmStringBaseline (fontlist, string)
        XmFontList    fontlist;
        XmString      string;
```

Description

XmStringBaseline returns the number of pixels between the top of the character box and the baseline of the first line of text in the provided compound string.

fontlist Specifies the font list

string Specifies the string

Return Value

Returns the number of pixels between the top of the character box and the baseline of the first line of text.

Related Information

XmStringCreate(3X).

XmStringByteCompare—A compound string function that indicates the results of a byte-by-byte comparison

Synopsis

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

Boolean XmStringByteCompare (s1, s2)
        XmString    s1;
        XmString    s2;
```

Description

XmStringByteCompare returns a Boolean indicating the results of a byte-by-byte comparison of two compound strings.

In general, if two compound strings are created with the same (**char** *) string using **XmStringCreateLocalized** in the same language environment, the compound strings compare as equal. If two compound strings are created with the same (**char** *) string and the same font list element tag set other than **XmFONTLIST_DEFAULT_TAG** using **XmStringCreate**, the strings compare as equal.

In some cases, once a compound string is put into a widget, that string is converted into an internal form to allow faster processing. Part of the conversion process strips out unnecessary or redundant information. If an application then does an **XtGetValues** to retrieve a compound string from a widget (specifically, Label and all of its subclasses), it is not guaranteed that the compound string returned is byte-for-byte the same as the string given to the widget originally.

s1 Specifies a compound string to be compared with *s2*

s2 Specifies a compound string to be compared with *s1*

Return Value

Returns True if two compound strings are identical byte-by-byte.

Related Information

XmStringCreate(3X) and **XmStringCreateLocalized(3X)**.

XmStringCompare—A compound string function that compares two strings

Synopsis

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

Boolean XmStringCompare (s1, s2)
        XmString    s1;
        XmString    s2;
```

Description

XmStringCompare returns a Boolean value indicating the results of a semantically equivalent comparison of two compound strings.

Semantically equivalent means that the strings have the same text components, font list element tags, directions, and separators. In general, if two compound strings are created with the same (**char** *) string using **XmStringCreateLocalized** in the same language environment, the compound strings compare as equal. If two compound strings are created with the same (**char** *) string and the same font list element tag other than **XmFONTLIST_DEFAULT_TAG** using **XmStringCreate**, the strings compare as equal.

s1 Specifies a compound string to be compared with *s2*

s2 Specifies a compound string to be compared with *s1*

Return Value

Returns True if two compound strings are equivalent.

Related Information

XmStringCreate(3X) and **XmStringCreateLocalized(3X)**.

XmStringConcat—A compound string function that appends one string to another

Synopsis

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

XmString XmStringConcat (s1, s2)
        XmString    s1;
        XmString    s2;
```

Description

XmStringConcat copies *s2* to the end of *s1* and returns a copy of the resulting compound string. The original strings are preserved. The space for the resulting compound string is allocated within the function. After using this function, free this space by calling **XmStringFree**.

s1	Specifies the compound string to which a copy of *s2* is appended
s2	Specifies the compound string that is appended to the end of *s1*

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X) and **XmStringFree(3X)**.

XmStringCopy—A compound string function that makes a copy of a string

Synopsis

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

XmString XmStringCopy (s1)
        XmString        s1;
```

Description

XmStringCopy makes a copy of a compound string. The space for the resulting compound string is allocated within the function. The application is responsible for managing the allocated space. The memory can be recovered with **XmStringFree**.

s1 Specifies the compound string to be copied

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X) and **XmStringFree(3X)**.

XmStringCreate—A compound string function that creates a compound string

Synopsis

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

XmString XmStringCreate (text, tag)
        char        *text;
        char        *tag;
```

Description

XmStringCreate creates a compound string with two components: text and a font list element tag.

text Specifies a NULL-terminated string to be used as the text component of the compound string.

tag Specifies the font list element tag to be associated with the given text. The value **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.

Return Value

Returns a new compound string.

Related Information

XmFontList(3X), XmFontListAdd(3X), XmFontListAppendEntry(3X), XmFontListCopy(3X), XmFontListCreate(3X), XmFontListEntryCreate(3X), XmFontListEntryFree(3X), XmFontListEntryGetFont(3X), XmFontListEntryGetTag(3X), XmFontListEntryLoad(3X), XmFontListFree(3X), XmFontListFreeFontContext(3X), XmFontListGetNextFont(3X), XmFontListInitFontContext(3X), XmFontListNextEntry(3X), XmFontListRemoveEntry(3X), XmString(3X), XmStringBaseline(3X), XmStringByteCompare(3X), XmStringCompare(3X), XmStringConcat(3X), XmStringCopy(3X), XmStringCreateLocalized(3X), XmStringCreateLtoR(3X), XmStringCreateSimple(3X), XmStringDirection(3X), XmStringDirectionCreate(3X), XmStringDraw(3X), XmStringDrawImage(3X), XmStringDrawUnderline(3X), XmStringEmpty(3X), XmStringExtent(3X), XmStringFree(3X),

XmStringFreeContext(3X), **XmStringGetLtoR(3X)**, **XmStringGetNextComponent(3X)**, **XmStringGetNextSegment(3X)**, **XmStringHasSubstring(3X)**, **XmStringHeight(3X)**, **XmStringInitContext(3X)**, **XmStringLength(3X)**, **XmStringLineCount(3X)**, **XmStringNConcat(3X)**, **XmStringNCopy(3X)**, **XmStringPeekNextComponent(3X)**, **XmStringSegmentCreate(3X)**, **XmStringSeparatorCreate(3X)**, **XmStringTable(3X)**, and **XmStringWidth(3X)**.

XmStringCreateLocalized—A compound string function that creates a compound string in the current locale

Synopsis

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

XmString XmStringCreateLocalized (text)
        char    *text;
```

Description

XmStringCreateLocalized creates a compound string containing the specified text and assigns **XmFONTLIST_DEFAULT_TAG** as the font list entry tag. An identical compound string would result from the function **XmStringCreate** called with **XmFONTLIST_DEFAULT_TAG** explicitly as the font list entry tag.

text Specifies a NULL-terminated string of text encoded in the current locale to be used as the text component of the compound string

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X).

XmStringCreateLtoR—A compound string function that creates a compound string

Synopsis

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

XmString XmStringCreateLtoR (text, tag)
        char        *text;
        char        *tag;
```

Description

XmStringCreateLtoR creates a compound string with two components: text and a font list element tag. This function imposes the semantic of scanning for **\n** characters in the text. When one is found, the text up to that point is put into a segment followed by a separator component. No final separator component is appended to the end of the compound string. The direction defaults to left-to-right. This function assumes that the encoding is single octet rather than double octet per character of text.

text Specifies a NULL-terminated string to be used as the text component of the compound string.

tag Specifies the font list element tag to be associated with the given text. The value **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X).

XmStringCreateSimple—A compound string function that creates a compound string in the language environment of a widget

Synopsis

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

XmString XmStringCreateSimple (text)
        char        * text;
```

Description

XmStringCreateSimple creates a compound string with two components: text and a character set. It derives the character set from the current language environment.

The routine attempts to derive a character set from the value of the LANG environment variable. If this does not result in a valid character set, the routine uses a vendor-specific default. If the vendor has not specified a different value, this default is ISO8859-1.

NOTE: This routine is obsolete and exists for compatibility with previous releases. It has been replaced by **XmStringCreateLocalized**.

text Specifies a NULL-terminated string to be used as the text component of the compound string

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X) and **XmStringCreateLocalized(3X)**.

XmStringDirection—Data type for the direction of display in a string

Synopsis

#include <Xm/Xm.h>

Description

XmStringDirection is the data type for specifying the direction in which the system displays characters of a string, or characters of a segment of a compound string. This is an enumeration with two possible values:

XmSTRING_DIRECTION_L_TO_R
Specifies left to right display

XmSTRING_DIRECTION_R_TO_L
Specifies right to left display

Related Information

XmString(3X).

XmStringDirectionCreate—A compound string function that creates a compound string

Synopsis

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

XmString XmStringDirectionCreate (direction)
        XmStringDirection direction;
```

Description

XmStringDirectionCreate creates a compound string with a single component, a direction with the given value.

direction Specifies the value of the directional component

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X).

XmStringDraw—A compound string function that draws a compound string in an X window

Synopsis

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

void XmStringDraw (d, w, fontlist, string, gc, x, y, width, alignment, layout_direction, clip)
        Display         * d;
        Window          w;
        XmFontList      fontlist;
        XmString        string;
        GC              gc;
        Position        x;
        Position        y;
        Dimension       width;
        unsigned char   alignment;
        unsigned char   layout_direction;
        XRectangle      * clip;
```

Description

XmStringDraw draws a compound string in an X Window. If a compound string segment uses a font list entry that defines a font set, the graphic context passed to this routine will have the GC font member left in an undefined state. The underlying **XmbStringDraw** function called by this routine modifies the font ID field of the GC passed into it and does not attempt to restore the font ID to the incoming value. If the compound string segment is not drawn using a font set, the graphic context must contain a valid font member. Graphic contexts created by **XtGetGC** are not valid for this routine; instead, use **XtAllocateGC** to create a graphic context.

d Specifies the display.

w Specifies the window.

fontlist Specifies the font list.

string Specifies the string.

gc Specifies the graphics context to use.

x Specifies a coordinate of the rectangle that will contain the displayed compound string.

y Specifies a coordinate of the rectangle that will contain the displayed compound string.

width Specifies the width of the rectangle that will contain the displayed compound string.

alignment Specifies how the string will be aligned within the specified rectangle. It is either **XmALIGNMENT_BEGINNING**, **XmALIGNMENT_CENTER**, or **XmALIGNMENT_END**.

layout_direction
Controls the direction in which the segments of the compound string will be laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string will be drawn. If the value is NULL, no clipping will be done.

Related Information

XmStringCreate(3X).

XmStringDrawImage—A compound string function that draws a compound string in an X Window and creates an image

Synopsis

#include <Xm/Xm.h>

void XmStringDrawImage (*d, w, fontlist, string, gc, x, y, width, alignment, layout_direction, clip*)

Display	* *d*;
Window	*w*;
XmFontList	*fontlist*;
XmString	*string*;
GC	*gc*;
Position	*x*;
Position	*y*;
Dimension	*width*;
unsigned char	*alignment*;
unsigned char	*layout_direction*;
XRectangle	* *clip*;

Description

XmStringDrawImage draws a compound string in an X Window and paints both the foreground and background bits of each character. If a compound string segment uses a font list entry that defines a font set, the graphic context passed to this routine will have the GC font member left in an undefined state. The underlying **XmbStringDraw** function called by this routine modifies the font ID field of the GC passed into it and does not attempt to restore the font ID to the incoming value. If the compound string segment is not drawn using a font set, the graphic context must contain a valid font member. Graphic contexts created by **XtGetGC** are not accepted by this routine; instead, use **XtAllocateGC** to create a graphic context.

d	Specifies the display.
w	Specifies the window.
fontlist	Specifies the font list.
string	Specifies the string.
gc	Specifies the graphics context to use.
x	Specifies a coordinate of the rectangle that will contain the displayed compound string.

y Specifies a coordinate of the rectangle that will contain the displayed compound string.

width Specifies the width of the rectangle that will contain the displayed compound string.

alignment Specifies how the string will be aligned within the specified rectangle. It is either **XmALIGNMENT_BEGINNING**, **XmALIGNMENT_CENTER**, or **XmALIGNMENT_END**.

layout_direction
Controls the direction in which the segments of the compound string will be laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string will be drawn. If NULL, no clipping will be done.

Related Information

XmStringCreate(3X).

XmStringDrawUnderline—A compound string function that underlines a string drawn in an X Window

Synopsis

#include <Xm/Xm.h>

void XmStringDrawUnderline (*d, w, fontlist, string, gc, x, y, width, alignment, layout_direction, clip, underline*)

Display	* *d*;
Window	*w*;
XmFontList	*fontlist*;
XmString	*string*;
GC	*gc*;
Position	*x*;
Position	*y*;
Dimension	*width*;
unsigned char	*alignment*;
unsigned char	*layout_direction*;
XRectangle	* *clip*;
XmString	*underline*;

Description

XmStringDrawUnderline draws a compound string in an X Window. If the substring identified by *underline* can be matched in *string*, the substring will be underlined. Once a match has occurred, no further matches or underlining will be done.

If a compound string segment uses a font list entry that defines a font set, the graphic context passed to this routine will have the GC font member left in an undefined state. The underlying **XmbStringDraw** function called by this routine modifies the font ID field of the GC passed into it and does not attempt to restore the font ID to the incoming value. If the compound string segment is not drawn using a font set, the graphic context must contain a valid font member. Graphic contexts created by **XtGetGC** are not accepted by this routine; instead, use **XtAllocateGC** to create a graphic context.

d	Specifies the display.
w	Specifies the window.
fontlist	Specifies the font list.
string	Specifies the string.
gc	Specifies the graphics context to use.

x Specifies a coordinate of the rectangle that will contain the displayed compound string.

y Specifies a coordinate of the rectangle that will contain the displayed compound string.

width Specifies the width of the rectangle that will contain the displayed compound string.

alignment Specifies how the string will be aligned within the specified rectangle. It is one of **XmALIGNMENT_BEGINNING**, **XmALIGNMENT_CENTER**, or **XmALIGNMENT_END**.

layout_direction
Controls the direction in which the segments of the compound string will be laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string will be drawn. If it is NULL, no clipping will be done.

underline Specifies the substring to be underlined.

Related Information

XmStringCreate(3X).

XmStringEmpty—A compound string function that provides information on the existence of non-zero-length text components

Synopsis

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

Boolean XmStringEmpty (s1)
        XmString    s1;
```

Description

XmStringEmpty returns a Boolean value indicating whether any non-zero-length text components exist in the provided compound string. It returns True if there are no text segments in the string. If this routine is passed NULL as the string, it returns True.

s1 Specifies the compound string

Return Value

Returns True if there are no text segments in the string. If this routine is passed NULL as the string, it returns True.

Related Information

XmStringCreate(3X).

XmStringExtent—A compound string function that determines the size of the smallest rectangle that will enclose the compound string

Synopsis

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

void XmStringExtent (fontlist, string, width, height)
        XmFontList    fontlist;
        XmString      string;
        Dimension     *width;
        Dimension     *height;
```

Description

XmStringExtent determines the width and height, in pixels, of the smallest rectangle that will enclose the provided compound string.

fontlist	Specifies the font list
string	Specifies the string
width	Specifies a pointer to the width of the rectangle
height	Specifies a pointer to the height of the rectangle

Related Information

XmStringCreate(3X).

XmStringFree—A compound string function that recovers memory

Synopsis

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

void XmStringFree (string)
        XmString        string;
```

Description

XmStringFree recovers memory used by a compound string.

string Specifies the compound string to be freed

Related Information

XmStringCreate(3X).

XmStringFreeContext—A compound string function that instructs the toolkit that the context is no longer needed

Synopsis

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

void XmStringFreeContext (context)
        XmStringContext context;
```

Description

XmStringFreeContext instructs the toolkit that the context is no longer needed and will not be used without reinitialization.

context Specifies the string context structure that was allocated by the **XmStringInitContext** function

Related Information

XmStringCreate(3X) and **XmStringInitContext(3X)**.

XmStringGetLtoR—A compound string function that searches for a text segment in the input compound string

Synopsis

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

Boolean XmStringGetLtoR (string, tag, text)
        XmString        string;
        XmStringCharSettag;
        char            **text;
```

Description

XmStringGetLtoR searches for a text segment in the input compound string that matches the given font list element tag.

string Specifies the compound string.

tag Specifies the font list element tag associated with the text. A value of **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.

text Specifies a pointer to a NULL-terminated string.

Return Value

Returns True if the matching text segment can be found. On return, *text* will have a NULL-terminated octet sequence containing the matched segment.

Related Information

XmStringCreate(3X).

XmStringGetNextComponent—A compound string function that returns the type and value of the next component in a compound string

Synopsis

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

XmStringComponentType XmStringGetNextComponent(context, text, tag, direction,
unknown_tag, unknown_length, unknown_value)
        XmStringContext         context;
        char                    **text;
        XmStringCharSet         *tag;
        XmStringDirection       *direction;
        XmStringComponentType   *unknown_tag;
        unsigned short          *unknown_length;
        unsigned char           **unknown_value;
```

Description

XmStringGetNextComponent returns the type and value of the next component in the compound string identified by *context*. It is a low-level component function. Components are returned one at a time. On return, only some output parameters will be valid; which ones can be determined by examining the return status. In the case of *text*, *tag*, and *direction* components, only one output parameter is valid. If the return status indicates an unknown component was encountered, the font list element tag, length, and value are returned. This function allocates the space necessary to hold returned values; freeing this space is the caller's responsibility.

context Specifies the string context structure that was allocated by the **XmStringInitContext** function.

text Specifies a pointer to a NULL-terminated string.

tag Specifies a pointer to the font list element tag associated with the text. The value **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.

direction Specifies a pointer to the direction of the text.

unknown_tag Specifies a pointer to the tag of an unknown component.

unknown_length
Specifies a pointer to the length of an unknown component.

unknown_value
Specifies a pointer to the value of an unknown component.

Return Value

Returns the type of component found. The possible values are

- **XmSTRING_COMPONENT_CHARSET**

  This component is obsolete and remains for compatibility with previous releases. This component has been replaced by **XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG**.

- **XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG**
- **XmSTRING_COMPONENT_LOCALE_TEXT**
- **XmSTRING_COMPONENT_TAG**
- **XmSTRING_COMPONENT_TEXT**
- **XmSTRING_COMPONENT_DIRECTION**
- **XmSTRING_COMPONENT_SEPARATOR**
- **XmSTRING_COMPONENT_END**
- **XmSTRING_COMPONENT_UNKNOWN**

Related Information

XmStringCreate(3X) and **XmStringInitContext(3X)**.

XmStringGetNextSegment—A compound string function that fetches the octets in the next segment of a compound string

Synopsis

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

Boolean XmStringGetNextSegment (context, text, tag, direction, separator)
        XmStringContext     context;
        char                **text;
        XmStringCharSet     *tag;
        XmStringDirection   *direction;
        Boolean             *separator;
```

Description

XmStringGetNextSegment fetches the octets in the next segment; repeated calls fetch sequential segments. The *text*, *tag*, and *direction* of the fetched segment are returned each time. A Boolean status is returned to indicate whether a valid segment was successfully parsed.

context Specifies the string context structure which was allocated by the **XmStringInitContext** function

text Specifies a pointer to a NULL-terminated string

tag Specifies a pointer to the font list element tag associated with the text

direction Specifies a pointer to the direction of the text

separator Specifies whether the next component of the compound string is a separator

Return Value

Returns True if a valid segment is found.

Related Information

XmStringCreate(3X) and **XmStringInitContext(3X)**.

XmStringHasSubstring—A compound string function that indicates whether one compound string is contained within another

Synopsis

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

Boolean XmStringHasSubstring (string, substring)
        XmString    string;
        XmString    substring;
```

Description

XmStringHasSubstring indicates whether or not one compound string is contained within another.

string Specifies the compound string to be searched

substring Specifies the compound string to be searched for

Return Value

Returns True if *substring* has a single segment and if its text is completely contained within any single segment of *string*; otherwise, it returns False. If two compound strings created using **XmStringCreateLocalized** in the same language environment satisfy this condition, the function returns True. If two compound strings created with the same character set using **XmStringCreate** satisfy this condition, the function returns True.

Related Information

XmStringCreate(3X) and **XmStringCreateLocalized(3X).**

XmStringHeight—A compound string function that returns the line height of the given compound string

Synopsis

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

Dimension XmStringHeight (fontlist, string)
        XmFontList   fontlist;
        XmString     string;
```

Description

XmStringHeight returns the height, in pixels, of the sum of all the line heights of the given compound string. Separator components delimit lines.

fontlist Specifies the font list

string Specifies the string

Return Value

Returns the height of the specified string.

Related Information

XmStringCreate(3X).

XmStringInitContext—A compound string function that allows applications to read out the content segment by segment

Synopsis

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

Boolean XmStringInitContext (context, string)
        XmStringContext  * context;
        XmString         string;
```

Description

XmStringInitContext maintains a context to allow applications to read out the contents of a compound string segment by segment. This function establishes the context for this read out. This context is used when reading subsequent segments out of the string. A Boolean status is returned to indicate if the input string could be parsed.

context	Specifies a pointer to the allocated context
string	Specifies the string

Return Value

Returns True if the context was allocated.

Related Information

XmStringCreate(3X).

XmStringLength—A compound string function that obtains the length of a compound string

Synopsis

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

int XmStringLength (s1)
        XmString    s1;
```

Description

XmStringLength obtains the length of a compound string. It returns the number of bytes in *s1* including all tags, direction indicators, and separators. If the compound string has an invalid structure, 0 (zero) is returned.

s1 Specifies the compound string

Return Value

Returns the length of the compound string.

Related Information

XmStringCreate(3X).

XmStringLineCount—A compound string function that returns the number of separators plus one in the provided compound string

Synopsis

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

int XmStringLineCount (string)
        XmString        string;
```

Description

XmStringLineCount returns the number of separators plus one in the provided compound string. In effect, it counts the lines of text.

string Specifies the string

Return Value

Returns the number of lines in the compound string

Related Information

XmStringCreate(3X).

XmStringNConcat—A compound string function that appends a specified number of bytes to a compound string

Synopsis

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

XmString XmStringNConcat (s1, s2, num_bytes)
        XmString    s1;
        XmString    s2;
        int         num_bytes;
```

Description

XmStringNConcat appends a specified number of bytes from *s2* to the end of *s1*, including tags, directional indicators, and separators. It then returns the resulting compound string. The original strings are preserved. The space for the resulting compound string is allocated within the function. The application is responsible for managing the allocated space. The memory can be recovered with **XmStringFree**.

s1 Specifies the compound string to which a copy of *s2* is appended.

s2 Specifies the compound string that is appended to the end of *s1*.

num_bytes Specifies the number of bytes of *s2* to append to *s1*. If this value is less than the length of *s2*, as many bytes as possible, but possibly fewer than this value, will be appended to *s1* such that the resulting string is still a valid compound string.

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X) and **XmStringFree(3X)**.

XmStringNCopy—A compound string function that creates a copy of a compound string

Synopsis

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

XmString XmStringNCopy (s1, num_bytes)
        XmString    s1;
        int         num_bytes;
```

Description

XmStringNCopy creates a copy of *s1* that contains a specified number of bytes, including tags, directional indicators, and separators. It then returns the resulting compound string. The original strings are preserved. The space for the resulting compound string is allocated within the function. The application is responsible for managing the allocated space. The memory can be recovered by calling **XmStringFree**.

s1 Specifies the compound string.

num_bytes Specifies the number of bytes of *s1* to copy. If this value is less than the length of *s1*, as many bytes as possible, but possibly fewer than this value, will be appended to *s1* such that the resulting string is still a valid compound string.

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X) and **XmStringFree(3X)**.

XmStringPeekNextComponent—A compound string function that returns the component type of the next component fetched

Synopsis

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

XmStringComponentType XmStringPeekNextComponent (context)
        XmStringContext context;
```

Description

XmStringPeekNextComponent examines the next component that would be fetched by **XmStringGetNextComponent** and returns the component type.

context Specifies the string context structure that was allocated by the **XmStringInitContext** function

Return Value

Returns the type of component found.

Related Information

XmStringCreate(3X), **XmStringGetNextComponent(3X)**, and **XmStringInitContext(3X)**.

XmStringSegmentCreate—A compound string function that creates a compound string

Synopsis

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

XmString XmStringSegmentCreate (text, tag, direction, separator)
        char            * text;
        char            *tag;
        XmStringDirection direction;
        Boolean         separator;
```

Description

XmStringSegmentCreate is a high-level function that assembles a compound string consisting of a font list element tag, a direction component, a text component, and an optional separator component.

text Specifies a NULL-terminated string to be used as the text component of the compound string.

tag Specifies the font list element tag to be associated with the text. The value **XmFONTLIST_DEFAULT_TAG** identifies a locale text segment.

direction Specifies the direction of the text.

separator Specifies separator addition. A value of False means the compound string does not have a separator at the end. A value of True, means a separator immediately follows the text component.

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X).

XmStringSeparatorCreate—A compound string function that creates a compound string

Synopsis

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

XmString XmStringSeparatorCreate ()
```

Description

XmStringSeparatorCreate creates a compound string with a single component, a separator.

Return Value

Returns a new compound string.

Related Information

XmStringCreate(3X).

XmStringTable—Data type for an array of compound strings

Synopsis

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

Description

XmStringTable is the data type for an array of compound strings (objects of type **XmString**).

Related Information

XmString(3X).

XmStringWidth—A compound string function that returns the width of the longest sequence of text components in a compound string

Synopsis

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

Dimension XmStringWidth (fontlist, string)
        XmFontList   fontlist;
        XmString     string;
```

Description

XmStringWidth returns the width, in pixels, of the longest sequence of text components in the provided compound string. Separator components are used to delimit sequences of text components.

fontlist Specifies the font list

string Specifies the string

Return Value

Returns the width of the compound string.

Related Information

XmStringCreate(3X).

XmTargetsAreCompatible—A function that tests whether the target types match between a drop site and source object

Synopsis

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

Boolean XmTargetsAreCompatible (display, export_targets, num_export_targets,
                                import_targets, num_import_targets)
        Display     *display;
        Atom        *export_targets;
        Cardinal    num_export_targets;
        Atom        *import_targets;
        Cardinal    num_import_targets;
```

Description

XmTargetsAreCompatible determines whether the import targets of the destination match any of the export targets of a source. If there is at least one target in common, the function returns True.

display Specifies the display connection.

export_targets
Specifies the list of target atoms associated with the source object. This resource identifies the selection targets the source can convert to.

num_export_targets
Specifies the number of entries in the list of export targets.

import_targets
Specifies the list of targets to be checked against the **XmNexportTargets** of the source associated with the specified DragContext.

num_import_targets
Specifies the number of entries in the *import_targets* list.

Return Value

Returns a Boolean value that indicates whether the destination targets are compatible with the source targets. If there is at least one target in common, the routine returns True; otherwise, returns False.

Related Information

XmDragContext(3X) and **XmDropSite(3X)**.

XmText—The Text widget class

Synopsis

#include <Xm/Text.h>

Description

Text provides a single-line and multiline text editor for customizing both user and programmatic interfaces. It can be used for single-line string entry, forms entry with verification procedures, and full-window editing. It provides an application with a consistent editing system for textual data. The screen's textual data adjusts to the application writer's needs.

Text provides separate callback lists to verify movement of the insert cursor, modification of the text, and changes in input focus. Each of these callbacks provides the verification function with the widget instance, the event that caused the callback, and a data structure specific to the verification type. From this information, the function can verify if the application considers this to be a legitimate state change and can signal the widget whether to continue with the action.

The user interface tailors a new set of translations. The default translations provide key bindings for insert cursor movement, deletion, insertion, and selection of text.

Text allows the user to select regions of text. Selection is based on the model specified in the *Inter-Client Communication Conventions Manual* (ICCCM). Text supports primary and secondary selection.

Mouse and Keyboard Selection

The Text widget allows text to be edited, inserted, and selected. The user can cut, copy, and paste text using the clipboard, primary transfer, or secondary transfer. Text also provides a Drag and Drop facility that enables the user to copy or move data within Text or to a different widget. When keyboard focus policy is set to EXPLICIT, the widget that receives focus is the destination widget. In POINTER mode, any keyboard or mouse operation (except secondary selection) in an editable widget establishes that widget as the destination.

If a destination widget becomes insensitive or uneditable, it forfeits its destination status. In EXPLICIT mode, when a widget becomes insensitive, the focus moves to another widget. If that widget is editable, it becomes the destination widget; otherwise, there is no destination widget. The text of any insensitive Text widget is stippled, indicating its state to the user.

The insertion cursor, displayed as an I-beam, shows where input is inserted. Input is inserted just before the insertion cursor.

Classes

Text inherits behavior and resources from **Core** and **Primitive**.

The class pointer is **xmTextWidgetClass**.

The class name is **XmText**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmText Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNautoShowCursorPosition XmCAutoShowCursorPosition	True Boolean	CSG
XmNcursorPosition XmCCursorPosition	0 XmTextPosition	CSG
XmNeditable XmCEditable	True Boolean	CSG
XmNeditMode XmCEditMode	XmSINGLE_LINE_EDIT int	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNgainPrimaryCallback XmCCallback	NULL XtCallbackList	C
XmNlosePrimaryCallback XmCCallback	NULL XtCallbackList	C
XmNlosingFocusCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	5 Dimension	CSG
XmNmarginWidth XmCMarginWidth	5 Dimension	CSG
XmNmaxLength XmCMaxLength	largest integer int	CSG
XmNmodifyVerifyCallback XmCCallback	NULL XtCallbackList	C
XmNmodifyVerifyCallbackWcs XmCCallback	NULL XtCallbackList	C
XmNmotionVerifyCallback XmCCallback	NULL XtCallbackList	C

Name Class	Default Type	Access
XmNsource XmCSource	Default source XmTextSource	CSG
XmNtopCharacter XmCTextPosition	0 XmTextPosition	CSG
XmNvalue XmCValue	"" String	CSG
XmNvalueChangedCallback XmCCallback	NULL XtCallbackList	C
XmNvalueWcs XmCvalueWcs	(wchar_t *)"" wchar_t *	CSG[1]
XmNverifyBell XmCVerifyBell	dynamic Boolean	CSG

[1] This resource cannot be set in a resource file.

XmNactivateCallback
Specifies the list of callbacks that is called when the user invokes an event that calls the **Activate()** function. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is **XmCR_ACTIVATE**.

XmNautoShowCursorPosition
Ensures that the visible text contains the insert cursor when set to True. If the insert cursor changes, the contents of Text may scroll in order to bring the insertion point into the window.

XmNcursorPosition
Indicates the position in the text where the current insert cursor is to be located. Position is determined by the number of characters from the beginning of the text. The first character position is 0 (zero).

XmNeditable When set to True, indicates that the user can edit the text string. Prohibits the user from editing the text when set to False.

XmNeditMode
Specifies the set of keyboard bindings used in Text. The default, **XmSINGLE_LINE_EDIT**, provides the set of key bindings to be used in editing single-line text. **XmMULTI_LINE_EDIT** provides the set of key bindings to be used in editing multiline text.

The results of placing a Text widget inside a ScrolledWindow when the Text's **XmNeditMode** is **XmSINGLE_LINE_EDIT** are undefined.

XmNfocusCallback
Specifies the list of callbacks called when Text accepts input focus. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is **XmCR_FOCUS**.

XmNgainPrimaryCallback
Specifies the list of callbacks called when an event causes the Text widget to gain ownership of the primary selection. The reason sent by the callback is **XmCR_GAIN_PRIMARY**.

XmNlosePrimaryCallback
Specifies the list of callbacks called when an event causes the Text widget to lose ownership of the primary selection. The reason sent by the callback is **XmCR_LOSE_PRIMARY**.

XmNlosingFocusCallback
Specifies the list of callbacks called before Text loses input focus. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is **XmCR_LOSING_FOCUS**.

XmNmarginHeight
Specifies the distance between the top edge of the widget window and the text, and between the bottom edge of the widget window and the text.

XmNmarginWidth
Specifies the distance between the left edge of the widget window and the text, and between the right edge of the widget window and the text.

XmNmaxLength
Specifies the maximum length of the text string that can be entered into text from the keyboard. This value must be nonnegative. Strings that are entered using the **XmNvalue** resource or the **XmTextSetString** function ignore this resource.

XmNmodifyVerifyCallback
Specifies the list of callbacks called before text is deleted from or inserted into Text. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is **XmCR_MODIFYING_TEXT_VALUE**. When

multiple Text widgets share the same source, only the widget that initiates the source change will generate the **XmNmodifyVerifyCallback**.

If both **XmNmodifyVerifyCallback** and **XmNmodifyVerifyCallbackWcs** are registered callback lists, the procedure(s) in the **XmNmodifyVerifyCallback** list is always executed first; and the resulting data, which may have been modified, is passed to the **XmNmodifyVerifyCallbackWcs** callback routines.

XmNmodifyVerifyCallbackWcs

Specifies the list of callbacks called before text is deleted from or inserted into Text. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStructWcs**. The reason sent by the callback is **XmCR_MODIFYING_TEXT_VALUE**. When multiple Text widgets share the same source, only the widget that initiates the source change will generate the **XmNmodifyVerifyCallbackWcs**.

If both **XmNmodifyVerifyCallback** and **XmNmodifyVerifyCallbackWcs** are registered callback lists, the procedure(s) in the **XmNmodifyVerifyCallback** list is always executed first; and the resulting data, which may have been modified, is passed to the **XmNmodifyVerifyCallbackWcs** callback routines.

XmNmotionVerifyCallback

Specifies the list of callbacks called before the insert cursor is moved to a new position. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is **XmCR_MOVING_INSERT_CURSOR**. It is possible for more than one **XmNmotionVerifyCallback** to be generated from a single action.

XmNsource Specifies the source with which the widget displays text. If no source is specified, the widget creates a default string source. This resource can be used to share text sources between Text widgets.

XmNtopCharacter

Displays the position of text at the top of the window. Position is determined by the number of characters from the beginning of the text. The first character position is 0 (zero).

If the **XmNeditMode** is **XmMULTI_LINE_EDIT**, the line of text that contains the top character is displayed at the top of the widget without shifting the text left or right. **XtGetValues** for **XmNtopCharacter** returns the position of the first character in the line that is displayed at the top of the widget.

XmNvalue Specifies the string value of the Text widget as a **char*** data value. If **XmNvalue** and **XmNvalueWcs** are both defined, the value of **XmNvalueWcs** supersedes that of **XmNvalue**. **XtGetValues** returns a copy of the value of the internal buffer and **XtSetValues** copies the string values into the internal buffer.

XmNvalueChangedCallback
Specifies the list of callbacks called after text is deleted from or inserted into Text. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is **XmCR_VALUE_CHANGED**. When multiple Text widgets share the same source, only the widget that initiates the source change will generate the **XmNvalueChangedCallback**. This callback represents a change in the source in the Text, not in the Text widget. The **XmNvalueChangedCallback** should occur only in pairs with an **XmNmodifyVerifyCallback**, assuming that the *doit* flag in the callback structure of the **XmNmodifyVerifyCallback** is not set to False.

XmNvalueWcs
Specifies the string value of the Text widget as a **wchar_t*** data value. This resource cannot be specified in a resource file.

If **XmNvalue** and **XmNvalueWcs** are both defined, the value of **XmNvalueWcs** supersedes that of **XmNvalue**. **XtGetValues** returns a copy of the value of the internal buffer encoded as a wide character string. **XtSetValues** copies the value of the wide character string into the internal buffer.

XmNverifyBell
Specifies whether the bell should sound when the verification returns without continuing the action. The default depends on the value of the ancestor VendorShell's **XmNaudibleWarning** resource.

XmText Input Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNpendingDelete XmCPendingDelete	True Boolean	CSG
XmNselectionArray XmCSelectionArray	default array XtPointer	CSG
XmNselectionArrayCount XmCSelectionArrayCount	4 int	CSG
XmNselectThreshold XmCSelectThreshold	5 int	CSG

XmNpendingDelete

Indicates that pending delete mode is on when the Boolean value is True. Pending deletion is defined as deletion of the selected text when an insertion is made.

XmNselectionArray

Defines the actions for multiple mouse clicks. The value of the resource is an array of **XmTextScanType** elements. **XmTextScanType** is an enumeration indicating possible actions. Each mouse click performed within half a second of the previous mouse click increments the index into this array and performs the defined action for that index. The possible actions in the order they occur in the default array are

XmSELECT_POSITION

Resets the insert cursor position

XmSELECT_WORD

Selects a word

XmSELECT_LINE

Selects a line of text

XmSELECT_ALL

Selects all of the text

XmNselectionArrayCount

Indicates the number of elements in the **XmNselectionArray** resource. The value must not be negative.

XmNselectThreshold

Specifies the number of pixels of motion that is required to select the next character when selection is performed using the click-drag mode of selection. The value must not be negative.

XmText Output Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNblinkRate XmCBlinkRate	500 int	CSG
XmNcolumns XmCColumns	dynamic short	CSG
XmNcursorPositionVisible XmCCursorPositionVisible	True Boolean	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNresizeHeight XmCResizeHeight	False Boolean	CSG
XmNresizeWidth XmCResizeWidth	False Boolean	CSG
XmNrows XmCRows	dynamic short	CSG
XmNwordWrap XmCWordWrap	False Boolean	CSG

XmNblinkRate

Specifies the blink rate of the text cursor in milliseconds. The time indicated in the blink rate relates to the time the cursor is visible and the time the cursor is invisible (that is, the time it takes to blink the insertion cursor on and off is twice the blink rate). The cursor does not blink when the blink rate is set to 0 (zero). The value must not be negative.

XmNcolumns

Specifies the initial width of the text window as an integer number of characters. The width equals the number of characters specified by this resource multiplied by the maximum character width of the associated font. For proportionate fonts, the actual number of characters that fit on a given line may be greater than the value

specified. The value must be greater than 0 (zero). The default value depends on the value of the **XmNwidth** resource. If no width is specified the default is 20.

XmNcursorPositionVisible
Indicates that the insert cursor position is marked by a blinking text cursor when the Boolean value is True.

XmNfontList Specifies the font list to be used for Text. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is subclass of the XmBulletinBoard or VendorShell widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

Text searches the font list for the first occurrence of a font set that has an **XmFONTLIST_DEFAULT_TAG**. If a default element is not found, the first font set in the font list is used. If the list contains no font sets, the first font in the font list will be used. Refer to **XmFontList(3X)** for more information on a font list structure.

XmNresizeHeight
Indicates that Text will attempt to resize its height to accommodate all the text contained in the widget when the Boolean value is True. If the Boolean value is set to True, the text is always displayed starting from the first position in the source, even if instructed otherwise. This attribute is ignored when the application uses a ScrolledText widget and when **XmNscrollVertical** is True.

XmNresizeWidth
Indicates that Text attempts to resize its width to accommodate all the text contained in the widget when the Boolean value is True. This attribute is ignored if **XmNwordWrap** is True.

XmNrows Specifies the initial height of the text window measured in character heights. This attribute is ignored if the text widget resource **XmNeditMode** is **XmSINGLE_LINE_EDIT**. The value must be greater than 0 (zero). The default value depends on the value of the **XmNheight** resource. If no height is specified the default is 1.

XmNwordWrap
Indicates that lines are to be broken at word breaks (that is, the text does not go off the right edge of the window) when the Boolean value is True. Words are defined as a sequence of characters

separated by white space. White space is defined as a space, tab, or newline. This attribute is ignored if the text widget resource **XmNeditMode** is **XmSINGLE_LINE_EDIT**.

The following resources are used only when text is created in a ScrolledWindow. See the reference page for **XmCreateScrolledText**.

XmText ScrolledText Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNscrollHorizontal XmCScroll	True Boolean	CG
XmNscrollLeftSide XmCScrollSide	dynamic Boolean	CG
XmNscrollTopSide XmCScrollSide	False Boolean	CG
XmNscrollVertical XmCScroll	True Boolean	CG

XmNscrollHorizontal

Adds a ScrollBar that allows the user to scroll horizontally through text when the Boolean value is True. This resource is forced to False when the Text widget is placed in a ScrolledWindow with **XmNscrollingPolicy** set to **XmAUTOMATIC**.

XmNscrollLeftSide

Indicates that the vertical ScrollBar should be placed on the left side of the scrolled text window when the Boolean value is True. This attribute is ignored if **XmNscrollVertical** is False or the Text resource **XmNeditMode** is **XmSINGLE_LINE_EDIT**. The default value may depend on the value of the **XmNstringDirection** resource.

XmNscrollTopSide

Indicates that the horizontal ScrollBar should be placed on the top side of the scrolled text window when the Boolean value is True.

XmNscrollVertical

Adds a ScrollBar that allows the user to scroll vertically through text when the Boolean value is True. This attribute is ignored if the Text resource **XmNeditMode** is **XmSINGLE_LINE_EDIT**. This resource is forced to False when the Text widget is placed in a ScrolledWindow with **XmNscrollingPolicy** set to **XmAUTOMATIC**.

Inherited Resources

Text inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

The Text widget defines a new callback structure for use with verification callbacks. Note that not all fields are relevant for every callback reason. The application must first look at the *reason* field and use only the structure members that are valid for the particular reason. The values *startPos*, *endPos*, and *text* in the callback structure **XmTextVerifyCallbackStruct** may be modified when the callback is received, and these changes will be reflected as changes made to the source of the Text widget. (For example, all keystrokes can be converted to spaces or NULL characters when a password is entered into a Text widget.) The application programmer should not overwrite the *text* field, but should attach data to that pointer.

A pointer to the following structure is passed to callbacks for **XmNlosingFocusCallback**, **XmNmodifyVerifyCallback**, and **XmNmotionVerifyCallback**:

```
typedef struct
{
    int             reason;
    XEvent          * event;
    Boolean         doit;
    XmTextPosition  currInsert, newInsert;
    XmTextPosition  startPos, endPos;
    XmTextBlock     text;
} XmTextVerifyCallbackStruct, *XmTextVerifyPtr;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback. It can be NULL. For example, changes made to the Text widget programmatically do not have an event that can be passed to the associated callback.

doit Indicates whether the action that invoked the callback is performed. Setting *doit* to False negates the action.

currInsert Indicates the current position of the insert cursor.

newInsert Indicates the position at which the user attempts to position the insert cursor.

startPos Indicates the starting position of the text to modify. If the callback is not a modify verification callback, this value is the same as *currInsert*.

endPos Indicates the ending position of the text to modify. If no text is replaced or deleted, the value is the same as *startPos*. If the callback is not a modify verification callback, this value is the same as *currInsert*.

text Points to a structure of type **XmTextBlockRec**. This structure holds the textual information to be inserted.

```
typedef struct
{
    char            *ptr;
    int             length;
    XmTextFormat    format;
} XmTextBlockRec, *XmTextBlock;
```

ptr	Points to the text to be inserted.
length	Specifies the length of the text to be inserted.
format	Specifies the format of the text, either **XmFMT_8_BIT** or **XmFMT_16_BIT**.

A pointer to the following structure is passed to callbacks for **XmNmodifyVerifyCallbackWcs**.

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Boolean         doit;
    XmTextPosition  currInsert, newInsert;
    XmTextPosition  startPos, endPos;
    XmTextBlockWcs  text;
} XmTextVerifyCallbackStructWcs, *XmTextVerifyPtrWcs;
```

reason	Indicates why the callback was invoked.
event	Points to the **XEvent** that triggered the callback. It can be NULL. For example, changes made to the Text widget programmatically do not have an event that can be passed to the associated callback.
doit	Indicates whether the action that invoked the callback is performed. Setting *doit* to False negates the action.
currInsert	Indicates the current position of the insert cursor.
newInsert	Indicates the position at which the user attempts to position the insert cursor.
startPos	Indicates the starting position of the text to modify. If the callback is not a modify verification callback, this value is the same as *currInsert*.
endPos	Indicates the ending position of the text to modify. If no text is replaced or deleted, the value is the same as *startPos*. If the callback is not a modify verification callback, this value is the same as *currInsert*.

text Points to the following structure of type **XmTextBlockRecWcs**. This structure holds the textual information to be inserted.

```
typedef struct
{
    wchar_t         *wcsptr;
    int             length;
} XmTextBlockRecWcs, *XmTextBlockWcs;
```

wcsptr Points to the wide character text to be inserted.

length Specifies the number of characters to be inserted.

The following table describes the reasons why the individual verification callback structure fields are valid.

Reason	Valid Fields
XmCR_LOSING_FOCUS	*reason, event, doit, currInsert, newInsert, startPos, endPos*
XmCR_MODIFYING_TEXT_VALUE	*reason, event, doit, currInsert, newInsert, startPos, endPos, text*
XmCR_MOVING_INSERT_CURSOR	*reason, event, doit, currInsert, newInsert*

Translations

XmText includes translations from **XmPrimitive**. The **XmText** translations are described in the following list. These translations may not directly correspond to a translation table. The actions represent the effective behavior of the associated events, and they may differ in a right-to-left language environment.

BSelect Press: **grab-focus()**

BSelect Motion: **extend-adjust()**

BSelect Release: **extend-end()**

BExtend Press: **extend-start()**

BExtend Motion: **extend-adjust()**

BExtend Release: **extend-end()**

BToggle Press: **move-destination()**

BTransfer Press: **process-bdrag()**

BTransfer Motion:	**secondary-adjust()**
BTransfer Release:	**copy-to()**
MCtrl BTransfer Press:	**process-bdrag()**
MCtrl BTransfer Motion:	**secondary-adjust()**
MCtrl BTransfer Release:	**copy-to()**
MAlt BTransfer Press:	**process-bdrag()**
MAlt BTransfer Motion:	**secondary-adjust()**
MAlt BTransfer Release:	**copy-to()**
MShift BTransfer Press:	**process-bdrag()**
MShift BTransfer Motion:	**secondary-adjust()**
MShift BTransfer Release:	**move-to()**
MAlt MCtrl BTransfer Release:	**copy-to()**
MAlt MShift BTransfer Release:	**move-to()**
KUp:	**process-up()**
MShift KUp:	**process-shift-up()**
MCtrl KUp:	**backward-paragraph()**
MShift MCtrl KUp:	**backward-paragraph(extend)**
KDown:	**process-down()**
MShift KDown:	**process-shift-down()**
MCtrl KDown:	**forward-paragraph()**
MShift MCtrl KDown:	**forward-paragraph(extend)**
KLeft:	**backward-character()**
MShift KLeft:	**key-select(left)**
MCtrl KLeft:	**backward-word()**
MShift MCtrl KLeft:	**backward-word(extend)**
KRight:	**forward-character()**

MShift KRight:	**key-select(right)**
MCtrl KRight:	**forward-word()**
MShift MCtrl KRight:	**forward-word(extend)**
KPageUp:	**previous-page()**
MShift KPageUp:	**previous-page(extend)**
KPageDown:	**next-page()**
MShift KPageDown:	**next-page(extend)**
KPageLeft:	**page-left()**
KPageRight:	**page-right()**
KBeginLine:	**beginning-of-line()**
MShift KBeginLine:	**beginning-of-line(extend)**
KEndLine:	**end-of-line()**
MShift KEndLine:	**end-of-line(extend)**
KBeginData:	**beginning-of-file()**
MShift KBeginData:	**beginning-of-file(extend)**
KEndData:	**end-of-file()**
MShift KEndData:	**end-of-file(extend)**
KTab:	**process-tab()**
KNextField:	**next-tab-group()**
KPrevField:	**prev-tab-group()**
KEnter:	**process-return()**
KActivate:	**activate()**
KDelete:	**delete-next-character()**
KBackSpace:	**delete-previous-character()**
KAddMode:	**toggle-add-mode()**
KSpace:	**self-insert()**

MShift KSpace:	**self-insert()**
KSelect:	**set-anchor()**
KExtend:	**key-select()**
MAny KCancel:	**process-cancel()**
KClear:	**clear-selection()**
KSelectAll:	**select-all()**
KDeselectAll:	**deselect-all()**
KCut:	**cut-clipboard()**
KCopy:	**copy-clipboard()**
KPaste:	**paste-clipboard()**
KPrimaryCut:	**cut-primary()**
KPrimaryCopy:	**copy-primary()**
KPrimaryPaste:	**copy-primary()**
KHelp:	**Help()**
KAny:	**self-insert()**

Action Routines

The **XmText** action routines are

activate(): Calls the callbacks for **XmNactivateCallback**. If the parent is a manager, passes the event to the parent.

backward-character():
Moves the insertion cursor one character to the left. For other effects, see the description of navigation operations in **Keyboard Selection**. This action may have different behavior in a right-to-left language environment.

backward-paragraph(*extend***)**:
If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with no argument, moves the insertion cursor to the first non-whitespace character following the first previous blank line or beginning of the text. If the insertion cursor is already at the beginning of a paragraph, moves the insertion cursor to the beginning of the previous paragraph. For other effects, see the description of navigation operations in **Keyboard Selection**.

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

backward-word(*extend*):
If this action is called with no argument, moves the insertion cursor to the first non-whitespace character after the first whitespace character to the left or after the beginning of the line. If the insertion cursor is already at the beginning of a word, moves the insertion cursor to the beginning of the previous word. For other effects, see the description of navigation operations in **Keyboard Selection**. This action may have different behavior in a locale other than the C locale.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

beep(): Causes the terminal to beep.

beginning-of-file(*extend*):
If this action is called with no argument, moves the insertion cursor to the beginning of the text. For other effects, see the description of navigation operations in **Keyboard Selection**.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

beginning-of-line(*extend*):
If this action is called with no argument, moves the insertion cursor to the beginning of the line. For other effects, see the description of navigation operations in **Keyboard Selection**.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

clear-selection():
Clears the current selection by replacing each character except **<Return>** with a **<space>** character.

copy-clipboard():
Copies the current selection to the clipboard.

copy-primary():
Copies the primary selection to just before the insertion cursor.

copy-to(): If a secondary selection exists, copies the secondary selection to just before the insertion cursor. If no secondary selection exists, copies the primary selection to the pointer location.

cut-clipboard():
Cuts the current selection to the clipboard.

cut-primary():
Cuts the primary selection to just before the insertion cursor.

delete-next-character():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the character following the insertion cursor. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character following the insertion cursor. This may impact the selection.

delete-next-word():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next space, tab or end-of-line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next space, tab or end-of-line character. This may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-previous-character():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. This may impact the selection.

delete-previous-word():

In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the next space, tab or beginning-of-line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the next space, tab or beginning-of-line character. This may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-selection():

Deletes the current selection.

delete-to-end-of-line():

In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next end of line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next end of line character. This may impact the selection.

delete-to-start-of-line():

In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the previous beginning-of-line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the previous beginning-of-line character. This may impact the selection.

deselect-all():

Deselects the current selection.

end-of-file(*extend***)**:

If this action is called with no argument, moves the insertion cursor to the end of the text. For other effects, see the description of navigation operations in **Keyboard Selection**.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

end-of-line(*extend*):
If this action is called with no argument, moves the insertion cursor to the end of the line. For other effects, see the description of navigation operations in **Keyboard Selection**. If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

extend-adjust():
Selects text from the anchor to the pointer position and deselects text outside that range. Moving the pointer over several lines selects text from the anchor to the end of each line the pointer moves over and up to the pointer position on the current line.

extend-end():
Moves the insertion cursor to the position of the pointer.

extend-start():
Adjusts the anchor using the balance-beam method. Selects text from the anchor to the pointer position and deselects text outside that range.

forward-character():
Moves the insertion cursor one character to the right. For other effects, see the description of navigation operations in **Keyboard Selection**. This action may have different behavior in a right-to-left language environment.

forward-paragraph(*extend*):
If **XmNeditMode** is **XmMULTI_LINE_EDIT**, and this action is called with no argument, moves the insertion cursor to the first non-whitespace character following the next blank line. If the insertion cursor is already at the beginning of a paragraph, moves the insertion cursor to the beginning of the next paragraph. For other effects, see the description of navigation operations in **Keyboard Selection**.

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection.**

forward-word(*extend*):

If this action is called with no argument, moves the insertion cursor to the first whitespace character or end-of-line following the next non-whitespace character. If the insertion cursor is already at the end of a word, moves the insertion cursor to the end of the next word. For other effects, see the description of navigation operations in **Keyboard Selection**. This action may have different behavior in a locale other than the C locale.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

grab-focus(): This key binding performs the action defined in the **XmNselectionArray**, depending on the number of multiple mouse clicks. The default selection array ordering is one click to move the insertion cursor to the pointer position, two clicks to select a word, three clicks to select a line of text, and four clicks to select all text. A single click also deselects any selected text and sets the anchor at the pointer position. This action may have different behavior in a locale other than the C locale.

Help(): Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

insert-string(*string*):

If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts *string* before the insertion cursor.

key-select(*direction*):

If called with an argument of **right**, moves the insertion cursor one character to the right and extends the current selection. If called with an argument of **left**, moves the insertion cursor one character to the left and extends the current selection. If called with no argument, extends the current selection. For other effects, see the description of shifted navigation operations and **KExtend** in **Keyboard Selection**.

kill-next-character():

In normal mode, if there is a nonnull selection, deletes the selection; otherwise, kills the character following the insertion cursor and stores the character in the cut buffer. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and

XmNpendingDelete is set to True, deletes the selection; otherwise, kills the character following the insertion cursor and stores the character in the cut buffer. This may impact the selection.

kill-next-word():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, kills the characters following the insertion cursor to the next space, tab or end-of-line character, and stores the characters in the cut buffer. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters following the insertion cursor to the next space, tab or end-of-line character, and stores the characters in the cut buffer. This may impact the selection. This action may have different behavior in a locale other than the C locale.

kill-previous-character():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, kills the character immediately preceding the insertion cursor and stores the character in the cut buffer. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the character immediately preceding the insertion cursor and stores the character in the cut buffer. This may impact the selection.

kill-previous-word():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, kills the characters preceding the insertion cursor up to the next space, tab or beginning-of-line character, and stores the characters in the cut buffer. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters preceding the insertion cursor up to the next space, tab or beginning-of-line character, and stores the characters in the cut buffer. This may impact the selection. This action may have different behavior in a locale other than the C locale.

kill-selection():
Kills the currently selected text and stores the text in the cut buffer.

kill-to-end-of-line():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, kills the characters following the insertion cursor to the next end of line character and stores the characters in the cut buffer. In add mode, if there is a nonnull selection, the cursor is not disjoint

from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters following the insertion cursor to the next end of line character and stores the characters in the cut buffer. This may impact the selection.

kill-to-start-of-line():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, kills the characters preceding the insertion cursor to the next beginning-of-line character and stores the characters in the cut buffer. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters preceding the insertion cursor to the next beginning-of-line character and stores the characters in the cut buffer. This may impact the selection.

move-destination():
Moves the insertion cursor to the pointer position without changing any existing current selection. If there is no current selection, sets the widget as the destination widget.

move-to(): If a secondary selection exists, cuts the secondary selection to the insertion cursor. If no secondary selection exists, cuts the primary selection to the pointer location.

newline(): If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts a newline before the insertion cursor.

newline-and-backup():
If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts a newline just before the insertion cursor and repositions the insertion cursor to the end of the line before the newline.

newline-and-indent():
If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts a newline and then the same number of whitespace characters as at the beginning of the previous line.

next-line(): Moves the insertion cursor to the next line. For other effects, see the description of navigation operations in **Keyboard Selection**.

next-page(*extend*):
If this action is called with no argument, moves the insertion cursor forward one page. For other effects, see the description of navigation operations in **Keyboard Selection**.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

next-tab-group():
Traverses to the next tab group.

page-left(): Scrolls the viewing window left one page of text.

page-right(): Scrolls the viewing window right one page of text.

paste-clipboard():
Pastes the contents of the clipboard before the insertion cursor.

prev-tab-group():
Traverses to the previous tab group.

previous-line():
Moves the insertion cursor to the previous line. For other effects, see the description of navigation operations in **Keyboard Selection**.

previous-page(*extend*):
If this action is called with no argument, moves the insertion cursor back one page. For other effects, see the description of navigation operations in **Keyboard Selection**.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection**.

process-bdrag()
The result of this action is determined by several factors: position of the location cursor, movement of the location cursor, and the interval between a **BTransfer** press and release.

This action copies the current selection to the insertion cursor if text is selected, the location cursor is disjoint from the current selection, and no motion is detected within a given time interval.

It performs a secondary selection and copies the selection to the position where the text was last edited if the cursor is disjoint from a current selection (if one exists), the time interval is exceeded, and movement of the location cursor is detected.

The action drags the current selection if the location cursor is positioned on the selection, the time interval is exceeded, and movement of the location cursor is detected. This action creates a

DragContext object whose **XmNexportTargets** resource value includes target types of **COMPOUND_TEXT**, **STRING**, and **TEXT**.

process-cancel():
Cancels the current **extend-adjust**(), **secondary-adjust**() or **process-bdrag**() operation and leaves the selection state as it was before the operation; otherwise, and if the parent is a manager, passes the event to the parent.

process-down():
If **XmNeditMode** is **XmSINGLE_LINE_EDIT**, and **XmNnavigationType** is **XmNONE**, traverses to the widget below the current one in the tab group.

If **XmNeditMode** is **XmMULTI_LINE_EDIT**, moves the insertion cursor down one line. For other effects, see the description of navigation operations in **Keyboard Selection**.

process-home():
Moves the insertion cursor to the beginning of the line. For other effects, see the description of navigation operations in **Keyboard Selection**.

process-return():
If **XmNeditMode** is **XmSINGLE_LINE_EDIT**, calls the callbacks for **XmNactivateCallback**, and if the parent is a manager, passes the event to the parent. If **XmNeditMode** is **XmMULTI_LINE_EDIT**, inserts a newline.

process-shift-down():
If **XmNeditMode** is **XmMULTI_LINE_EDIT**, moves the insertion cursor down one line. For other effects, see the description of navigation operations in **Keyboard Selection**.

process-shift-up():
If **XmNeditMode** is **XmMULTI_LINE_EDIT**, moves the insertion cursor up one line. For other effects, see the description of navigation operations in **Keyboard Selection**.

process-tab():
If **XmNeditMode** is **XmSINGLE_LINE_EDIT**, traverses to the next tab group. If **XmNeditMode** is **XmMULTI_LINE_EDIT**, inserts a tab.

process-up(): If **XmNeditMode** is **XmSINGLE_LINE_EDIT** and **XmNnavigationType** is **XmNONE**, traverses to the widget above the current one in the tab group.

If **XmNeditMode** is **XmMULTI_LINE_EDIT**, moves the insertion cursor up one line. For other effects, see the description of navigation operations in **Keyboard Selection**.

redraw-display():
Redraws the contents of the text window.

scroll-cursor-vertically(*percentage*):
Scrolls the line containing the insertion cursor vertically to an intermediate position in the visible window based on an input percentage. A value of 0 indicates the top of the window; a value of 100, the bottom of the window. If this action is called with no argument, the line containing the insertion cursor is scrolled vertically to a new position designated by the *y* position of the event passed to the routine.

scroll-one-line-down():
Scrolls the text area down one line.

scroll-one-line-up():
Scrolls the text area up one line.

secondary-adjust():
Extends the secondary selection to the pointer position.

secondary-notify():
Copies the secondary selection to the insertion cursor of the destination widget.

secondary-start():
Marks the beginning of a secondary selection.

select-adjust():
Extends the current selection. The amount of text selected depends on the number of mouse clicks, as specified by the **XmNselectionArray** resource.

select-all(): Selects all text.

select-end(): Extends the current selection. The amount of text selected depends on the number of mouse clicks, as specified by the **XmNselectionArray** resource.

select-start(): Marks the beginning of a new selection region.

self-insert(): If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts the character associated with the key pressed at the insertion cursor.

set-anchor(): Resets the anchor point for extended selections. Resets the destination of secondary selection actions.

set-insertion-point():
Sets the insertion position.

set-selection-hint():
Sets the text source and location of the current selection.

toggle-add-mode():
Toggles the state of Add Mode.

toggle-overstrike():
Toggles the state of the text insertion mode. By default, characters typed into the Text widget are inserted at the position of the insertion cursor. In overstrike mode, characters entered into the Text widget replace the characters that directly follow the insertion cursor. In overstrike mode, when the end of a line is reached, characters are appended to the end of the line.

traverse-home():
Traverses to the first widget in the tab group.

traverse-next():
Traverses to the next widget in the tab group.

traverse-prev():
Traverses to the previous widget in the tab group.

unkill(): Restores last killed text to the position of the insertion cursor.

Additional Behavior

This widget has the following additional behavior:

<FocusIn>: Draws the insertion cursor as solid and starts blinking the cursor.

<FocusOut>: Displays the insertion cursor as a stippled I-beam unless it is the destination widget.

Virtual Bindings

The bindings for virtual keys are vendor specific. The following table lists the Text-specific bindings of virtual keys to actual key event descriptions in OSF/Motif:

Virtual Key Bindings	
Virtual Key	**Actual Key Events**
KActivate	**Ctrl<Key>Return** **<Key>osfActivate**
KExtend	**Ctrl Shift<Key>space** **Shift<Key>osfSelect**
KNextField	**Ctrl<Key>Tab**
KSelect	**Ctrl<Key>space** **<Key>osfSelect**

For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateScrolledText(3X)**, **XmCreateText(3X)**, **XmFontList(3X)**, **XmFontListAppendEntry(3X)**, **XmPrimitive(3X)**, **XmTextClearSelection(3X)**, **XmTextCopy(3X)**, **XmTextCut(3X)**, **XmTextEnableRedisplay(3X)**, **XmTextDisableRedisplay(3X)**, **XmTextField(3X)**, **XmTextFindString(3X)**, **XmTextFindStringWcs(3X)**, **XmTextGetBaseline(3X)**, **XmTextGetEditable(3X)**, **XmTextGetInsertionPosition(3X)**, **XmTextGetLastPosition(3X)**, **XmTextGetMaxLength(3X)**, **XmTextGetSelection(3X)**, **XmTextGetSelectionWcs(3X)**, **XmTextGetSelectionPosition(3X)**, **XmTextGetSource(3X)**, **XmTextGetString(3X)**, **XmTextGetStringWcs(3X)**, **XmTextGetSubstring(3X)**, **XmTextGetSubstringWcs(3X)**, **XmTextGetTopCharacter(3X)**, **XmTextInsert(3X)**, **XmTextInsertWcs(3X)**, **XmTextPaste(3X)**, **XmTextPosToXY(3X)**, **XmTextPosition(3X)**, **XmTextRemove(3X)**, **XmTextReplace(3X)**, **XmTextReplaceWcs(3X)**, **XmTextScroll(3X)**, **XmTextSetAddMode(3X)**, **XmTextSetEditable(3X)**, **XmTextSetHighlight(3X)**, **XmTextSetInsertionPosition(3X)**, **XmTextSetMaxLength(3X)**, **XmTextSetSelection(3X)**, **XmTextSetSource(3X)**, **XmTextSetString(3X)**, **XmTextSetStringWcs(3X)**, **XmTextSetTopCharacter(3X)**, **XmTextShowPosition(3X)**, and **XmTextXYToPos(3X)**.

XmTextClearSelection—A Text function that clears the primary selection

Synopsis

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

void XmTextClearSelection (widget, time)
        Widget      widget;
        Time        time;
```

Description

XmTextClearSelection clears the primary selection in the Text widget.

widget Specifies the Text widget ID.

time Specifies the server time at which the selection value is desired. This should be the time of the event that triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed()**.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextCopy—A Text function that copies the primary selection to the clipboard

Synopsis

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

Boolean XmTextCopy (widget, time)
        Widget      widget;
        Time        time;
```

Description

XmTextCopy copies the primary selected text to the clipboard.

widget Specifies the Text widget ID.

time Specifies the server time at which the selection value is to be modified. This should be the time of the event which triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed()**.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

This function returns False if the primary selection is NULL, if the *widget* doesn't own the primary selection, or if the function is unable to gain ownership of the clipboard selection. Otherwise, it returns True.

Related Information

XmText(3X).

XmTextCut—A Text function that copies the primary selection to the clipboard and deletes the selected text

Synopsis

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

Boolean XmTextCut (widget, time)
        Widget      widget;
        Time        time;
```

Description

XmTextCut copies the primary selected text to the clipboard and then deletes the primary selected text. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the Text widget ID.

time Specifies the server time at which the selection value is to be modified. This should be the time of the event that triggered this request. One source of a valid time stamp is the function **XtLastTimestampProcessed()**.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, or if the function is unable to gain ownership of the clipboard selection. Otherwise, it returns True.

Related Information

XmText(3X).

XmTextDisableRedisplay—A Text function that temporarily prevents visual update of the Text widget

Synopsis

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

void XmTextDisableRedisplay (widget)
        Widget          widget;
```

Description

XmTextDisableRedisplay prevents redisplay of the specified Text widget even though its visual attributes have been modified. The visual appearance of the widget remains unchanged until **XmTextEnableRedisplay** is called. This allows an application to make multiple changes to the widget without causing intermediate visual updates.

widget Specifies the Text widget ID

Related Information

XmTextEnableRedisplay(3X).

XmTextEnableRedisplay—A Text function that forces the visual update of a Text widget

Synopsis

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

void XmTextEnableRedisplay (widget)
        Widget          widget;
```

Description

XmTextEnableRedisplay is used in conjunction with **XmTextDisableRedisplay**, which suppresses visual update of the Text widget. When **XmTextEnableRedisplay** is called, it determines if any visual attributes have been set or modified for the specified widget since **XmTextDisableRedisplay** was called. If so, it forces the widget to update its visual display for all of the intervening changes. Any subsequent changes that affect visual appearance cause the widget to update its visual display.

widget Specifies the Text widget ID

Related Information

XmTextDisableRedisplay(3X).

XmTextField—The TextField class

Synopsis

#include <Xm/TextF.h>

Description

The TextField widget provides a single line text editor for customizing both user and programmatic interfaces. It is used for single-line string entry, and forms entry with verification procedures. It provides an application with a consistent editing system for textual data.

TextField provides separate callback lists to verify movement of the insert cursor, modification of the text, and changes in input focus. Each of these callbacks provides the verification function with the widget instance, the event that caused the callback, and a data structure specific to the verification type. From this information the function can verify if the application considers this to be a legitimate state change and can signal the widget whether to continue with the action.

The user interface tailors a new set of actions. The key bindings have been added for insert cursor movement, deletion, insertion, and selection of text.

TextField allows the user to select regions of text. Selection is based on the model specified in the *Inter-Client Communication Conventions Manual* (ICCCM). TextField supports primary and secondary selection.

Classes

TextField widget inherits behavior and resources from **Core** and **Primitive**.

The class pointer is **xmTextFieldWidgetClass**.

The class name is **XmTextField**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lower case or upper case, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmTextFieldResource Set		
Name **Class**	**Default** **Type**	**Access**
XmNactivateCallback XmCCallback	NULL XtCallbackList	C
XmNblinkRate XmCBlinkRate	500 int	CSG
XmNcolumns XmCColumns	dynamic short	CSG
XmNcursorPosition XmCCursorPosition	0 XmTextPosition	CSG
XmNcursorPositionVisible XmCCursorPositionVisible	True Boolean	CSG
XmNeditable XmCEditable	True Boolean	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNgainPrimaryCallback XmCCallback	NULL XtCallbackList	C
XmNlosePrimaryCallback XmCCallback	NULL XtCallbackList	C
XmNlosingFocusCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	5 Dimension	CSG
XmNmarginWidth XmCMarginWidth	5 Dimension	CSG
XmNmaxLength XmCMaxLength	largest integer int	CSG
XmNmodifyVerifyCallback XmCCallback	NULL XtCallbackList	C

Name Class	Default Type	Access
XmNmodifyVerifyCallbackWcs XmCCallback	NULL XtCallbackList	C
XmNmotionVerifyCallback XmCCallback	NULL XtCallbackList	C
XmNpendingDelete XmCPendingDelete	True Boolean	CSG
XmNresizeWidth XmCResizeWidth	False Boolean	CSG
XmNselectionArray XmCSelectionArray	default array XtPointer	CSG
XmNselectionArrayCount XmCSelectionArrayCount	3 int	CSG
XmNselectThreshold XmCSelectThreshold	5 int	CSG
XmNvalue XmCValue	"" String	CSG
XmNvalueChangedCallback XmCCallback	NULL XtCallbackList	C
XmNvalueWcs XmCValueWcs	(wchar_t *)"" wchar_t *	CSG[1]
XmNverifyBell XmCVerifyBell	dynamic Boolean	CSG

[1] This resource cannot be specified in a resource file.

XmNactivateCallback

Specifies the list of callbacks that is called when the user invokes an event that calls the **Activate()** function. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is **XmCR_ACTIVATE**.

XmNblinkRate

Specifies the blink rate of the text cursor in milliseconds. The time indicated in the blink rate relates to the length of time the cursor is visible and the time the cursor is invisible (that is, the time it will take to blink the insertion cursor on and off will be two times the blink rate). The cursor will not blink when the blink rate is set to 0 (zero). The value must not be negative.

XmNcolumns
Specifies the initial width of the text window as an integer number of characters. The width equals the number of characters specified by this resource multiplied by the maximum character width of the associated font. For proportionate fonts, the actual number of characters that fit on a given line may be greater than the value specified. The value must be greater than 0 (zero). The default value depends on the value of the **XmNwidth** resource. If no width is specified the default is 20.

XmNcursorPosition
Indicates the position in the text where the current insert cursor is to be located. Position is determined by the number of characters from the beginning of the text.

XmNcursorPositionVisible
Indicates that the insert cursor position is marked by a blinking text cursor when the Boolean is True.

XmNeditable When set to True, indicates that the user can edit the text string. A false value will prohibit the user from editing the text.

XmNfocusCallback
Specifies the list of callbacks called when TextField accepts input focus. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is **XmCR_FOCUS**.

XmNfontList Specifies the font list to be used for TextField. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the BulletinBoard or VendorShell widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList(3X)** for more information on a font list structure.

TextField searches the font list for the first occurrence of a font set that has an **XmFONTLIST_DEFAULT_TAG**. If a default element is not found, the first font set in the font list is used. If the list contains no font sets, the first font in the font list is used.

XmNgainPrimaryCallback
Specifies the list of callbacks that are called when the user invokes an event that causes the text widget to gain ownership of the primary selection. The callback reason for this callback is **XmCR_GAIN_PRIMARY**.

XmNlosePrimaryCallback
Specifies the list of callbacks that are called when the user invokes an event that cause the text widget to lose ownership of the primary selection. The callback reason for this callback is **XmCR_LOSE_PRIMARY**.

XmNlosingFocusCallback
Specifies the list of callbacks that are called before TextField widget loses input focus. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is **XmCR_LOSING_FOCUS**.

XmNmarginHeight
Specifies the distance between the top edge of the widget window and the text, and the bottom edge of the widget window and the text.

XmNmarginWidth
Specifies the distance between the left edge of the widget window and the text, and the right edge of the widget window and the text.

XmNmaxLength
Specifies the maximum length of the text string that can be entered into text from the keyboard. This value must be nonnegative. Strings that are entered using the **XmNvalue** resource or the **XmTextFieldSetString** function ignore this resource.

XmNmodifyVerifyCallback
Specifies the list of callbacks that is called before text is deleted from or inserted into TextField. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is **XmCR_MODIFYING_TEXT_VALUE**. When multiple TextField widgets share the same source, only the widget that initiates the source change will generate the **XmNmodifyVerifyCallback**.

If both **XmNmodifyVerifyCallback** and **XmNmodifyVerifyCallbackWcs** are registered callback lists, the procedure(s) in the **XmNmodifyVerifyCallback** list is always executed first; and the resulting data, which may have been modified, is passed to the **XmNmodifyVerifyCallbackWcs** callback routines.

XmNmodifyVerifyCallbackWcs

Specifies the list of callbacks called before text is deleted from or inserted into Text. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStructWcs**. The reason sent by the callback is **XmCR_MODIFYING_TEXT_VALUE**. When multiple TextField widgets share the same source, only the widget that initiates the source change will generate the **XmNmodifyVerifyCallbackWcs**.

If both **XmNmodifyVerifyCallback** and **XmNmodifyVerifyCallbackWcs** are registered callback lists, the procedure(s) in the **XmNmodifyVerifyCallback** list is always executed first; and the resulting data, which may have been modified, is passed to the **XmNmodifyVerifyCallbackWcs** callback routines.

XmNmotionVerifyCallback

Specifies the list of callbacks that is called before the insert cursor is moved to a new position. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is **XmCR_MOVING_INSERT_CURSOR**. It is possible for more than one **XmNmotionVerifyCallback**s to be generated from a single action.

XmNpendingDelete

Indicates that pending delete mode is on when the Boolean is True. Pending deletion is defined as deletion of the selected text when an insertion is made.

XmNresizeWidth

Indicates that TextField widget will attempt to resize its width to accommodate all the text contained in the widget when Boolean is True.

XmNselectionArray
Defines the actions for multiple mouse clicks. Each mouse click performed within a half of a second of the previous mouse click will increment the index into this array and perform the defined action for that index. The possible actions are

XmSELECT_POSITION
Resets the insert cursor position

XmSELECT_WORD
Selects a word

XmSELECT_LINE
Selects a line of text

XmNselectionArrayCount
Specifies the number of actions that are defined in the **XmNselectionArray** resource. The value must not be negative.

XmNselectThreshold
Specifies the number of pixels of motion that is required to select the next character when selection is performed using the click-drag mode of selection. The value must not be negative.

XmNvalue Specifies the string value of the TextField widget as a **char*** data value. If **XmNvalue** and **XmNvalueWcs** are both defined, the value of **XmNvalueWcs** supersedes that of **XmNvalue**. **XtGetValues** returns a copy of the value of the internal buffer and **XtSetValues** copies the string values into the internal buffer.

XmNvalueChangedCallback
Specifies the list of callbacks that is called after text is deleted from or inserted into TextField. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is **XmCR_VALUE_CHANGED**. When multiple TextField widgets share the same source, only the widget that initiates the source change will generate the **XmNvalueChangedCallback**. This callback represents a change in the source in the TextField, not in the TextField widget. The **XmNvalueChangedCallback** should occur only in pairs with a **XmNmodifyVerifyCallback**, assuming that the *doit* flag in the callback structure of the **XmNmodifyVerifyCallback** is not set to False.

XmNvalueWcs

Specifies the string value of the TextField widget as a **wchar_t*** data value. This resource cannot be specified in a resource file.

If **XmNvalue** and **XmNvalueWcs** are both defined, the value of **XmNvalueWcs** supersedes that of **XmNvalue**. **XtGetValues** returns a copy of the value of the internal buffer encoded as a wide character string. **XtSetValues** copies the value of the wide character string into the internal buffer.

XmNverifyBell

Specifies whether a bell will sound when an action is reversed during a verification callback. The default depends on the value of the ancestor VendorShell's **XmNaudibleWarning** resource.

Inherited Resources

TextField widget inherits behavior and resources from the superclasses in the following tables. For a complete description of these resources, refer to the reference page for that superclass.

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

The TextField widget defines a new callback structure for use with verification callbacks. Note that not all of the fields are relevant for every callback reason. The application must first look at the *reason* field and use only the structure members that are valid for the particular reason. The values *startPos*, *endPos*, and *text* in the callback structure **XmTextVerifyCallbackStruct** may be modified upon receiving the callback, and these changes will be reflected as the change made to the source of the TextField widget. (For example, all keystrokes can be converted to spaces or NULL characters when a password is entered into a TextField widget.) The application programmer should not overwrite the *text* field, but should attach data to that pointer.

A pointer to the following structure is passed to the callbacks for **XmNlosingFocusCallback**, **XmNmodifyVerifyCallback**, and **XmNmotionVerifyCallback**.

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Boolean         doit;
    XmTextPosition  currInsert, newInsert;
    XmTextPosition  startPos, endPos;
    XmTextBlock     text;
} XmTextVerifyCallbackStruct, *XmTextVerifyPtr;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** the triggered the callback. It can be NULL. For example, changes made to the Text widget programmatically do not have an event that can be passed to the associated callback.

doit Indicates whether the action that invoked the callback will be performed. Setting *doit* to False negates the action.

currInsert Indicates the current position of the insert cursor.

newInsert Indicates the position at which the user attempts to position the insert cursor.

startPos Indicates the starting position of the text to modify. If the callback is not a modify verification callback, this value is the same as *currInsert*.

endPos Indicates the ending position of the text to modify. If no text is replaced or deleted, then the value is the same as *startPos*. If the callback is not a modify verification callback, this value is the same as *currInsert*.

text Points to the following structure of type **XmTextBlockRec**. This structure holds the textual information to be inserted.

```
typedef struct
{
    char            *ptr;
    int             length;
    XmTextFormat    format
} XmTextBlockRec, *XmTextBlock;
```

ptr — The text to be inserted. *ptr* points to a temporary storage space that is reused after the callback is finished. Therefore, if an application needs to save the text to be inserted, it should copy the text into its own data space.

length — Specifies the length of the text to be inserted.

format — Specifies the format of the text, either **XmFMT_8_BIT** or **XmFMT_16_BIT**.

A pointer to the following structure is passed to callbacks for **XmNmodifyVerifyCallbackWcs**.

```
typedef struct
{
    int             reason;
    XEvent          *event;
    Boolean         doit;
    XmTextPosition  currInsert, newInsert;
    XmTextPosition  startPos, endPos;
    XmWcsTextBlock  text;
} XmTextVerifyCallbackStructWcs, *XmTextVerifyPtrWcs;
```

reason — Indicates why the callback was invoked.

event — Points to the **XEvent** that triggered the callback. It can be NULL. For example, changes made to the Text widget programmatically do not have an event that can be passed to the associated callback.

doit — Indicates whether the action that invoked the callback is performed. Setting *doit* to False negates the action.

currInsert — Indicates the current position of the insert cursor.

newInsert — Indicates the position at which the user attempts to position the insert cursor.

startPos — Indicates the starting position of the text to modify. If the callback is not a modify verification callback, this value is the same as *currInsert*.

endPos — Indicates the ending position of the text to modify. If no text is replaced or deleted, the value is the same as *startPos*. If the callback is not a modify verification callback, this value is the same as *currInsert*.

text Points to the following structure of type **XmTextBlockRecWcs**. This structure holds the textual information to be inserted.

```
typedef struct
{
    wchar_t         *wcsptr;
    int             length;
} XmTextBlockRecWcs, *XmTextBlockWcs;
```

wcsptr Points to the wide character text to be inserted

length Specifies the number of characters to be inserted

The following table describes the reasons for which the individual verification callback structure fields are valid.

Reason	Valid Fields
XmCR_LOSING_FOCUS	*reason, event, doit*
XmCR_MODIFYING_TEXT_VALUE	*reason, event, doit, currInsert, newInsert, startPos, endPos, text*
XmCR_MOVING_INSERT_CURSOR	*reason, event, doit, currInsert, newInsert*

Translations

XmTextField includes translations from **XmPrimitive**. The **XmTextField** translations are described in the following list. These translations may not directly correspond to a translation table. The actions represent the effective behavior of the associated events, and they may differ in a right-to-left language environment.

BSelect Press: **grab-focus()**

BSelect Motion: **extend-adjust()**

BSelect Release: **extend-end()**

BExtend Press: **extend-start()**

BExtend Motion: **extend-adjust()**

BExtend Release: **extend-end()**

BToggle Press: **move-destination()**

BTransfer Press: **process-bdrag()**

BTransfer Motion: **secondary-adjust()**

BTransfer Release:	**copy-to()**
MCtrl BTransfer Press:	**process-bdrag()**
MCtrl BTransfer Motion:	**secondary-adjust()**
MCtrl BTransfer Release:	**copy-to()**
MShift BTransfer Press:	**process-bdrag()**
MShift BTransfer Motion:	**secondary-adjust()**
MShift BTransfer Release:	**move-to()**
MAlt BTransfer Press:	**process-bdrag()**
MAlt BTransfer Motion:	**secondary-adjust()**
MAlt BTransfer Release:	**copy-to()**
MAlt MCtrl BTransfer Release:	**copy-to()**
MAlt MShift BTransfer Release:	**move-to()**
KUp:	**traverse-prev()**
KDown:	**traverse-next()**
KLeft:	**backward-character()**
MShift KLeft:	**key-select(left)**
MCtrl KLeft:	**backward-word()**
MShift MCtrl KLeft:	**backward-word(extend)**
KRight:	**forward-character()**
MShift KRight:	**key-select(right)**
MCtrl KRight:	**forward-word()**
MShift MCtrl KRight:	**forward-word(extend)**
KPageLeft:	**page-left()**
KPageRight:	**page-right()**
KBeginLine:	**beginning-of-line()**
MShift KBeginLine:	**beginning-of-line(extend)**

KEndLine:	**end-of-line()**
MShift KEndLine:	**end-of-line(extend)**
KNextField:	**next-tab-group()**
KPrevField:	**prev-tab-group()**
KActivate:	**activate()**
KDelete:	**delete-next-character()**
KBackSpace:	**delete-previous-character()**
KAddMode:	**toggle-add-mode()**
KSpace:	**self-insert()**
MShift KSpace:	**self-insert()**
KSelect:	**set-anchor()**
KExtend:	**key-select()**
MAny KCancel:	**process-cancel()**
KClear:	**clear-selection()**
KSelectAll:	**select-all()**
KDeselectAll:	**deselect-all()**
KCut:	**cut-clipboard()**
KCopy:	**copy-clipboard()**
KPaste:	**paste-clipboard()**
KPrimaryCut:	**cut-primary()**
KPrimaryCopy:	**copy-primary()**
KPrimaryPaste:	**copy-primary()**
KHelp:	**Help()**
KAny:	**self-insert()**

Action Routines

The **XmText** action routines are

activate(): Calls the callbacks for **XmNactivateCallback**. If the parent is a manager, passes the event to the parent.

backward-character():
Moves the insertion cursor one character to the left. For other effects, see the description of navigation operations in **Keyboard Selection** in **XmText(3X)**. This action may have different behavior in a right-to-left language environment.

backward-word(*extend*):
If this action is called with no argument, moves the insertion cursor to the first non-whitespace character after the first whitespace character to the left or after the beginning of the line. If the insertion cursor is already at the beginning of a word, moves the insertion cursor to the beginning of the previous word. For other effects, see the description of navigation operations in **Keyboard Selection** in **XmText(3X)**. This action may have different behavior in a locale other than the C locale.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection** in **XmText(3X)**.

beginning-of-line(*extend*):
If this action is called with no argument, moves the insertion cursor to the beginning of the line. For other effects, see the description of navigation operations in **Keyboard Selection** in **XmText(3X)**.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection** in **XmText(3X)**.

clear-selection():
Clears the current selection by replacing each character except **<Return>** with a **<space>** character.

copy-clipboard():
Copies the current selection to the clipboard.

copy-primary():
Copies the primary selection to just before the insertion cursor.

copy-to(): If a secondary selection exists, copies the secondary selection to just before the insertion cursor. If no secondary selection exists, copies the primary selection to the pointer location.

cut-clipboard():
Cuts the current selection to the clipboard.

cut-primary():
Cuts the primary selection to just before the insertion cursor.

delete-next-character():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the character following the insertion cursor. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character following the insertion cursor. This may impact the selection.

delete-next-word():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next space, tab or end-of-line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next space, tab or end-of-line character. This may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-previous-character():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. This may impact the selection.

delete-previous-word():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the next space, tab or beginning-of-line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the next space, tab or beginning-of-line character. This

may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-selection():
Deletes the current selection.

delete-to-end-of-line():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next end-of- line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next end of line character. This may impact the selection.

delete-to-start-of-line():
In normal mode, if there is a nonnull selection, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the previous beginning-of-line character. In add mode, if there is a nonnull selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the previous beginning-of-line character. This may impact the selection.

deselect-all():
Deselects the current selection.

end-of-line(*extend*):
If this action is called with no argument, moves the insertion cursor to the end of the line. For other effects, see the description of navigation operations in **Keyboard Selection** in **XmText(3X)**. If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection** in **XmText(3X)**.

extend-adjust():
Selects text from the anchor to the pointer position and deselects text outside that range.

extend-end():
Moves the insertion cursor to the position of the pointer.

extend-start():

Adjusts the anchor using the balance-beam method. Selects text from the anchor to the pointer position and deselects text outside that range.

forward-character():

Moves the insertion cursor one character to the right. For other effects, see the description of navigation operations in the **Keyboard Selection** in **XmText(3X)**. This action may have different behavior in a right-to-left language environment.

forward-word(*extend***)**:

If this action is called with no argument, moves the insertion cursor to the first whitespace character or end-of-line following the next non-whitespace character. If the insertion cursor is already at the end of a word, moves the insertion cursor to the end of the next word. For other effects, see the description of navigation operations in the **Keyboard Selection** in **XmText(3X)**. This action may have different behavior in a locale other than the C locale.

If called with an argument of **extend**, moves the insertion cursor as in the case of no argument and extends the current selection. For other effects, see the description of shifted navigation operations in **Keyboard Selection** in **XmText(3X)**.

grab-focus(): This key binding performs the action defined in the **XmNselectionArray**, depending on the number of multiple mouse clicks. The default selection array ordering is one click to move the insertion cursor to the pointer position, two clicks to select a word, three clicks to select a line of text, and four clicks to select all text. A single click also deselects any selected text and sets the anchor at the pointer position. This action may have different behavior in a locale other than the C locale.

Help(): Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

key-select(*direction***)**:

If called with an argument of **right**, moves the insertion cursor one character to the right and extends the current selection. If called with an argument of **left**, moves the insertion cursor one character to the left and extends the current selection. If called with no argument, extends the current selection. For other effects, see the description of shifted navigation operations and **KExtend** in **Keyboard Selection** in **XmText(3X)**.

move-destination():
Moves the insertion cursor to the pointer position without changing any existing current selection. If there is no current selection, sets the widget as the destination widget.

move-to(): If a secondary selection exists, cuts the secondary selection to just before the insertion cursor. If no secondary selection exists, cuts the primary selection to the pointer location.

next-tab-group():
Traverses to the next tab group.

page-left(): Scrolls the viewing window left one page of text.

page-right(): Scrolls the viewing window right one page of text.

paste-clipboard():
Pastes the contents of the clipboard before the insertion cursor.

prev-tab-group():
Traverses to the previous tab group.

process-bdrag()
The result of this action is determined by several factors: position of the location cursor, movement of the location cursor, and the interval between a **BTransfer** press and release.

This action copies the current selection to the insertion cursor if text is selected, the location cursor is disjoint from the selection, and no motion is detected within a given time interval.

It performs a secondary selection and copies the selection to the position where the text was last edited if the cursor is disjoint from a current selection (if one exists), the time interval is exceeded, and movement of the location cursor is detected.

The action drags the current selection if the location cursor is positioned on the selection, the time interval is exceeded, and movement of the location cursor is detected. This action creates a DragContext object whose **XmNexportTargets** resource value includes target types of **COMPOUND_TEXT**, **STRING**, and **TEXT**.

process-cancel():
Cancels the current **extend-adjust()**, **secondary-adjust()** or **process-bdrag** operation and leaves the selection state as it was before the operation; otherwise, and the parent is a manager, it passes the event to the parent.

secondary-adjust():
Extends the secondary selection to the pointer position.

secondary-start():
Marks the beginning of a secondary selection.

select-all(): Selects all text.

self-insert(): If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts the character associated with the key pressed before the insertion cursor.

set-anchor(): Resets the anchor point for extended selections. Resets the destination of secondary selection actions.

toggle-add-mode():
Toggles the state of Add Mode.

toggle-overstrike():
Toggles the state of the text insertion mode. By default, characters typed into the TextField widget are inserted before the position of the insertion cursor. In overstrike mode, characters entered into the TextField widget replace the characters that directly follow the insertion cursor. In overstrike mode, when the end of a line is reached, characters are appended to the end of the line.

traverse-home():
Traverses to the first widget in the tab group.

traverse-next():
Traverses to the next widget in the tab group.

traverse-prev():
Traverses to the previous widget in the tab group.

Additional Behavior

This widget has the following additional behavior:

<FocusIn>: Draws the insertion cursor as solid and starts blinking the cursor.

<FocusOut>: Displays the insertion cursor as a stippled I-beam unless it is the destination widget.

Virtual Bindings

The bindings for virtual keys are vendor specific. The following table lists the TextField-specific bindings of virtual keys to actual key event descriptions in OSF/Motif.

Virtual Key Bindings	
Virtual Key	**Actual Key Events**
KExtend	**Ctrl Shift<Key>space** **Shift<Key>osfSelect**
KSelect	**Ctrl<Key>space** **<Key>osfSelect**

For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateTextField(3X)**, **XmFontList(3X)**, **XmFontListAppendEntry(3X)**, **XmPrimitive(3X)**, **XmTextFieldClearSelection(3X)**, **XmTextFieldCopy(3X)**, **XmTextFieldCut(3X)**, **XmTextFieldGetBaseline(3X)**, **XmTextFieldGetEditable(3X)**, **XmTextFieldGetInsertionPosition(3X)**, **XmTextFieldGetLastPosition(3X)**, **XmTextFieldGetMaxLength(3X)**, **XmTextFieldGetSelection(3X)**, **XmTextFieldGetSelectionPosition(3X)**, **XmTextFieldGetSelectionWcs(3X)**, **XmTextFieldGetString(3X)**, **XmTextFieldGetStringWcs(3X)**, **XmTextFieldGetSubstring(3X)**, **XmTextFieldGetSubstringWcs(3X)**, **XmTextFieldInsert(3X)**, **XmTextFieldInsertWcs(3X)**, **XmTextFieldPaste(3X)**, **XmTextFieldPosToXY(3X)**, **XmTextFieldRemove(3X)**, **XmTextFieldReplace(3X)**, **XmTextFieldReplaceWcs(3X)**, **XmTextFieldSetAddMode(3X)**, **XmTextFieldSetEditable(3X)**, **XmTextFieldSetHighlight(3X)**, **XmTextFieldSetInsertionPosition(3X)**, **XmTextFieldSetMaxLength(3X)**, **XmTextFieldSetSelection(3X)**, **XmTextFieldSetString(3X)**, **XmTextFieldSetStringWcs(3X)**, **XmTextFieldShowPosition(3X)**, and **XmTextFieldXYToPos(3X)**.

XmTextFieldClearSelection—A TextField function that clears the primary selection

Synopsis

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

void XmTextFieldClearSelection (widget, time)
        Widget      widget;
        Time        time;
```

Description

XmTextFieldClearSelection clears the primary selection in the TextField widget.

widget Specifies the TextField widget ID.

time Specifies the time at which the selection value is desired. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X).

XmTextFieldCopy—A TextField function that copies the primary selection to the clipboard

Synopsis

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

Boolean XmTextFieldCopy (widget, time)
        Widget      widget;
        Time        time;
```

Description

XmTextFieldCopy copies the primary selected text to the clipboard.

widget Specifies the TextField widget ID.

time Specifies the time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, or if the function is unable to gain ownership of the clipboard selection. Otherwise, it returns True.

Related Information

XmTextField(3X).

XmTextFieldCut—A TextField function that copies the primary selection to the clipboard and deletes the selected text

Synopsis

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

Boolean XmTextFieldCut (widget, time)
        Widget      widget;
        Time        time;
```

Description

XmTextFieldCut copies the primary selected text to the clipboard and then deletes the primary selected text. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the TextField widget ID.

time Specifies the time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, or if the function is unable to gain ownership of the clipboard selection. Otherwise, it returns True.

Related Information

XmTextField(3X).

XmTextFieldGetBaseline—A TextField function that accesses the *x* position of the first baseline

Synopsis

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

int XmTextFieldGetBaseline (widget)
        Widget          widget;
```

Description

XmTextFieldGetBaseline accesses the *x* position of the first baseline in the TextField widget, relative to the *x* position of the top of the widget.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns an integer value that indicates the *x* position of the first baseline in the TextField widget. The calculation takes into account the margin height, shadow thickness, highlight thickness, and font ascent of the first font in the fontlist. In this calculation, the *x* position of the top of the widget is 0 (zero).

Related Information

XmTextField(3X).

XmTextFieldGetEditable—A TextField function that accesses the edit permission state

Synopsis

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

Boolean XmTextFieldGetEditable (widget)
        Widget          widget;
```

Description

XmTextFieldGetEditable accesses the edit permission state of the TextField widget.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns a Boolean value that indicates the state of the **XmNeditable** resource.

Related Information

XmTextField(3X).

XmTextFieldGetInsertionPosition—A TextField function that accesses the position of the insertion cursor

Synopsis

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

XmTextPosition XmTextFieldGetInsertionPosition (widget)
        Widget          widget;
```

Description

XmTextFieldGetInsertionPosition accesses the insertion cursor position of the TextField widget.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns an **XmTextPosition** value that indicates the state of the **XmNcursorPosition** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

Related Information

XmTextField(3X).

XmTextFieldGetLastPosition—A TextField function that accesses the position of the last text character

Synopsis

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

XmTextPosition XmTextFieldGetLastPosition (widget)
        Widget          widget;
```

Description

XmTextFieldGetLastPosition accesses the position of the last character in the text buffer of the TextField widget.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns an **XmTextPosition** value that indicates the position of the last character in the text buffer. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

Related Information

XmTextField(3X).

XmTextFieldGetMaxLength—A TextField function that accesses the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

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

int XmTextFieldGetMaxLength (widget)
        Widget          widget;
```

Description

XmTextFieldGetMaxLength accesses the value of the current maximum allowable length of the text string in the TextField widget entered from the keyboard. The maximum allowable length prevents the user from entering a text string larger than this limit.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns the integer value that indicates the string's maximum allowable length that can be entered from the keyboard.

Related Information

XmTextField(3X).

XmTextFieldGetSelection—A TextField function that retrieves the value of the primary selection

Synopsis

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

char * XmTextFieldGetSelection (widget)
        Widget          widget;
```

Description

XmTextFieldGetSelection retrieves the value of the primary selection. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns a character pointer to the string that is associated with the primary selection.

Related Information

XmTextField(3X) and **XmTextFieldGetSelectionWcs(3X)**.

XmTextFieldGetSelectionPosition—A TextField function that accesses the position of the primary selection

Synopsis

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

Boolean XmTextFieldGetSelectionPosition (widget, left, right)
        Widget          widget;
        XmTextPosition *left;
        XmTextPosition *right;
```

Description

XmTextFieldGetSelectionPosition accesses the left and right position of the primary selection in the text buffer of the TextField widget.

widget Specifies the TextField widget ID.

left Specifies the pointer in which the position of the left boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

right Specifies the pointer in which the position of the right boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

This function returns True if the widget owns the primary selection; otherwise, it returns False.

Related Information

XmTextField(3X).

XmTextFieldGetSelectionWcs—A TextField function that retrieves the value of a wide character encoded primary selection

Synopsis

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

wchar_t * XmTextFieldGetSelectionWcs (widget)
        Widget          widget;
```

Description

XmTextFieldGetSelectionWcs retrieves the value of the primary selection, encoded in a wide character format. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the wide character buffer by calling **XtFree**.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns the wide character string that is associated with the primary selection in the TextField widget.

Related Information

XmTextField(3X) and **XmTextFieldGetSelection(3X)**.

XmTextFieldGetString—A TextField function that accesses the string value

Synopsis

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

char * XmTextFieldGetString (widget)
        Widget          widget;
```

Description

XmTextFieldGetString accesses the string value of the TextField widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns a character pointer to the string value of the TextField widget. Returns an empty string if the length of the TextField widget's string is 0 (zero).

Related Information

XmTextField(3X) and **XmTextFieldGetStringWcs(3X)**.

XmTextFieldGetStringWcs—A TextField function that retrieves a copy of the wide character string value of a TextField widget

Synopsis

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

wchar_t * XmTextFieldGetStringWcs (widget)
        Widget          widget;
```

Description

XmTextFieldGetStringWcs retrieves a copy of the wide character string value of the TextField widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns the wide character string value of the TextField widget. The function returns an empty string if the length of the TextField widget's string is 0 (zero).

Related Information

XmTextField(3X) and **XmTextFieldGetString(3X)**.

XmTextFieldGetSubstring—A TextField function that retrieves a copy of a portion of the internal text buffer

Synopsis

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

int XmTextFieldGetSubstring (widget, start, num_chars, buffer_size, buffer)
        Widget          widget;
        XmTextPosition  start;
        int             num_chars;
        int             buffer_size;
        char            *buffer;
```

Description

XmTextFieldGetSubstring retrieves a copy of a portion of the internal text buffer of a TextField widget. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

The size of the required buffer depends on the maximum number of bytes per character (**MB_CUR_MAX**) for the current locale. **MB_CUR_MAX** is a macro defined in **stdlib.h**. The buffer should be large enough to contain the substring to be copied and a NULL terminator. Use the following equation to calculate the size of buffer the application should provide:

buffer_size = (*num_chars* * **MB_CUR_MAX**) + **1**

widget Specifies the TextField widget ID.

start Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

num_chars Specifies the number of characters to be copied into the provided buffer.

buffer_size Specifies the size of the supplied buffer in bytes. This size should account for a NULL terminator.

buffer Specifies the character buffer into which the internal text buffer will be copied.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

XmCOPY_SUCCEEDED
The function was successful.

XmCOPY_FAILED
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget's buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmTextField(3X) and **XmTextFieldGetSubstringWcs(3X)**.

XmTextFieldGetSubstringWcs—A TextField function that retrieves a a portion of a wide character internal text buffer

Synopsis

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

int XmTextFieldGetSubstringWcs (widget, start, num_chars, buffer_size, buffer)
        Widget          widget;
        XmTextPosition  start;
        int             num_chars;
        int             buffer_size;
        wchar_t         *buffer;
```

Description

XmTextFieldGetSubstringWcs retrieves a copy of a portion of the internal text buffer of a TextField widget that is stored in a wide character format. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

widget Specifies the TextField widget ID.

start Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

num_chars Specifies the number of **wchar_t** characters to be copied into the provided buffer.

buffer_size Specifies the size of the supplied buffer as a number of **wchar_t** storage locations. The minimum size is *num_chars* + 1.

buffer Specifies the wide character buffer into which the internal text buffer will be copied.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

XmCOPY_SUCCEEDED
The function was successful.

XmCOPY_FAILED
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
The requested number of characters extended beyond the internal buffer. The function copied characters to the end of the buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmTextField(3X) and **XmTextFieldGetSubstring(3X)**.

XmTextFieldInsert—A TextField function that inserts a character string into a text string

Synopsis

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

void XmTextFieldInsert (widget, position, value)
        Widget          widget;
        XmTextPosition  position;
        char            * value;
```

Description

XmTextFieldInsert inserts a character string into the text string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the *position* parameter must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the TextField widget ID

position Specifies the position in the text string where the character string is to be inserted

value Specifies the character string value to be added to the text widget

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X) and **XmTextFieldInsertWcs(3X)**.

XmTextFieldInsertWcs—A TextField function that inserts a wide character string into a TextField widget

Synopsis

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

void XmTextFieldInsertWcs (widget, position, wcstring)
        Widget          widget;
        XmTextPosition  position;
        wchar_t         *wcstring;
```

Description

XmTextFieldInsertWcs inserts a wide character string into the TextField widget at a specified location. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the *position* parameter must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the TextField widget ID

position Specifies the position in the text string where the new character string is to be inserted

wcstring Specifies the wide character string value to be added to the TextField widget

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X) and **XmTextFieldInsert(3X)**.

XmTextFieldPaste—A TextField function that inserts the clipboard selection

Synopsis

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

Boolean XmTextFieldPaste (widget)
        Widget      widget;
```

Description

XmTextFieldPaste inserts the clipboard selection at the insertion cursor of the destination widget. If **XmNpendingDelete** is True and the insertion cursor is inside the current selection, the clipboard selection replaces the selected text.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

This function returns False if the *widget* does not own the primary selection. Otherwise, it returns True.

Related Information

XmTextField(3X).

XmTextFieldPosToXY—A TextField function that accesses the *x* and *y* position of a character position

Synopsis

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

Boolean XmTextFieldPosToXY (widget, position, x, y)
        Widget          widget;
        XmTextPosition  position;
        Position        *x;
        Position        *y;
```

Description

XmTextFieldPosToXY accesses the *x* and *y* position, relative to the upper left corner of the TextField widget, of a given character position in the text buffer.

widget Specifies the TextField widget ID

position Specifies the character position in the text for which the *x* and *y* position is accessed. This is an integer number of characters from the beginning of the buffer. The first character position is 0.

x Specifies the pointer in which the *x* position, relative to the upper left corner of the widget, is returned. This value is meaningful only if the function returns True.

y Specifies the pointer in which the *y* position, relative to the upper left corner of the widget, is returned. This value is meaningful only if the function returns True.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

This function returns True if the character position is displayed in the TextField widget; otherwise, it returns False, and no *x* or *y* value is returned.

Related Information

XmTextField(3X).

XmTextFieldRemove—A TextField function that deletes the primary selection

Synopsis

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

Boolean XmTextFieldRemove (widget)
        Widget      widget;
```

Description

XmTextFieldRemove deletes the primary selected text. If there is a selection, this routine also calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the TextField widget ID

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

This function returns False if the primary selection is NULL or if the *widget* does not own the primary selection. Otherwise, it returns True.

Related Information

XmTextField(3X).

XmTextFieldReplace—A TextField function that replaces part of a text string

Synopsis

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

void XmTextFieldReplace (widget, from_pos, to_pos, value)
        Widget          widget;
        XmTextPosition  from_pos;
        XmTextPosition  to_pos;
        char            * value;
```

Description

XmTextFieldReplace replaces part of the text string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the parameter *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both parameters, *from_pos* and *to_pos*, must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the TextField widget ID

from_pos Specifies the start position of the text to be replaced

to_pos Specifies the end position of the text to be replaced

value Specifies the character string value to be added to the text widget

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X). **XmTextFieldReplaceWcs(3X)**.

XmTextFieldReplaceWcs—A TextField function that replaces part of a wide character string in a TextField widget

Synopsis

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

void XmTextFieldReplaceWcs (widget, from_pos, to_pos, wcstring)
        Widget          widget;
        XmTextPosition  from_pos;
        XmTextPosition  to_pos;
        wchar_t         *wcstring;
```

Description

XmTextFieldReplaceWcs replaces part of the wide character string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the parameter *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both parameters, *from_pos* and *to_pos*, must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the TextField widget ID

from_pos Specifies the start position of the text to be replaced

to_pos Specifies the end position of the text to be replaced

wcstring Specifies the wide character string value to be added to the TextField widget

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X) and **XmTextFieldReplace(3X)**.

XmTextFieldSetAddMode—A TextField function that sets the state of Add mode

Synopsis

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

void XmTextFieldSetAddMode (widget, state)
        Widget      widget;
        Boolean     state;
```

Description

XmTextFieldSetAddMode controls whether or not the TextField widget is in Add mode. When the widget is in Add mode, the insert cursor can be moved without disturbing the primary selection.

widget Specifies the TextField widget ID

state Specifies whether or not the widget is in Add mode. A value of True turns on Add mode; a value of False turns off Add mode.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X).

XmTextFieldSetEditable—A TextField function that sets the edit permission

Synopsis

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

void XmTextFieldSetEditable (widget, editable)
        Widget      widget;
        Boolean     editable;
```

Description

XmTextFieldSetEditable sets the edit permission state of the TextField widget. When set to True, the text string can be edited.

widget Specifies the TextField widget ID

editable Specifies a Boolean value that when True allows text string edits

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X).

XmTextFieldSetHighlight—A TextField function that highlights text

Synopsis

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

void XmTextFieldSetHighlight (widget, left, right, mode)
        Widget          widget;
        XmTextPosition  left;
        XmTextPosition  right;
        XmHighlightMode mode;
```

Description

XmTextFieldSetHighlight highlights text between the two specified character positions. The *mode* parameter determines the type of highlighting. Highlighting text merely changes the visual appearance of the text; it does not set the selection.

widget Specifies the TextField widget ID.

left Specifies the position of the left boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

right Specifies the position of the right boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

mode Specifies the type of highlighting to be done. A value of **XmHIGHLIGHT_NORMAL** removes highlighting. A value of **XmHIGHLIGHT_SELECTED** highlights the test using reverse video. A value of **XmHIGHLIGHT_SECONDARY_SELECTED** highlights the text using underlining.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X).

XmTextFieldSetInsertionPosition—A TextField function that sets the position of the insertion cursor

Synopsis

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

void XmTextFieldSetInsertionPosition (widget, position)
        Widget          widget;
        XmTextPosition  position;
```

Description

XmTextFieldSetInsertionPosition sets the insertion cursor position of the TextField widget. This routine also calls the widget's **XmNmotionVerifyCallback** callbacks if the insertion cursor position changes.

widget Specifies the TextField widget ID.

position Specifies the position of the insert cursor. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X).

XmTextFieldSetMaxLength—A TextField function that sets the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

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

void XmTextFieldSetMaxLength (widget, max_length)
        Widget      widget;
        int         max_length;
```

Description

XmTextFieldSetMaxLength sets the value of the current maximum allowable length of the text string in the TextField widget. The maximum allowable length prevents the user from entering a text string from the keyboard that is larger than this limit. Strings that are entered using the **XmNvalue** (or **XmNvalueWcs**) resource, or the **XmTextFieldSetString** (or **XmTextFieldSetStringWcs**) function ignore this resource.

widget Specifies the TextField widget ID

max_length Specifies the maximum allowable length of the text string

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmText(3X), **XmTextFieldSetString(3X)**, and **XmTextFieldSetStringWcs(3X)**.

XmTextFieldSetSelection—A TextField function that sets the primary selection of the text

Synopsis

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

void XmTextFieldSetSelection (widget, first, last, time)
        Widget          widget;
        XmTextPosition  first;
        XmTextPosition  last;
        Time            time;
```

Description

XmTextFieldSetSelection sets the primary selection of the text in the widget. It also sets the insertion cursor position to the last position of the selection and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the TextField widget ID.

first Marks the first character position of the text to be selected.

last Marks the last position of the text to be selected.

time Specifies the time at which the selection value is desired. This should be the same as the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X).

XmTextFieldSetString—A TextField function that sets the string value

Synopsis

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

void XmTextFieldSetString (widget, value)
        Widget      widget;
        char        * value;
```

Description

XmTextFieldSetString sets the string value of the TextField widget. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. It also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the TextField widget ID

value Specifies the character pointer to the string value and places the string into the text edit window

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X) and **XmTextFieldSetStringWcs(3X)**.

XmTextFieldSetStringWcs—A TextField function that sets a wide character string value

Synopsis

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

void XmTextFieldSetStringWcs (widget, wcstring)
        Widget      widget;
        wchar_t     *wcstring;
```

Description

XmTextFieldSetStringWcs sets the wide character string value of the TextField widget. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. It also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the TextField widget ID

wcstring Specifies the wide character string value and places the string into the text edit window

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X) and **XmTextFieldSetString(3X)**.

XmTextFieldShowPosition—A TextField function that forces text at a given position to be displayed

Synopsis

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

void XmTextFieldShowPosition (widget, position)
        Widget          widget;
        XmTextPositionposition;
```

Description

XmTextFieldShowPosition forces text at the specified position to be displayed. If the **XmNautoShowCursorPosition** resource is True, the application should also set the insert cursor to this position.

widget Specifies the TextField widget ID.

position Specifies the character position to be displayed. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Related Information

XmTextField(3X).

XmTextFieldXYToPos—A TextField function that accesses the character position nearest an *x* and *y* position

Synopsis

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

XmTextPosition XmTextFieldXYToPos (widget, x, y)
        Widget      widget;
        Position    x;
        Position    y;
```

Description

XmTextFieldXYToPos accesses the character position nearest to the specified *x* and *y* position, relative to the upper left corner of the TextField widget.

widget Specifies the TextField widget ID

x Specifies the *x* position, relative to the upper left corner of the widget

y Specifies the *y* position, relative to the upper left corner of the widget

For a complete definition of TextField and its associated resources, see **XmTextField(3X)**.

Return Value

Returns the character position in the text nearest the *x* and *y* position specified. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

Related Information

XmTextField(3X).

XmTextFindString—A Text function that finds the beginning position of a text string

Synopsis

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

Boolean XmTextFindString (widget, start, string, direction, position)
        Widget          widget;
        XmTextPosition  start;
        char            *string;
        XmTextDirection direction;
        XmTextPosition  *position;
```

Description

XmTextFindString locates the beginning position of a specified text string. This routine searches forward or backward for the first occurrence of the string starting from the given start position. If it finds a match, the function returns the position of the first character of the string in *position*.

widget Specifies the Text widget ID.

start Specifies the character position from which the search proceeds. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

string Specifies the search string.

direction Indicates the search direction. It is relative to the primary direction of the text. The possible values are

XmTEXT_FORWARD
The search proceeds toward the end of the text buffer.

XmTEXT_BACKWARD
The search proceeds toward the beginning of the text buffer.

position Specifies the pointer in which the first character position of the string match is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero). If the function returns False, this value is undefined.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns True if a string match is found; otherwise, returns False.

Related Information

XmText(3X) and **XmTextFindStringWcs(3X)**.

XmTextFindStringWcs—A Text function that finds the beginning position of a wide character text string

Synopsis

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

Boolean XmTextFindStringWcs (widget, start, wcstring, direction, position)
        Widget          widget;
        XmTextPosition  start;
        wchar_t         *wcstring;
        XmTextDirection direction;
        XmTextPosition  *position;
```

Description

XmTextFindStringWcs locates the beginning position of a specified wide character text string. This routine searches forward or backward for the first occurrence of the string, starting from the given start position. If a match is found, the function returns the position of the first character of the string in *position*.

widget Specifies the Text widget ID.

start Specifies the character position from which the search proceeds. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

wcstring Specifies the wide character search string.

direction Indicates the search direction. It is relative to the primary direction of the text. The possible values are

XmTEXT_FORWARD
The search proceeds toward the end of the buffer.

XmTEXT_BACKWARD
The search proceeds toward the beginning of the buffer.

position Specifies the pointer in which the first character position of the string match is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero). If the function returns False, this value is undefined.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns True if a string match is found; otherwise, returns False.

Related Information

XmText(3X) and **XmTextFindString(3X)**.

XmTextGetBaseline—A Text function that accesses the *x* position of the first baseline

Synopsis

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

int XmTextGetBaseline (widget)
        Widget          widget;
```

Description

XmTextGetBaseline accesses the *x* position of the first baseline in the Text widget, relative to the *x* position of the top of the widget.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns an integer value that indicates the *x* position of the first baseline in the Text widget. The calculation takes into account the margin height, shadow thickness, highlight thickness, and font ascent of the first font in the fontlist. In this calculation the *x* position of the top of the widget is 0 (zero).

Related Information

XmText(3X).

XmTextGetEditable—A Text function that accesses the edit permission state

Synopsis

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

Boolean XmTextGetEditable (widget)
        Widget          widget;
```

Description

XmTextGetEditable accesses the edit permission state of the Text widget.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns a Boolean value that indicates the state of the **XmNeditable** resource.

Related Information

XmText(3X).

XmTextGetInsertionPosition—A Text function that accesses the position of the insert cursor

Synopsis

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

XmTextPosition XmTextGetInsertionPosition (widget)
        Widget      widget;
```

Description

XmTextGetInsertionPosition accesses the insertion cursor position of the Text widget.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns an **XmTextPosition** value that indicates the state of the **XmNcursorPosition** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

Related Information

XmText(3X).

XmTextGetLastPosition—A Text function that accesses the last position in the text

Synopsis

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

XmTextPosition XmTextGetLastPosition (widget)
        Widget          widget;
```

Description

XmTextGetLastPosition accesses the last position in the text buffer of the Text widget. This is an integer number of characters from the beginning of the buffer, and represents the position following which text that is added to the end of the buffer is placed. The first character position is 0 (zero). The last character position is equal to the number of characters in the text buffer.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns an **XmTextPosition** value that indicates the last position in the text buffer.

Related Information

XmText(3X).

XmTextGetMaxLength—A Text function that accesses the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

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

int XmTextGetMaxLength (widget)
        Widget          widget;
```

Description

XmTextGetMaxLength accesses the value of the current maximum allowable length of the text string in the Text widget entered from the keyboard. The maximum allowable length prevents the user from entering a text string larger than this limit.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns the integer value that indicates the string's maximum allowable length that can be entered from the keyboard.

Related Information

XmText(3X).

XmTextGetSelection—A Text function that retrieves the value of the primary selection

Synopsis

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

char * XmTextGetSelection (widget)
        Widget          widget;
```

Description

XmTextGetSelection retrieves the value of the primary selection. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns a character pointer to the string that is associated with the primary selection.

Related Information

XmText(3X) and **XmTextGetSelectionWcs(3X)**.

XmTextGetSelectionPosition—A Text function that accesses the position of the primary selection

Synopsis

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

Boolean XmTextGetSelectionPosition (widget, left, right)
        Widget          widget;
        XmTextPosition *left;
        XmTextPosition *right;
```

Description

XmTextGetSelectionPosition accesses the left and right position of the primary selection in the text buffer of the Text widget.

widget Specifies the Text widget ID

left Specifies the pointer in which the position of the left boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

right Specifies the pointer in which the position of the right boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

This function returns True if the widget owns the primary selection; otherwise, it returns False.

Related Information

XmText(3X).

XmTextGetSelectionWcs—A Text function that retrieves the value of a wide character encoded primary selection

Synopsis

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

wchar_t * XmTextGetSelectionWcs (widget)
        Widget          widget;
```

Description

XmTextGetSelectionWcs retrieves the value of the primary selection that is encoded in a wide character format. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the wide character buffer by calling **XtFree**.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns the wide character string that is associated with the primary selection in the Text widget.

Related Information

XmText(3X) and **XmTextGetSelection(3X)**.

XmTextGetSource—A Text function that accesses the source of the widget

Synopsis

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

XmTextSource XmTextGetSource (widget)
        Widget          widget;
```

Description

XmTextGetSource accesses the source of the Text widget. Text widgets can share sources of text so that editing in one widget is reflected in another. This function accesses the source of one widget so that it can be made the source of another widget, using the function **XmTextSetSource(3X)**.

Setting a new text source destroys the old text source if no other Text widgets are using that source. To replace a text source but keep it for later use, create an unmanaged Text widget and set its source to the text source you want to keep.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns an **XmTextSource** value that represents the source of the Text widget.

Related Information

XmText(3X).

XmTextGetString—A Text function that accesses the string value

Synopsis

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

char * XmTextGetString (widget)
        Widget          widget;
```

Description

XmTextGetString accesses the string value of the Text widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns a character pointer to the string value of the text widget. Returns an empty string if the length of the Text widget's string is 0 (zero).

Related Information

XmText(3X) and **XmTextGetStringWcs(3X)**.

XmTextGetStringWcs—A Text function that retrieves a copy of the wide character string value of a Text widget

Synopsis

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

wchar_t * XmTextGetStringWcs (widget)
        Widget          widget;
```

Description

XmTextGetStringWcs retrieves a copy of the wide character string value of the Text widget. The application is responsible for freeing the storage associated with the string by calling **XtFree**.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns the wide character string value of the Text widget. The function returns an empty string if the length of the Text widget's string is 0 (zero).

Related Information

XmText(3X) and **XmTextGetString(3X)**.

XmTextGetSubstring—A Text function that retrieves a copy of a portion of the internal text buffer

Synopsis

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

int XmTextGetSubstring (widget, start, num_chars, buffer_size, buffer)
        Widget          widget;
        XmTextPosition  start;
        int             num_chars;
        int             buffer_size;
        char            *buffer;
```

Description

XmTextGetSubstring retrieves a copy of a portion of the internal text buffer of a Text widget. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

The size of the required buffer depends on the maximum number of bytes per character (**MB_CUR_MAX**) for the current locale. **MB_CUR_MAX** is a macro defined in **stdlib.h**. The buffer should be large enough to contain the substring to be copied and a NULL terminator. Use the following equation to calculate the size of buffer the application should provide:

buffer_size = (*num_chars* * **MB_CUR_MAX**) + **1**

widget Specifies the Text widget ID.

start Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

num_chars Specifies the number of characters to be copied into the provided buffer.

buffer_size Specifies the size of the supplied buffer in bytes. This size should account for a NULL terminator.

buffer Specifies the character buffer into which the internal text buffer will be copied.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

XmCOPY_SUCCEEDED
The function was successful.

XmCOPY_FAILED
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget's buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmText(3X) and **XmTextGetSubstringWcs(3X)**.

XmTextGetSubstringWcs—A Text function that retrieves a portion of a wide character internal text buffer

Synopsis

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

int XmTextGetSubstringWcs (widget, start, num_chars, buffer_size, buffer)
        Widget          widget;
        XmTextPosition  start;
        int             num_chars;
        int             buffer_size;
        wchar_t         *buffer;
```

Description

XmTextGetSubstringWcs retrieves a copy of a portion of the internal text buffer of a Text widget that is stored in a wide character format. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

widget Specifies the Text widget ID.

start Specifies the beginning character position from which the data will be retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

num_chars Specifies the number of **wchar_t** characters to be copied into the provided buffer.

buffer_size Specifies the size of the supplied buffer as a number of **wchar_t** storage locations. The minimum size is *num_chars* + 1.

buffer Specifies the wide character buffer into which the internal text buffer will be copied.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

XmCOPY_SUCCEEDED
The function was successful.

XmCOPY_FAILED
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

XmCOPY_TRUNCATED
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget's buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

Related Information

XmText(3X) and **XmTextGetSubstring(3X)**.

XmTextGetTopCharacter—A Text function that accesses the position of the first character displayed

Synopsis

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

XmTextPosition XmTextGetTopCharacter (widget)
        Widget          widget;
```

Description

XmTextGetTopCharacter accesses the position of the text at the top of the Text widget.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns an **XmTextPosition** value that indicates the state of the **XmNtopCharacter** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

Related Information

XmText(3X).

XmTextInsert—A Text function that inserts a character string into a text string

Synopsis

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

void XmTextInsert(widget, position, value)
        Widget          widget;
        XmTextPosition  position;
        char            * value;
```

Description

XmTextInsert inserts a character string into the text string in the Text widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the parameter *position* must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the Text widget ID.

position Specifies the position in the text string where the character string is to be inserted.

value Specifies the character string value to be added to the text widget.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X) and **XmTextInsertWcs(3X)**.

XmTextInsertWcs—A Text function that inserts a wide character string into a Text widget

Synopsis

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

void XmTextInsertWcs (widget, position, wcstring)
        Widget          widget;
        XmTextPosition  position;
        wchar_t         *wcstring;
```

Description

XmTextInsertWcs inserts a wide character string into the Text widget at a specified location. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the *position* parameter must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the Text widget ID

position Specifies the position in the text string where the new character string is to be inserted

wcstring Specifies the wide character string value to be added to the Text widget

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X) and **XmTextInsert(3X)**.

XmTextPaste—A Text function that inserts the clipboard selection

Synopsis

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

Boolean XmTextPaste (widget)
        Widget      widget;
```

Description

XmTextPaste inserts the clipboard selection at the insertion cursor of the destination widget. If **XmNpendingDelete** is True and the insertion cursor is inside the current selection, the clipboard selection replaces the selected text.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the Text widget ID

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

This function returns False if the *widget* does not own the primary selection. Otherwise, it returns True.

Related Information

XmText(3X).

XmTextPosToXY—A Text function that accesses the *x* and *y* position of a character position

Synopsis

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

Boolean XmTextPosToXY (widget, position, x, y)
        Widget          widget;
        XmTextPosition  position;
        Position        *x;
        Position        *y;
```

Description

XmTextPosToXY accesses the *x* and *y* position, relative to the upper left corner of the Text widget, of a given character position in the text buffer.

widget Specifies the Text widget ID

position Specifies the character position in the text for which the *x* and *y* position is accessed. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

x Specifies the pointer in which the *x* position, relative to the upper left corner of the widget, is returned. This value is meaningful only if the function returns True.

y Specifies the pointer in which the *y* position, relative to the upper left corner of the widget, is returned. This value is meaningful only if the function returns True.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

This function returns True if the character position is displayed in the Text widget; otherwise, it returns False, and no *x* or *y* value is returned.

Related Information

XmText(3X).

XmTextPosition—Data type for a character position within a text string

Synopsis

#include <Xm/Xm.h>

Description

XmTextPosition is the data type for a character position within a text string. The text position is an integer representing the number of characters from the beginning of the string. The first character position in the string is 0 (zero).

Related Information

XmText(3X).

XmTextRemove—A Text function that deletes the primary selection

Synopsis

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

Boolean XmTextRemove (widget)
        Widget      widget;
```

Description

XmTextRemove deletes the primary selected text. If there is a selection, this routine also calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

This function returns False if the primary selection is NULL or if the *widget* does not own the primary selection. Otherwise, it returns True.

Related Information

XmText(3X).

XmTextReplace—A Text function that replaces part of a text string

Synopsis

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

void XmTextReplace (widget, from_pos, to_pos, value)
        Widget          widget;
        XmTextPosition  from_pos;
        XmTextPosition  to_pos;
        char            *value;
```

Description

XmTextReplace replaces part of the text string in the Text widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the parameter *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both parameters, *from_pos* and *to_pos*, must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the Text widget ID

from_pos Specifies the start position of the text to be replaced

to_pos Specifies the end position of the text to be replaced

value Specifies the character string value to be added to the text widget

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X) and **XmTextReplaceWcs(3X)**.

XmTextReplaceWcs—A Text function that replaces part of a wide character string in a Text widget

Synopsis

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

void XmTextReplaceWcs (widget, from_pos, to_pos, wcstring)
        Widget          widget;
        XmTextPosition  from_pos;
        XmTextPosition  to_pos;
        wchar_t         *wcstring;
```

Description

XmTextReplaceWcs replaces part of the wide character string in the Text widget. The character positions begin at zero and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the *from_pos* parameter must be 1 and the *to_pos* parameter must be 3. To insert a string after the fourth character, both the *from_pos* and *to_pos* parameters must be 4.

This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks.

widget Specifies the Text widget ID

from_pos Specifies the start position of the text to be replaced

to_pos Specifies the end position of the text to be replaced

wcstring Specifies the wide character string value to be added to the Text widget

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X) and **XmTextReplace(3X)**.

XmTextScroll—A Text function that scrolls text

Synopsis

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

void XmTextScroll (widget, lines)
        Widget          widget;
        int             lines;
```

Description

XmTextScroll scrolls text in a Text widget.

widget Specifies the Text widget ID

lines Specifies the number of lines of text to scroll. A positive value causes text to scroll upward; a negative value causes text to scroll downward.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextSetAddMode—A Text function that sets the state of Add mode

Synopsis

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

void XmTextSetAddMode (widget, state)
        Widget      widget;
        Boolean     state;
```

Description

XmTextSetAddMode controls whether or not the Text widget is in Add mode. When the widget is in Add mode, the insert cursor can be moved without disturbing the primary selection.

widget Specifies the Text widget ID

state Specifies whether or not the widget is in Add mode. A value of True turns on Add mode; a value of False turns off Add mode.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextSetEditable—A Text function that sets the edit permission

Synopsis

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

void XmTextSetEditable (widget, editable)
        Widget      widget;
        Boolean     editable;
```

Description

XmTextSetEditable sets the edit permission state of the Text widget. When set to True, the text string can be edited.

widget Specifies the Text widget ID

editable Specifies a Boolean value that when True allows text string edits

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextSetHighlight—A Text function that highlights text

Synopsis

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

void XmTextSetHighlight (widget, left, right, mode)
        Widget          widget;
        XmTextPosition  left;
        XmTextPosition  right;
        XmHighlightMode mode;
```

Description

XmTextSetHighlight highlights text between the two specified character positions. The *mode* parameter determines the type of highlighting. Highlighting text merely changes the visual appearance of the text; it does not set the selection.

widget Specifies the Text widget ID

left Specifies the position of the left boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

right Specifies the position of the right boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

mode Specifies the type of highlighting to be done. A value of **XmHIGHLIGHT_NORMAL** removes highlighting. A value of **XmHIGHLIGHT_SELECTED** highlights the text using reverse video. A value of **XmHIGHLIGHT_SECONDARY_SELECTED** highlights the text using underlining.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextSetInsertionPosition—A Text function that sets the position of the insert cursor

Synopsis

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

void XmTextSetInsertionPosition (widget, position)
        Widget          widget;
        XmTextPositionposition;
```

Description

XmTextSetInsertionPosition sets the insertion cursor position of the Text widget. This routine also calls the widget's **XmNmotionVerifyCallback** callbacks if the insertion cursor position changes.

widget Specifies the Text widget ID

position Specifies the position of the insertion cursor. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextSetMaxLength—A Text function that sets the value of the current maximum allowable length of a text string entered from the keyboard

Synopsis

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

void XmTextSetMaxLength (widget, max_length)
        Widget      widget;
        int         max_length;
```

Description

XmTextSetMaxLength sets the value of the current maximum allowable length of the text string in the Text widget. The maximum allowable length prevents the user from entering a text string from the keyboard that is larger than this limit. Strings that are entered using the **XmNvalue** (or **XmNvalueWcs**) resource, or the **XmTextSetString** (or **XmTextSetStringWcs**) function ignore this resource.

widget Specifies the Text widget ID

max_length Specifies the maximum allowable length of the text string

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X), **XmTextSetString(3X)**, and **XmTextSetStringWcs(3X)**.

XmTextSetSelection—A Text function that sets the primary selection of the text

Synopsis

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

void XmTextSetSelection (widget, first, last, time)
        Widget          widget;
        XmTextPosition  first;
        XmTextPosition  last;
        Time            time;
```

Description

XmTextSetSelection sets the primary selection of the text in the widget. It also sets the insertion cursor position to the last position of the selection and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget	Specifies the Text widget ID
first	Marks the first character position of the text to be selected
last	Marks the last position of the text to be selected
time	Specifies the time at which the selection value is desired. This should be the same as the time of the event that triggered this request.

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextSetSource—A Text function that sets the source of the widget

Synopsis

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

void XmTextSetSource (widget, source, top_character, cursor_position)
        Widget          widget;
        XmTextSource    source;
        XmTextPosition  top_character;
        XmTextPosition  cursor_position;
```

Description

XmTextSetSource sets the source of the Text widget. Text widgets can share sources of text so that editing in one widget is reflected in another. This function sets the source of one widget so that it can share the source of another widget.

Setting a new text source destroys the old text source if no other Text widgets are using that source. To replace a text source but keep it for later use, create an unmanaged Text widget and set its source to the text source you want to keep.

widget Specifies the Text widget ID.

source Specifies the source with which the widget displays text. This can be a value returned by the **XmTextGetSource(3X)** function. If no source is specified, the widget creates a default string source.

top_character Specifies the position in the text to display at the top of the widget. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

cursor_position
Specifies the position in the text at which the insert cursor is located. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextSetString—A Text function that sets the string value

Synopsis

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

void XmTextSetString (widget, value)
        Widget      widget;
        char        * value;
```

Description

XmTextSetString sets the string value of the Text widget. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. This function also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the Text widget ID

value Specifies the character pointer to the string value and places the string into the text edit window

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X) and **XmTextSetStringWcs(3X)**.

XmTextSetStringWcs—A Text function that sets a wide character string value

Synopsis

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

void XmTextSetStringWcs (widget, wcstring)
        Widget      widget;
        wchar_t     *wcstring;
```

Description

XmTextSetStringWcs sets the wide character string value of the Text widget. This routine calls the widget's **XmNvalueChangedCallback** and verification callbacks, either **XmNmodifyVerifyCallback** or **XmNmodifyVerifyCallbackWcs**, or both. If both verification callback lists are registered, the procedures of the **XmNmodifyVerifyCallback** list are executed first and the resulting data is passed to the **XmNmodifyVerifyCallbackWcs** callbacks. This function also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the Text widget ID

value Specifies the wide character string value and places the string into the text edit window

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X) and **XmTextSetString(3X)**.

XmTextSetTopCharacter—A Text function that sets the position of the first character displayed

Synopsis

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

void XmTextSetTopCharacter (widget, top_character)
        Widget          widget;
        XmTextPosition  top_character;
```

Description

XmTextSetTopCharacter sets the position of the text at the top of the Text widget. If the **XmNeditMode** is **XmMULTI_LINE_EDIT**, the line of text that contains *top_character* is displayed at the top of the widget without the text shifting left or right.

widget Specifies the Text widget ID

top_character Specifies the position in the text to display at the top of the widget. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextShowPosition—A Text function that forces text at a given position to be displayed

Synopsis

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

void XmTextShowPosition (widget, position)
        Widget          widget;
        XmTextPositionposition;
```

Description

XmTextShowPosition forces text at the specified position to be displayed. If the **XmNautoShowCursorPosition** resource is True, the application should also set the insert cursor to this position.

widget Specifies the Text widget ID

position Specifies the character position to be displayed. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Related Information

XmText(3X).

XmTextXYToPos—A Text function that accesses the character position nearest an *x* and *y* position

Synopsis

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

XmTextPosition XmTextXYToPos (widget, x, y)
        Widget      widget;
        Position    x;
        Position    y;
```

Description

XmTextXYToPos accesses the character position nearest to the specified *x* and *y* position, relative to the upper left corner of the Text widget.

widget Specifies the Text widget ID

x Specifies the *x* position, relative to the upper left corner of the widget

y Specifies the *y* position, relative to the upper left corner of the widget

For a complete definition of Text and its associated resources, see **XmText(3X)**.

Return Value

Returns the character position in the text nearest the *x* and *y* position specified. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

Related Information

XmText(3X).

XmToggleButton—The ToggleButton widget class

Synopsis

#include <Xm/ToggleB.h>

Description

ToggleButton sets nontransitory state data within an application. Usually this widget consists of an indicator (square or diamond) with either text or a pixmap on one side of it. However, it can also consist of just text or a pixmap without the indicator.

The toggle graphics display a **1-of-many** or **N-of-many** selection state. When a toggle indicator is displayed, a square indicator shows an **N-of-many** selection state and a diamond indicator shows a **1-of-many** selection state.

ToggleButton implies a selected or unselected state. In the case of a label and an indicator, an empty indicator (square or diamond shaped) indicates that ToggleButton is unselected, and a filled indicator shows that it is selected. In the case of a pixmap toggle, different pixmaps are used to display the selected/unselected states.

The default behavior associated with a ToggleButton in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the ToggleButton. In addition, **BMenu** controls the behavior of the ToggleButton if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Label's resource **XmNmarginLeft** may be increased to accommodate the toggle indicator when it is created.

Classes

ToggleButton inherits behavior and resources from **Core**, **XmPrimitive**, and **XmLabel**.

The class pointer is **xmToggleButtonWidgetClass**.

The class name is **XmToggleButton**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmToggleButton Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNarmCallback XmCArmCallback	NULL XtCallbackList	C
XmNdisarmCallback XmCDisarmCallback	NULL XtCallbackList	C
XmNfillOnSelect XmCFillOnSelect	dynamic Boolean	CSG
XmNindicatorOn XmCIndicatorOn	True Boolean	CSG
XmNindicatorSize XmCIndicatorSize	dynamic Dimension	CSG
XmNindicatorType XmCIndicatorType	dynamic unsigned char	CSG
XmNselectColor XmCSelectColor	dynamic Pixel	CSG
XmNselectInsensitivePixmap XmCSelectInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNselectPixmap XmCSelectPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNset XmCSet	False Boolean	CSG
XmNspacing XmCSpacing	4 Dimension	CSG
XmNvalueChangedCallback XmCValueChangedCallback	NULL XtCallbackList	C
XmNvisibleWhenOff XmCVisibleWhenOff	dynamic Boolean	CSG

XmNarmCallback

Specifies the list of callbacks called when the ToggleButton is armed. To arm this widget, press the active mouse button while the pointer is inside the ToggleButton. For this callback, the reason is **XmCR_ARM**.

XmNdisarmCallback

Specifies the list of callbacks called when ToggleButton is disarmed. To disarm this widget, press and release the active mouse button while the pointer is inside the ToggleButton. This widget is also disarmed when the user moves out of the widget and releases the mouse button when the pointer is outside the widget. For this callback, the reason is **XmCR_DISARM**.

XmNfillOnSelect

Fills the indicator with the color specified in **XmNselectColor** and switches the top and bottom shadow colors when set to True. Otherwise, it switches only the top and bottom shadow colors. The default is set to the value of **XmNindicatorOn**. When **XmNindicatorOn** is False, and **XmNfillOnSelect** is set explicitly to True, the background is filled with the color specified by **XmNselectColor**.

XmNindicatorOn

Specifies that a toggle indicator is drawn to one side of the toggle text or pixmap when set to True. When set to False, no space is allocated for the indicator, and it is not displayed. If **XmNindicatorOn** is True, the indicator shadows are switched when the button is selected or unselected, but, any shadows around the entire widget are not switched. However, if **XmNindicatorOn** is False, any shadows around the entire widget are switched when the toggle is selected or unselected.

XmNindicatorSize

Sets the size of the indicator. If no value is specified, the size of the indicator is based on the size of the label string or pixmap. If the label string or pixmap changes, the size of the indicator is recomputed based on the size of the label string or pixmap. Once a value has been specified for **XmNindicatorSize**, the indicator has that size, regardless of the size of the label string or pixmap, until a new value is specified.

XmNindicatorType

Specifies if the indicator is a **1-of** or **N-of** indicator. For the **1-of** indicator, the value is **XmONE_OF_MANY**. For the **N-of** indicator, the value is **XmN_OF_MANY**. The **N-of-many** indicator is square. The **1-of-many** indicator is diamond shaped. This resource specifies only the visuals and does not enforce the behavior. When the ToggleButton is in a RadioBox, the default is **XmONE_OF_MANY**; otherwise, the default is **XmN_OF_MANY**.

XmNselectColor

Allows the application to specify what color fills the center of the square or diamond-shaped indicator when it is set. If this color is the same as either the top or the bottom shadow color of the indicator, a one-pixel-wide margin is left between the shadows and the fill; otherwise, it is filled completely. This resource's default for a color display is a color between the background and the bottom shadow color. For a monochrome display, the default is set to the foreground color. To set the background of the button to **XmNselectColor** when **XmNindicatorOn** is False, the value of **XmNfillOnSelect** must be explicitly set to True.

XmNselectInsensitivePixmap

Specifies a pixmap used as the button face when the ToggleButton is selected, the button is insensitive, and the Label resource **XmNlabelType** is set to **XmPIXMAP**. If the ToggleButton is unselected and the button is insensitive, the pixmap in **XmNlabelInsensitivePixmap** is used as the button face. If no value is specified for **XmNlabelInsensitivePixmap**, that resource is set to the value specified for **XmNselectInsensitivePixmap**.

XmNselectPixmap

Specifies the pixmap to be used as the button face when **XmNlabelType** is **XmPIXMAP** and the ToggleButton is selected. When the ToggleButton is unselected, the pixmap specified in the Label's **XmNlabelPixmap** is used. If no value is specified for **XmNlabelPixmap**, that resource is set to the value specified for **XmNselectPixmap**.

XmNset Represents the state of the ToggleButton. A value of False indicates that the ToggleButton is not set. A value of True indicates that the ToggleButton is set. Setting this resource sets the state of the ToggleButton.

XmNspacing Specifies the amount of spacing between the toggle indicator and the toggle label (text or pixmap).

XmNvalueChangedCallback

Specifies the list of callbacks called when the ToggleButton value is changed. To change the value, press and release the active mouse button while the pointer is inside the ToggleButton. This action also causes this widget to be disarmed. For this callback, the reason is **XmCR_VALUE_CHANGED**.

XmNvisibleWhenOff

Indicates that the toggle indicator is visible in the unselected state when the Boolean value is True. When the ToggleButton is in a menu, the default value is False. When the ToggleButton is in a RadioBox, the default value is True.

Inherited Resources

ToggleButton inherits behavior and resources from the superclasses in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmLabel Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	CSG
XmNacceleratorText XmCAcceleratorText	NULL XmString	CSG
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	dynamic Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	dynamic Dimension	CSG
XmNmarginRight XmCMarginRight	0 Dimension	CSG
XmNmarginTop XmCMarginTop	dynamic Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG
XmNmnemonic XmCMnemonic	NULL KeySym	CSG

Name Class	Default Type	Access
XmNmnemonicCharSet XmCMnemonicCharSet	XmFONTLIST_DEFAULT_TAG String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmPrimitive Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

Core Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	CSG
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	0 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen *	CG
XmNsensitive XmCSensitive	True Boolean	CSG

Name Class	Default Type	Access
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
    int             set;
} XmToggleButtonCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

set Reflects the ToggleButton's current state when the callback occurred, either True (selected) or False (unselected)

Translations

XmToggleButton includes translations from **Primitive**. Additional **XmToggleButton** translations for buttons not in a menu system are described in the following list. These translations may not directly correspond to a translation table.

Note that altering translations in **#override** or **#augment** mode is undefined.

BTransferPress:	**ProcessDrag()**
BSelect Press:	**Arm()**
BSelect Release:	**Select()** **Disarm()**
KHelp:	**Help()**
KSelect:	**ArmAndActivate()**

XmToggleButton inherits menu traversal translations from **XmLabel**. Additional **XmToggleButton** translations for **ToggleButtons** in a menu system are described in the following list. In a Popup menu system, **BMenu** also performs the **BSelect** actions. These translations may not directly correspond to a translation table.

BSelect Press:	**BtnDown()**
BSelect Release:	**BtnUp()**
KHelp:	**Help()**
KActivate:	**ArmAndActivate()**
KSelect:	**ArmAndActivate()**
MAny KCancel:	**MenuShellPopdownOne()**

Action Routines

The XmToggleButton action routines are

Arm(): If the button was previously unset, this action does the following: if **XmNindicatorOn** is True, it draws the indicator shadow so that the indicator looks pressed; if **XmNfillOnSelect** is True, it fills the indicator with the color specified by **XmNselectColor**. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks pressed. If **XmNlabelType** is **XmPIXMAP**, the **XmNselectPixmap** is used as the button face. This action calls the **XmNarmCallback** callbacks.

If the button was previously set, this action does the following: if both **XmNindicatorOn** and **XmNvisibleWhenOff** are True, it draws the indicator shadow so that the indicator looks raised; if **XmNfillOnSelect** is True, it fills the indicator with the background

color. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks raised. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used as the button face. This action calls the **XmNarmCallback** callbacks.

ArmAndActivate():
If the ToggleButton was previously set, unsets it; if the ToggleButton was previously unset, sets it.

In a menu, this action unposts all menus in the menu hierarchy. Unless the button is already armed, it calls the **XmNarmCallback** callbacks. This action calls the **XmNvalueChangedCallback** and **XmNdisarmCallback** callbacks.

Outside a menu, if the button was previously unset, this action does the following: if **XmNindicatorOn** is True, it draws the indicator shadow so that the indicator looks pressed; if **XmNfillOnSelect** is True, it fills the indicator with the color specified by **XmNselectColor**. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks pressed. If **XmNlabelType** is **XmPIXMAP**, the **XmNselectPixmap** is used as the button face. This action calls the **XmNarmCallback**, **XmNvalueChangedCallback**, and **XmNdisarmCallback** callbacks.

Outside a menu, if the button was previously set, this action does the following: if both **XmNindicatorOn** and **XmNvisibleWhenOff** are True, it draws the indicator shadow so that the indicator looks raised; if **XmNfillOnSelect** is True, it fills the indicator with the background color. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks raised. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used as the button face. This action calls the **XmNarmCallback**, **XmNvalueChangedCallback**, and **XmNdisarmCallback** callbacks.

BtnDown(): This action unposts any menus posted by the ToggleButton's parent menu, disables keyboard traversal for the menu, and enables mouse traversal for the menu. It draws the shadow in the armed state and, unless the button is already armed, calls the **XmNarmCallback** callbacks.

BtnUp(): This action unposts all menus in the menu hierarchy. If the ToggleButton was previously set, unsets it; if the ToggleButton was previously unset, sets it. It calls the **XmNvalueChangedCallback** callbacks and then the **XmNdisarmCallback** callbacks.

Disarm(): Calls the callbacks for **XmNdisarmCallback**.

Help(): In a Pulldown or Popup MenuPane, unposts all menus in the menu hierarchy and restores keyboard focus to the widget that had the focus before the menu system was entered. Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MenuShellPopdownOne():
In a top-level Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu.

In a Popup MenuPane, unposts the menu and restores keyboard focus to the widget from which the menu was posted.

ProcessDrag():
Drags the contents of a ToggleButton label, identified when **BTransfer** is pressed. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT** for a label type of **XmSTRING**, or **PIXMAP** if the label type is **XmPIXMAP**. This action is undefined for ToggleButtons used in a menu system.

Select(): If the pointer is within the button, takes the following actions: If the button was previously unset, sets it; if the button was previously set, unsets it. This action calls the **XmNvalueChangedCallback** callbacks.

Additional Behavior

This widget has the following additional behavior:

<EnterWindow>:

In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the armed state and calls the **XmNarmCallback** callbacks.

If the ToggleButton is not in a menu and the cursor leaves and then reenters the ToggleButton's window while the button is pressed, this action restores the button's armed appearance.

<LeaveWindow>:

In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the unarmed state and calls the **XmNdisarmCallback** callbacks.

If the ToggleButton is not in a menu and the cursor leaves the ToggleButton's window while the button is pressed, this action restores the button's unarmed appearance.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Core(3X), **XmCreateRadioBox(3X)**, **XmCreateToggleButton(3X)**, **XmLabel(3X)**, **XmPrimitive(3X)**, **XmRowColumn(3X)**, **XmToggleButtonGetState(3X)**, and **XmToggleButtonSetState(3X)**.

XmToggleButtonGadget—The ToggleButtonGadget widget class

Synopsis

#include <Xm/ToggleBG.h>

Description

ToggleButtonGadget sets nontransitory state data within an application. Usually this gadget consists of an indicator (square or diamond-shaped) with either text or a pixmap on one side of it. However, it can also consist of just text or a pixmap without the indicator.

The toggle graphics display a **1-of-many** or **N-of-many** selection state. When a toggle indicator is displayed, a square indicator shows an **N-of-many** selection state and a diamond-shaped indicator shows a **1-of-many** selection state.

ToggleButtonGadget implies a selected or unselected state. In the case of a label and an indicator, an empty indicator (square or diamond-shaped) indicates that ToggleButtonGadget is unselected, and a filled indicator shows that it is selected. In the case of a pixmap toggle, different pixmaps are used to display the selected/unselected states.

The default behavior associated with a ToggleButtonGadget in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the ToggleButtonGadget. In addition, **BMenu** controls the behavior of the ToggleButtonGadget if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Label's resource **XmNmarginLeft** may be increased to accommodate the toggle indicator when it is created.

Classes

ToggleButtonGadget inherits behavior and resources from **Object**, **RectObj**, **XmGadget** and **XmLabelGadget**.

The class pointer is **xmToggleButtonGadgetClass**.

The class name is **XmToggleButtonGadget**.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmToggleButtonGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNarmCallback XmCArmCallback	NULL XtCallbackList	C
XmNdisarmCallback XmCDisarmCallback	NULL XtCallbackList	C
XmNfillOnSelect XmCFillOnSelect	dynamic Boolean	CSG
XmNindicatorOn XmCIndicatorOn	True Boolean	CSG
XmNindicatorSize XmCIndicatorSize	dynamic Dimension	CSG
XmNindicatorType XmCIndicatorType	dynamic unsigned char	CSG
XmNselectColor XmCSelectColor	dynamic Pixel	CSG
XmNselectInsensitivePixmap XmCSelectInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNselectPixmap XmCSelectPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNset XmCSet	False Boolean	CSG
XmNspacing XmCSpacing	4 Dimension	CSG
XmNvalueChangedCallback XmCValueChangedCallback	NULL XtCallbackList	C
XmNvisibleWhenOff XmCVisibleWhenOff	dynamic Boolean	CSG

XmNarmCallback

Specifies a list of callbacks that is called when the ToggleButtonGadget is armed. To arm this gadget, press the active mouse button while the pointer is inside the ToggleButtonGadget. For this callback, the reason is **XmCR_ARM**.

XmNdisarmCallback

Specifies a list of callbacks called when ToggleButtonGadget is disarmed. To disarm this gadget, press and release the active mouse button while the pointer is inside the ToggleButtonGadget. The gadget is also disarmed when the user moves out of the gadget and releases the mouse button when the pointer is outside the gadget. For this callback, the reason is **XmCR_DISARM**.

XmNfillOnSelect

Fills the indicator with the color specified in **XmNselectColor** and switches the top and bottom shadow colors when set to True. Otherwise, it switches only the top and bottom shadow colors. The default is set to the value of **XmNindicatorOn**. When **XmNindicatorOn** is False, and **XmNfillOnSelect** is set explicitly to True, the background is filled with the color specified by **XmNselectColor**.

XmNindicatorOn

Specifies that a toggle indicator is drawn to one side of the toggle text or pixmap when set to True. When set to False, no space is allocated for the indicator, and it is not displayed. If **XmNindicatorOn** is True, the indicator shadows are switched when the button is selected or unselected, but any shadows around the entire gadget are not switched. However, if **XmNindicatorOn** is False, any shadows around the entire gadget are switched when the toggle is selected or unselected.

XmNindicatorSize

Sets the size of the indicator. If no value is specified, the size of the indicator is based on the size of the label string or pixmap. If the label string or pixmap changes, the size of the indicator is recomputed based on the size of the label string or pixmap. Once a value has been specified for **XmNindicatorSize**, the indicator has that size, regardless of the size of the label string or pixmap, until a new value is specified.

XmNindicatorType

Specifies if the indicator is a **1-of** or an **N-of** indicator. For the **1-of** indicator, the value is **XmONE_OF_MANY**. For the **N-of** indicator, the value is **XmN_OF_MANY**. The **N-of-many**

indicator is square. The **1-of-many** indicator is diamond-shaped. This resource specifies only the visuals and does not enforce the behavior. When the ToggleButtonGadget is in a RadioBox, the default is **XmONE_OF_MANY**; otherwise, the default is **XmN_OF_MANY**.

XmNselectColor
Allows the application to specify what color fills the center of the square or diamond-shaped indicator when it is set. If this color is the same as either the top or the bottom shadow color of the indicator, a one-pixel-wide margin is left between the shadows and the fill; otherwise, it is filled completely. This resource's default for a color display is a color between the background and the bottom shadow color. For a monochrome display, the default is set to the foreground color. To set the background of the button to **XmNselectColor** when **XmNindicatorOn** is False, the value of **XmNfillOnSelect** must be explicitly set to True.

XmNselectInsensitivePixmap
Specifies a pixmap used as the button face when the ToggleButtonGadget is selected, the button is insensitive, and the LabelGadget resource **XmNlabelType** is **XmPIXMAP**. If the ToggleButtonGadget is unselected and the button is insensitive, the pixmap in **XmNlabelInsensitivePixmap** is used as the button face. If no value is specified for **XmNlabelInsensitivePixmap**, that resource is set to the value specified for **XmNselectInsensitivePixmap**.

XmNselectPixmap
Specifies the pixmap to be used as the button face if **XmNlabelType** is **XmPIXMAP** and the ToggleButtonGadget is selected. When the ToggleButtonGadget is unselected, the pixmap specified in LabelGadget's **XmNlabelPixmap** is used. If no value is specified for **XmNlabelPixmap**, that resource is set to the value specified for **XmNselectPixmap**.

XmNset Represents the state of the ToggleButton. A value of false indicates that the ToggleButton is not set. A value of true indicates that the ToggleButton is set. Setting this resource sets the state of the ToggleButton.

XmNspacing Specifies the amount of spacing between the toggle indicator and the toggle label (text or pixmap).

XmNvalueChangedCallback

Specifies a list of callbacks called when the ToggleButtonGadget value is changed. To change the value, press and release the active mouse button while the pointer is inside the ToggleButtonGadget. This action also causes the gadget to be disarmed. For this callback, the reason is **XmCR_VALUE_CHANGED**.

XmNvisibleWhenOff

Indicates that the toggle indicator is visible in the unselected state when the Boolean value is True. When the ToggleButtonGadget is in a menu, the default value is False. When the ToggleButtonGadget is in a RadioBox, the default value is True.

Inherited Resources

ToggleButtonGadget inherits behavior and resources from the superclasses described in the following tables. For a complete description of each resource, refer to the reference page for that superclass.

XmLabelGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNaccelerator XmCAccelerator	NULL String	CSG
XmNacceleratorText XmCAcceleratorText	NULL XmString	CSG
XmNalignment XmCAlignment	dynamic unsigned char	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginBottom XmCMarginBottom	dynamic Dimension	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginLeft XmCMarginLeft	dynamic Dimension	CSG
XmNmarginRight XmCMarginRight	0 Dimension	CSG
XmNmarginTop XmCMarginTop	dynamic Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG

Name Class	Default Type	Access
XmNmnemonic XmCMnemonic	NULL KeySym	CSG
XmNmnemonicCharSet XmCMnemonicCharSet	dynamic String	CSG
XmNrecomputeSize XmCRecomputeSize	True Boolean	CSG
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CSG

XmGadget Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	G
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	G
XmNhighlightOnEnter XmCHighlightOnEnter	False Boolean	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	dynamic Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	G
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

RectObj Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNborderWidth XmCBorderWidth	0 Dimension	N/A
XmNheight XmCHeight	dynamic Dimension	CSG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Object Resource Set		
Name **Class**	**Default** **Type**	**Access**
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

Callback Information

A pointer to the following structure is passed to each callback:

```
typedef struct
{
    int             reason;
    XEvent          * event;
    int             set;
} XmToggleButtonCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the **XEvent** that triggered the callback

set Reflects the ToggleButtonGadget's current state when the callback occurred, either True (selected) or False (unselected)

Behavior

XmToggleButtonGadget includes behavior from **XmGadget**. **XmToggleButtonGadget** includes menu traversal behavior from **XmLabelGadget**. Additional **XmToggleButtonGadget** behavior is described in the following list:

BTransfer Press:

Drags the contents of a ToggleButtonGadget label, identified by pressing **BTransfer**. This action creates a DragContext object whose **XmNexportTargets** resource is set to **COMPOUND_TEXT** for a label type of **XmSTRING**, or **PIXMAP** if the label type is **XmPIXMAP**. This action is undefined for ToggleButtonGadgets used in a menu system.

BSelect Press:

In a menu, this action unposts any menus posted by the ToggleButtonGadget's parent menu, disables keyboard traversal for the menu, and enables mouse traversal for the menu. It draws the shadow in the armed state and, unless the button is already armed, calls the **XmNarmCallback** callbacks.

Outside a menu, if the button was previously unset, this action does the following: if **XmNindicatorOn** is True, it draws the indicator shadow so that the indicator looks pressed; if **XmNfillOnSelect** is True, it fills the indicator with the color specified by **XmNselectColor**. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks pressed. If **XmNlabelType** is **XmPIXMAP**, the **XmNselectPixmap** is used as the button face. This resource calls the **XmNarmCallback** callbacks.

Outside a menu, if the button was previously set, this action does the following: if both **XmNindicatorOn** and **XmNvisibleWhenOff** are True, it draws the indicator shadow so that the indicator looks raised; if **XmNfillOnSelect** is True, it fills the indicator with the background color. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks raised. If **XmNlabelType** is **XmPIXMAP**, the **XmNlabelPixmap** is used as the button face. This resource calls the **XmNarmCallback** callbacks.

BSelect Release:

In a menu, this action unposts all menus in the menu hierarchy. If the ToggleButtonGadget was previously set, this action unsets it; if the ToggleButtonGadget was previously unset, this action sets it. It calls the **XmNvalueChangedCallback** callbacks and then the **XmNdisarmCallback** callbacks.

If the button is outside a menu and the pointer is within the button, this action does the following: if the button was previously unset, sets it; if the button was previously set, unsets it. This action calls the **XmNvalueChangedCallback** callbacks.

If the button is outside a menu, this action calls the **XmNdisarmCallback** callbacks.

KHelp: In a Pulldown or Popup MenuPane, unposts all menus in the menu hierarchy and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the menu system was entered. Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

KActivate: In a menu, this action unposts all menus in the menu hierarchy. Unless the button is already armed, this action calls the **XmNarmCallback** callbacks; and calls the **XmNvalueChangedCallback** and **XmNdisarmCallback** callbacks. Outside a menu, if the parent is a manager, this action passes the event to the parent.

KSelect: If the ToggleButtonGadget was previously set, this action unsets it; if the ToggleButtonGadget was previously unset, this action sets it.

In a menu, this action unposts all menus in the menu hierarchy. Unless the button is already armed, this action calls the **XmNarmCallback**, the **XmNvalueChangedCallback**, and **XmNdisarmCallback** callbacks.

Outside a menu, if the button was previously unset, this action does the following: If **XmNindicatorOn** is True, it draws the indicator shadow so that the indicator looks pressed; if **XmNfillOnSelect** is True, it fills the indicator with the color specified by **XmNselectColor**. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks pressed. If **XmNlabelType** is **XmPIXMAP**, the **XmNselectPixmap** is used as the button face. This action calls the **XmNarmCallback**, **XmNvalueChangedCallback**, **XmNdisarmCallback** callbacks.

Outside a menu, if the button was previously set, this action does the following: If both **XmNindicatorOn** and **XmNvisibleWhenOff** are True, it draws the indicator shadow so that the indicator looks raised; if **XmNfillOnSelect** is True, it fills the indicator with the background color. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks raised. If **XmNlabelType** is

XmPIXMAP, the **XmNlabelPixmap** is used as the button face. Calls the **XmNarmCallback**, **XmNvalueChangedCallback**, and **XmNdisarmCallback** callbacks.

MAny KCancel:

In a top-level Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is **XmEXPLICT**, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, this action unposts the menu. Outside a menu, if the parent is a manager, this action passes the event to the parent.

In a Popup MenuPane, this action unposts the menu and restores keyboard focus to the widget from which the menu was posted.

<Enter>: In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the armed state and calls the **XmNarmCallback** callbacks.

If the ToggleButtonGadget is not in a menu and the cursor leaves and then reenters the ToggleButtonGadget while the button is pressed, this action restores the button's armed appearance.

<Leave>: In a menu, if keyboard traversal is enabled, this action does nothing. Otherwise, it draws the shadow in the unarmed state and calls the **XmNdisarmCallback** callbacks.

If the ToggleButtonGadget is not in a menu and the cursor leaves the ToggleButtonGadget while the button is pressed, this action restores the button's unarmed appearance.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings(3X)**.

Related Information

Object(3X), **RectObj(3X)**, **XmCreateRadioBox(3X)**, **XmCreateToggleButtonGadget(3X)**, **XmGadget(3X)**, **XmLabelGadget(3X)**, **XmRowColumn(3X)**, **XmToggleButtonGadgetGetState(3X)**, and **XmToggleButtonGadgetSetState(3X)**.

XmToggleButtonGadgetGetState—A ToggleButtonGadget function that obtains the state of a ToggleButtonGadget

Synopsis

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

Boolean XmToggleButtonGadgetGetState (widget)
        Widget          widget;
```

Description

XmToggleButtonGadgetGetState obtains the state of a ToggleButtonGadget.

widget Specifies the ToggleButtonGadget ID

For a complete definition of ToggleButtonGadget and its associated resources, see **XmToggleButtonGadget(3X)**.

Return Value

Returns True if the button is selected and False if the button is unselected.

Related Information

XmToggleButtonGadget(3X).

XmToggleButtonGadgetSetState—A ToggleButtonGadget function that sets or changes the current state

Synopsis

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

void XmToggleButtonGadgetSetState (widget, state, notify)
        Widget      widget;
        Boolean     state;
        Boolean     notify;
```

Description

XmToggleButtonGadgetSetState sets or changes the ToggleButtonGadget's current state.

widget Specifies the ToggleButtonGadget widget ID.

state Specifies a Boolean value that indicates whether the ToggleButtonGadget state is selected or unselected. If the value is True, the button state is selected; if it is False, the button state is unselected.

notify Indicates whether **XmNvalueChangedCallback** is called; it can be either True or False. The **XmNvalueChangedCallback** is only called when this function changes the state of the ToggleButtonGadget. When this argument is True and the ToggleButtonGadget is a child of a RowColumn widget whose **XmNradioBehavior** is True, setting the ToggleButtonGadget causes other ToggleButton and ToggleButtonGadget children of the RowColumn to be unselected.

For a complete definition of ToggleButtonGadget and its associated resources, see **XmToggleButtonGadget(3X)**.

Related Information

XmToggleButtonGadget(3X).

XmToggleButtonGetState—A ToggleButton function that obtains the state of a ToggleButton

Synopsis

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

Boolean XmToggleButtonGetState (widget)
        Widget          widget;
```

Description

XmToggleButtonGetState obtains the state of a ToggleButton.

widget Specifies the ToggleButton widget ID

For a complete definition of ToggleButton and its associated resources, see **XmToggleButton(3X)**.

Return Value

Returns True if the button is selected and False if the button is unselected.

Related Information

XmToggleButton(3X).

XmToggleButtonSetState—A ToggleButton function that sets or changes the current state

Synopsis

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

void XmToggleButtonSetState (widget, state, notify)
        Widget      widget;
        Boolean     state;
        Boolean     notify;
```

Description

XmToggleButtonSetState sets or changes the ToggleButton's current state.

widget Specifies the ToggleButton widget ID.

state Specifies a Boolean value that indicates whether the ToggleButton state is selected or unselected. If the value is True, the button state is selected; if it is False, the button state is unselected.

notify Indicates whether **XmNvalueChangedCallback** is called; it can be either True or False. The **XmNvalueChangedCallback** is only called when this function changes the state of the ToggleButton. When this argument is True and the ToggleButton is a child of a RowColumn widget whose **XmNradioBehavior** is True, setting the ToggleButton causes other ToggleButton and ToggleButtonGadget children of the RowColumn to be unselected.

For a complete definition of ToggleButton and its associated resources, see **XmToggleButton(3X)**.

Related Information

XmToggleButton(3X).

XmTrackingEvent—A Toolkit function that provides a modal interaction

Synopsis

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

Widget XmTrackingEvent (widget, cursor, confine_to, event_return)
        Widget      widget;
        Cursor      cursor;
        Boolean     confine_to;
        XEvent      *event_return;
```

Description

XmTrackingEvent provides a modal interface for selection of a component. It is intended to support context help. The function grabs the pointer and discards succeeding events until **BSelect** is released or a key is pressed and then released. The function then returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released.

widget Specifies the widget ID of a widget to use as the basis of the modal interaction. That is, the widget within which the interaction must occur, usually a top-level shell.

cursor Specifies the cursor to be used for the pointer during the interaction. This is a standard X cursor name.

confine_to Specifies whether or not the cursor should be confined to *widget*.

event_return Returns the ButtonRelease or KeyRelease event that causes the function to return.

Return Value

Returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released. If no widget or gadget contains the pointer, the function returns NULL.

Related Information

XmTrackingLocate(3X).

XmTrackingLocate—A Toolkit function that provides a modal interaction

Synopsis

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

Widget XmTrackingLocate (widget, cursor, confine_to)
        Widget      widget;
        Cursor      cursor;
        Boolean     confine_to;
```

Description

XmTrackingLocate provides a modal interface for selection of a component. It is intended to support context help. The function grabs the pointer and discards succeeding events until **BSelect** is released or a key is pressed and then released. The function then returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released.

NOTE: This function is obsolete and exists for compatibility with previous releases. It has been replaced by **XmTrackingEvent**.

widget Specifies the widget ID of a widget to use as the basis of the modal interaction. That is, the widget within which the interaction must occur, usually a top-level shell.

cursor Specifies the cursor to be used for the pointer during the interaction. This is a standard X cursor name.

confine_to Specifies whether or not the cursor should be confined to *widget*

Return Value

Returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released. If no widget or gadget contains the pointer, the function returns NULL.

Related Information

XmTrackingEvent(3X).

XmTranslateKey—The default keycode-to-keysym translator

Synopsis

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

void XmTranslateKey (display, keycode, modifiers, modifiers_return, keysym_return)
        Display     *display;
        KeyCode     keycode;
        Modifiers   modifiers;
        Modifiers   *modifiers_return;
        KeySym      *keysym_return;
```

Description

XmTranslateKey is the default **XtKeyProc** translation procedure for Motif applications. The function takes a keycode and modifiers and returns the corresponding keysym.

XmTranslateKey serves two main purposes: to enable new translators with expanded functionality to get the default Motif keycode-to-keysym translation in addition to whatever they add, and to reinstall the default translator. This function enables keysyms defined by the Motif virtual bindings to be used when an application requires its own **XtKeyProc** to be installed.

display Specifies the display that the keycode is from

keycode Specifies the keycode to translate

modifiers Specifies the modifier keys to be applied to the keycode

modifiers_return
Specifies a mask of the modifier keys actually used to generate the keysym (an AND of *modifiers* and any default modifiers applied by the currently registered translator)

keysym_return
Specifies a pointer to the resulting keysym

Related Information

VirtualBindings(3X).

XmUninstallImage—A pixmap caching function that removes an image from the image cache

Synopsis

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

Boolean XmUninstallImage (image)
        XImage          * image;
```

Description

XmUninstallImage removes an image from the image cache.

image Points to the image structure given to the **XmInstallImage**() routine

Return Value

Returns True when successful; returns False if the *image* is NULL, or if it cannot be found to be uninstalled.

Related Information

XmInstallImage(3X), **XmGetPixmap(3X)**, and **XmDestroyPixmap(3X)**.

XmUpdateDisplay—A function that processes all pending exposure events immediately

Synopsis

```
void XmUpdateDisplay (widget)
        Widget          widget;
```

Description

XmUpdateDisplay provides the application with a mechanism for forcing all pending exposure events to be removed from the input queue and processed immediately. When a user selects a button within a MenuPane, the MenuPanes are unposted and then any activation callbacks registered by the application are invoked. If one of the callbacks performs a time-consuming action, the portion of the application window that was covered by the MenuPanes will not have been redrawn; normal exposure processing does not occur until all of the callbacks have been invoked. If the application writer suspects that a callback will take a long time, then the callback may choose to invoke **XmUpdateDisplay** before starting its time-consuming operation. This function is also useful any time a transient window, such as a dialog box, is unposted; callbacks are invoked before normal exposure processing can occur.

widget Specifies any widget or gadget.

XmVaCreateSimpleCheckBox—A RowColumn widget convenience creation function

Synopsis

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

Widget XmVaCreateSimpleCheckBox (parent, name, callback, arg...)
        Widget      parent;
        String      name;
        XtCallbackProc callback;
```

Description

XmVaCreateSimpleCheckBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (**varargs**) calling convention.

This routine creates a CheckBox and its ToggleButtonGadget children. A CheckBox is similar to a RadioBox, except that more than one button can be selected at a time. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. Buttons are named and created in the order in which they are specified in the variable portion of the argument list.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

callback Specifies a callback procedure to be called when a button's value changes. This callback function is added to each button after creation as the button's **XmNvalueChangedCallback**. The callback function is called when a button's value changes, and the button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. The following list describes the possible first arguments in each group of **varargs**:

XmVaCHECKBUTTON
This is followed by four additional arguments. The set specifies one button in the CheckBox and some of its resource values. The following list describes the additional four arguments, in order.

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**. This is ignored in this release.

accelerator The accelerator, of type **String**. This is ignored in this release.

accelerator_text
The accelerator text, of type **XmString**. This is ignored in this release.

resource_name
This is followed by one additional argument, the value of the resource, of type **XtArgVal**. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg
This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name The resource name, of type **String**

type The type of the resource value supplied, of type **String**

value The resource value (or a pointer to the resource value, depending on the type and size of the value), of type **XtArgVal**

size The size of the resource value in bytes, of type **int**

XtVaNestedList
This is followed by one additional argument of type **XtVarArgsList**. This argument is a nested list of **varargs** returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3X), **XmCreateRowColumn(3X)**, **XmCreateSimpleCheckBox(3X)**, **XmCreateSimpleRadioBox(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimpleRadioBox(3X)**.

XmVaCreateSimpleMenuBar—A RowColumn widget convenience creation function

Synopsis

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

Widget XmVaCreateSimpleMenuBar (parent, name, arg...)
        Widget      parent;
        String      name;
```

Description

XmVaCreateSimpleMenuBar creates an instance of a RowColumn widget of type **XmMENU_BAR** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (**varargs**) calling convention.

This routine creates a MenuBar and its CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. Buttons are named and created in the order in which they are specified in the variable portion of the argument list.

parent Specifies the parent widget ID

name Specifies the name of the created widget

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. Following are the possible first arguments in each group of **varargs**:

XmVaCASCADEBUTTON

This is followed by two additional arguments. The set specifies one button in the MenuBar and some of its resource values. Following are the additional two arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

resource_name

This is followed by one additional argument, the value of the resource, of type **XtArgVal**. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg

This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type **XtArgVal**
size	The size of the resource value in bytes, of type **int**

XtVaNestedList

This is followed by one additional argument of type **XtVarArgsList**. This argument is a nested list of **varargs** returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateMenuBar(3X), **XmCreateRowColumn(3X)**, **XmCreateSimpleMenuBar(3X)**, and **XmRowColumn(3X)**.

XmVaCreateSimpleOptionMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmVaCreateSimpleOptionMenu (parent, name, option_label,
option_mnemonic, button_set, callback, arg...)
        Widget      parent;
        String      name;
        XmString    option_label;
        KeySym      option_mnemonic;
        int         button_set;
        XtCallbackProccallback;
```

Description

XmVaCreateSimpleOptionMenu creates an instance of a RowColumn widget of type **XmMENU_OPTION** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (**varargs**) calling convention.

This routine creates an OptionMenu and its Pulldown submenu containing PushButtonGadget or CascadeButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of separators in the menu. Buttons and separators are named and created in the order in which they are specified in the variable portion of the argument list.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

option_label Specifies the label string to be used on the left side of the OptionMenu.

option_mnemonic
Specifies a keysym for a key that, when pressed by the user, posts the associated Pulldown MenuPane.

button_set Specifies which PushButtonGadget is initially set. The value is the integer *n* that corresponds to the *n*th PushButtonGadget specified in the variable portion of the argument list. Only a PushButtonGadget can be set, and only PushButtonGadgets are counted in determining the integer *n*. The first PushButtonGadget is number 0 (zero).

callback Specifies a callback procedure to be called when a button is activated. This callback function is added to each button after creation as the button's **XmNactivateCallback**. The callback function is called when a button is activated, and the button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. Following are the possible first arguments in each group of **varargs**:

XmVaPUSHBUTTON
This is followed by four additional arguments. The set specifies one button in the OptionMenu's Pulldown submenu and some of its resource values. The button created is a PushButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaSEPARATOR
This is followed by no additional arguments. It specifies one separator in the OptionMenu's Pulldown submenu.

XmVaDOUBLE_SEPARATOR
This is followed by no additional arguments. It specifies one separator in the OptionMenu's Pulldown submenu. The separator type is **XmDOUBLE_LINE**.

resource_name
This is followed by one additional argument, the value of the resource, of type **XtArgVal**. The pair specifies a resource and its value for the Pulldown submenu.

XtVaTypedArg

This is followed by four additional arguments. The set specifies a resource and its value for the Pulldown submenu. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type **XtArgVal**
size	The size of the resource value in bytes, of type **int**

XtVaNestedList

This is followed by one additional argument of type **XtVarArgsList**. This argument is a nested list of **varargs** returned by **XtVaCreateArgsList**.

The user can specify resources in a resource file for the automatically created widgets and gadgets of an OptionMenu. The following list identifies the names of these widgets (or gadgets) and the associated OptionMenu areas:

Option Menu Label Gadget	**OptionLabel**
Option Menu Cascade Button	**OptionButton**

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateOptionMenu(3X), **XmCreateRowColumn(3X)**, **XmCreateSimpleOptionMenu(3X)**, and **XmRowColumn(3X)**.

XmVaCreateSimplePopupMenu—A RowColumn widget convenience creation function

Synopsis

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

Widget XmVaCreateSimplePopupMenu (parent, name, callback, arg...)
        Widget      parent;
        String      name;
        XtCallbackProc callback;
```

Description

XmVaCreateSimplePopupMenu creates an instance of a RowColumn widget of type **XmMENU_POPUP** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (**varargs**) calling convention.

This routine creates a Popup MenuPane and its button children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu. The name of each separator is **separator_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of separators in the menu. The name of each title is **label_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of titles in the menu. Buttons, separators, and titles are named and created in the order in which they are specified in the variable portion of the argument list.

parent Specifies the widget ID of the parent of the MenuShell

name Specifies the name of the created widget

callback Specifies a callback procedure to be called when a button is activated or when its value changes. This callback function is added to each button after creation. For a CascadeButtonGadget or a PushButtonGadget, the callback is added as the button's **XmNactivateCallback**, and it is called when the button is activated. For a ToggleButtonGadget, the callback is added as the button's **XmNvalueChangedCallback**, and it is called when the button's value changes. The button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. The following list describes the possible first arguments in each group of **varargs**.

XmVaCASCADEBUTTON

This is followed by two additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a CascadeButtonGadget. Following are the additional two arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

XmVaPUSHBUTTON

This is followed by four additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a PushButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaRADIOBUTTON

This is followed by four additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaCHECKBUTTON
This is followed by four additional arguments. The set specifies one button in the PopupMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaTITLE This is followed by one additional argument. The pair specifies a title LabelGadget in the PopupMenu. Following is the additional argument:

title The title string, of type **XmString**

XmVaSEPARATOR
This is followed by no additional arguments. It specifies one separator in the PopupMenu.

XmVaDOUBLE_SEPARATOR
This is followed by no additional arguments. It specifies one separator in the PopupMenu. The separator type is **XmDOUBLE_LINE**.

resource_name
This is followed by one additional argument, the value of the resource, of type **XtArgVal**. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg
This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name The resource name, of type **String**

type The type of the resource value supplied, of type **String**

value The resource value (or a pointer to the resource value, depending on the type and size of the value), of type **XtArgVal**

size The size of the resource value in bytes, of type **int**

XtVaNestedList

This is followed by one additional argument of type **XtVarArgsList**. This argument is a nested list of **varargs** returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreatePopupMenu(3X), **XmCreateRowColumn(3X)**, **XmCreateSimplePopupMenu(3X)**, and **XmRowColumn(3X)**.

XmVaCreateSimplePulldownMenu—A RowColumn widget convenience creation function

Synopsis

#include <Xm/RowColumn.h>

Widget XmVaCreateSimplePulldownMenu (*parent, name, post_from_button, callback, arg...*)

Widget *parent*;
String *name*;
int *post_from_button*;
XtCallbackProc*callback*;

Description

XmVaCreateSimplePulldownMenu creates an instance of a RowColumn widget of type **XmMENU_PULLDOWN** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (**varargs**) calling convention.

This routine creates a Pulldown MenuPane and its button children. The name of each button is **button_***n*, where *n* is an integer from 0 to 1 minus the number of buttons in the menu. The name of each separator is **separator_***n*, where *n* is an integer from 0 to 1 minus the number of separators in the menu. The name of each title is **label_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of titles in the menu. Buttons, separators, and titles are named and created in the order in which they are specified in the variable portion of the argument list.

This routine also attaches the PulldownMenu to a CascadeButton or CascadeButtonGadget in the parent. The PulldownMenu is then posted from this button.

parent Specifies the widget ID of the parent of the MenuShell.

name Specifies the name of the created widget.

post_from_button
Specifies the CascadeButton or CascadeButtonGadget in the parent to which the Pulldown MenuPane is attached. The value is the integer *n* that corresponds to the *n*th CascadeButton or CascadeButtonGadget specified for the parent of the Pulldown MenuPane. A Pulldown MenuPane can be attached only to a CascadeButton or CascadeButtonGadget, and only CascadeButtons and CascadeButtonGadgets are counted in determining the integer *n*. The first CascadeButton or CascadeButtonGadget is number 0 (zero).

callback Specifies a callback procedure to be called when a button is activated or when its value changes. This callback function is added to each button after creation. For a CascadeButtonGadget or a PushButtonGadget, the callback is added as the button's **XmNactivateCallback**, and it is called when the button is activated. For a ToggleButtonGadget, the callback is added as the button's **XmNvalueChangedCallback**, and it is called when the button's value changes. The button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. Following are the possible first arguments in each group of **varargs**:

XmVaCASCADEBUTTON

This is followed by two additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a CascadeButtonGadget. Following are the additional two arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

XmVaPUSHBUTTON

This is followed by four additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a PushButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaRADIOBUTTON
This is followed by four additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaCHECKBUTTON
This is followed by four additional arguments. The set specifies one button in the PulldownMenu and some of its resource values. The button created is a ToggleButtonGadget. Following are the additional four arguments, in order:

label The label string, of type **XmString.**

mnemonic The mnemonic, of type **KeySym**

accelerator The accelerator, of type **String**

accelerator_text
The accelerator text, of type **XmString**

XmVaTITLE This is followed by one additional argument. The pair specifies a title LabelGadget in the PulldownMenu. Following is the additional argument:

title The title string, of type **XmString**

XmVaSEPARATOR
This is followed by no additional arguments. It specifies one separator in the PulldownMenu.

XmVaDOUBLE_SEPARATOR
This is followed by no additional arguments. It specifies one separator in the PulldownMenu. The separator type is **XmDOUBLE_LINE**.

resource_name
This is followed by one additional argument, the value of the resource, of type **XtArgVal**. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg
This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in order:

name	The resource name, of type **String**
type	The type of the resource value supplied, of type **String**
value	The resource value (or a pointer to the resource value, depending on the type and size of the value), of type **XtArgVal**
size	The size of the resource value in bytes, of type **int**

XtVaNestedList
This is followed by one additional argument of type XtVarArgsList. This argument is a nested list of **varargs** returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreatePulldownMenu(3X), **XmCreateRowColumn(3X)**, **XmCreateSimplePulldownMenu**, and **XmRowColumn(3X)**.

XmVaCreateSimpleRadioBox—A RowColumn widget convenience creation function

Synopsis

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

Widget XmVaCreateSimpleRadioBox (parent, name, button_set, callback, arg...)
        Widget      parent;
        String      name;
        int         button_set;
        XtCallbackProc callback;
```

Description

XmVaCreateSimpleRadioBox creates an instance of a RowColumn widget of type **XmWORK_AREA** and returns the associated widget ID. This routine uses the ANSI C variable-length argument list (**varargs**) calling convention.

This routine creates a RadioBox and its ToggleButtonGadget children. The name of each button is **button_***n*, where *n* is an integer from 0 (zero) to 1 minus the number of buttons in the menu.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

button_set Specifies which button is initially set. The value is the integer *n* in the button name **button_***n*.

callback Specifies a callback procedure to be called when a button's value changes. This callback function is added to each button after creation as the button's **XmNvalueChangedCallback**. The callback function is called when a button's value changes, and the button number is returned in the *client_data* field.

The variable portion of the argument list consists of groups of arguments. The first argument in each group is a constant or a string and determines which arguments follow in that group. The last argument in the list must be NULL. The following list describes the possible first arguments in each group of **varargs**.

XmVaRADIOBUTTON
This is followed by four additional arguments. The set specifies one button in the RadioBox and some of its resource values. Following are the additional four arguments, in order:

label The label string, of type **XmString**.

mnemonic The mnemonic, of type **KeySym**. This is ignored in this release.

accelerator The accelerator, of type **String**. This is ignored in this release.

accelerator_text
The accelerator text, of type **XmString**. This is ignored in this release.

resource_name
This is followed by one additional argument, the value of the resource, of type **XtArgVal**. The pair specifies a resource and its value for the RowColumn widget.

XtVaTypedArg
This is followed by four additional arguments. The set specifies a resource and its value for the RowColumn widget. A resource type conversion is performed if necessary. Following are the additional four arguments, in this order:

name The resource name, of type **String**

type The type of the resource value supplied, of type **String**

value The resource value (or a pointer to the resource value, depending on the type and size of the value), of type **XtArgVal**

size The size of the resource value in bytes, of type **int**

XtVaNestedList
This is followed by one additional argument of type **XtVarArgsList**. This argument is a nested list of **varargs** returned by **XtVaCreateArgsList**.

For more information on variable-length argument lists, see the X Toolkit Intrinsics documentation.

A number of resources exist specifically for use with this and other simple menu creation routines. For a complete definition of RowColumn and its associated resources, see **XmRowColumn(3X)**.

Return Value

Returns the RowColumn widget ID.

Related Information

XmCreateRadioBox(3X), **XmCreateRowColumn(3X)**, **XmCreateSimpleCheckBox(3X)**, **XmCreateSimpleRadioBox(3X)**, **XmRowColumn(3X)**, and **XmVaCreateSimpleCheckBox(3X)**,

XmWidgetGetBaselines—Retrieves baseline information for a widget

Synopsis

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

Boolean XmWidgetGetBaselines (widget, baselines, line_count)
        Widget      widget;
        Dimension   **baselines;
        int         *line_count;
```

Description

XmWidgetGetBaselines returns an array that contains one or more baseline values associated with the specified widget. The baseline of any given line of text is a vertical offset in pixels from the origin of the widget's bounding box to the given baseline. This routine allocates memory for the returned data. The application must free this memory using **XtFree**.

widget Specifies the ID of the widget for which baseline values are requested

baselines Returns an array that contains the value of each baseline of text in the widget

line_count Returns the number of lines in the widget

Return Value

Returns a Boolean value that indicates whether the widget contains a baseline. If the value is True, the function returns a value for each baseline of text. If it is False, the function was unable to return a baseline value.

Related Information

XmWidgetGetDisplayRect(3X).

XmWidgetGetDisplayRect—Retrieves display rectangle information for a widget

Synopsis

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

Boolean XmWidgetGetDisplayRect (widget, displayrect)
        Widget      widget;
        XRectangle  *displayrect;
```

Description

XmWidgetGetDisplayRect returns the width, height and the x and y-coordinates of the upper left corner of the display rectangle of the specified widget. The display rectangle is the smallest rectangle that encloses either a string or a pixmap.

If the widget contains a string, the return values specify the x and y-coordinates of the upper left corner of the display rectangle relative to the origin of the widget and the width and height in pixels.

In the case of a pixmap, the return values specify the x and y-coordinates of the upper left corner of the pixmap relative to the origin, and the width and height of the pixmap in pixels.

widget Specifies the widget ID

displayrect Specifies a pointer to an XRectangle structure in which the x and y-coordinates, width and height of the display rectangle are returned

Return Value

Returns True if the specified widget has an associated display rectangle; otherwise, returns False.

Related Information

XmWidgetGetBaselines(3X).

UIL—The user interface language file format

Synopsis

MODULE *module_name*
[**NAMES = CASE_INSENSITIVE** | **CASE_SENSITIVE**]
[**CHARACTER_SET** = *character_set*]
[**OBJECTS** = { *widget_name* = **GADGET** | **WIDGET**; [...] }]
{ [
[*value_section*] |
[*procedure_section*] |
[*list_section*] |
[*object_section*] |
[*identifier_section*]
[...]
] }
END MODULE;

Description

The UIL language is used for describing the initial state of a user interface for a widget based application. UIL describes the widgets used in the interface, the resources of those widgets, and the callbacks of those widgets. The UIL file is compiled into a UID file using the command **uil** or by the callable compiler **Uil()**. The contents of the compiled UID file can then by accessed by the various Motif Resource Management (MRM) functions from within an application program.

File Format

UIL is a free-form language. This means that high-level constructs such as object and value declarations do not need to begin in any particular column and can span any number of lines. Low-level constructs such as keywords and punctuation characters can also begin in any column; however, except for string literals and comments, they cannot span lines.

The UIL compiler accepts input lines up to 132 characters in length.

MODULE *module_name*
The name by which the UIL module is known in the UID file. This name is stored in the UID file for later use in the retrieval of resources by the MRM. This name is always stored in uppercase in the UID file.

NAMES = CASE_INSENSITIVE | CASE_SENSITIVE
Indicates whether names should be treated as case sensitive or case insensitive. The default is case sensitive. The case-sensitivity clause should be the first clause in the module header, and in any case must precede any statement that contains a name. If names are case sensitive in a UIL module, UIL keywords in that module must be in

lowercase. Each name is stored in the UIL file in the same case as it appears in the UIL module. If names are case insensitive, then keywords can be in uppercase, lowercase, or mixed case, and the uppercase equivalent of each name is stored in the UID file.

CHARACTER_SET = *character_set*
Specifies the default character set for string literals in the module that do not explicitly set their character set. The default character set, in the absence of this clause is the codeset component of the **LANG** environment variable, or the value of **XmFALLBACK_CHARSET** if **LANG** is not set or has no codeset component. The value of **XmFALLBACK_CHARSET** is defined by the UIL supplier, but is usually ISO8859-1 (equivalent to ISO_LATIN1). Use of this clause turns off all localized string literal processing turned on by the compiler flag **-s** or the **Uil_command_type** data structure element **use_setlocale_flag**.

OBJECTS = { *widget_name* = **GADGET** | **WIDGET**; }
Indicates whether the widget or gadget form of the control specified by *widget_name* is used by default. By default, the widget form is used, so the gadget keyword is usually the only one used. The specified control should be one that has both a widget and gadget version: **XmCascadeButton**, **XmLabel**, **XmPushButton**, **XmSeparator**, and **XmToggleButton**. The form of more than one control can be specified by delimiting them with semicolons. The gadget or widget form of an instance of a control can be specified with the **GADGET** and **WIDGET** keywords in a particular object declaration.

value_section
Provides a way to name a value expression or literal. The value name can then be referred to by declarations that occur elsewhere in the UIL module in any context where a value can be used. Values can be forward referenced. Value sections are described in more detail later in the reference page.

procedure_section
Defines the callback routines used by a widget and the creation routines for user-defined widgets. These definitions are used for error checking. Procedure sections are described in more detail later in the reference page.

list_section Provides a way to group together a set of arguments, controls (children), callbacks, or procedures for later use in the UIL module. Lists can contain other lists, so that you can set up a hierarchy to

clearly show which arguments, controls, callbacks, and procedures are common to which widgets. List sections are described in more detail later in the reference page.

object_section
Defines the objects that make up the user interface of the application. You can reference the object names in declarations that occur elsewhere in the UIL module in any context where an object name can be used (for example, in a controls list, as a symbolic reference to a widget ID, or as the *tag_value* argument for a callback procedure). Objects can be forward referenced. Object sections are described in more detail later in the reference page.

identifier_section
Defines a run-time binding of data to names that appear in the UIL module. Identifier sections are described in more detail later in the reference page.

The UIL file can also contain comments and include directives, which are described along with the main elements of the UIL file format in the following sections.

Comments

Comments can take one of two forms, as follows:

- The comment is introduced with the sequence **/*** followed by the text of the comment and terminated with the sequence ***/**. This form of comment can span multiple source lines.
- The comment is introduced with an ! (exclamation point), followed by the text of the comment and terminated by the end of the source line.

Neither form of comment can be nested.

Value sections

A value section consists of the keyword **VALUE** followed by a sequence of value declarations. It has the following syntax:

VALUE *value_name* **:**
[EXPORTED | PRIVATE] *value_expression* **|**
IMPORTED *value_type* **;**

Where *value_expression* is assigned to *value_name* or a *value_type* is assigned to an imported value name. A value declaration provides a way to name a value expression or literal. The value name can be referred to by declarations that occur later in the UIL module in any context where a value can be used. Values can be forward referenced.

EXPORTED
A value that you define as exported is stored in the UID file as a named resource, and therefore can be referenced by name in other UID files. When you define a value as exported, MRM looks outside the module in which the exported value is declared to get its value at run time.

PRIVATE A private value is a value that is not imported or exported. A value that you define as private is not stored as a distinct resource in the UID file. You can reference a private value only in the UIL module containing the value declaration. The value or object is directly incorporated into anything in the UIL module that references the declaration.

IMPORTED
A value that you define as imported is one that is defined as a named resource in a UID file. MRM resolves this declaration with the corresponding exported declaration at application run time.

By default, values and objects are private. The following is a list of the supported value types in UIL:

- **ANY**
- **ARGUMENT**
- **BOOLEAN**
- **COLOR**
- **COLOR_TABLE**
- **COMPOUND_STRING**
- **FLOAT**
- **FONT**
- **FONT_TABLE**
- **FONTSET**
- **ICON**
- **INTEGER**
- **INTEGER_TABLE**
- **KEYSYM**

- **REASON**
- **SINGLE_FLOAT**
- **STRING**
- **STRING_TABLE**
- **TRANSLATION_TABLE**
- **WIDE_CHARACTER**
- **WIDGET**

Procedure sections

A procedure section consists of the keyword **PROCEDURE** followed by a sequence of procedure declarations. It has the following syntax:

PROCEDURE
procedure_name [([*value_type*])] ;

Use a procedure declaration to declare

- A routine that can be used as a callback routine for a widget
- The creation function for a user-defined widget

You can reference a procedure name in declarations that occur later in the UIL module in any context where a procedure can be used. Procedures can be forward referenced. You cannot use a name you used in another context as a procedure name.

In a procedure declaration, you have the option of specifying that a parameter will be passed to the corresponding callback routine at run time. This parameter is called the callback tag. You can specify the data type of the callback tag by putting the data type in parentheses following the procedure name. When you compile the module, the UIL compiler checks that the argument you specify in references to the procedure is of this type. Note that the data type of the callback tag must be one of the valid UIL data types. You can use a widget as a callback tag, as long as the widget is defined in the same widget hierarchy as the callback, that is they have a common ancestor that is in the same UIL hierarchy.

The following list summarizes how the UIL compiler checks argument type and argument count, depending on the procedure declaration.

No parameters
: No argument type or argument count checking occurs. You can supply either zero or one arguments in the procedure reference.

( )
: Checks that the argument count is 0 (zero).

(ANY)
: Checks that the argument count is 1. Does not check the argument type. Use the **ANY** type to prevent type checking on procedure tags.

(*type*)
: Checks for one argument of the specified type.

(*class_name*)
: Checks for one widget argument of the specified widget class.

While it is possible to use any UIL data type to specify the type of a tag in a procedure declaration, you must be able to represent that data type in the programming language you are using. Some data types (such as integer, Boolean, and string) are common data types recognized by most programming languages. Other UIL data types (such as string tables) are more complicated and may require that you set up an appropriate corresponding data structure in the application in order to pass a tag of that type to a callback routine.

You can also use a procedure declaration to specify the creation function for a user-defined widget. In this case, you specify no formal parameters. The procedure is invoked with the standard three arguments passed to all widget creation functions. (See the Motif Toolkit documentation for more information about widget creation functions.)

List sections

A list section consists of the keyword **LIST** followed by a sequence of list declarations. It has the following syntax:

LIST
list_name **:** { *list_item*; [...] }
[...]

You can also use list sections to group together a set of arguments, controls (children), callbacks, or procedures for later use in the UIL module. Lists can contain other lists, so that you can set up a hierarchy to clearly show which arguments, controls, callbacks, and procedures are common to which widgets. You cannot mix the different types of lists; a list of a particular type cannot contain entries of a different list type or reference the name of a different list type. A list name is always private to the UIL module in which you declare the list and cannot be stored as a named resource in a UID file.

The additional list types are described in the following sections.

Arguments List Structure

An arguments list defines which arguments are to be specified in the arguments list parameter when the creation routine for a particular object is called at run time. An arguments list also specifies the values for those arguments. Argument lists have the following syntax:

LIST
list_name **: ARGUMENTS {**
argument_name = *value_expression*;
[...] } [...]

The argument name must be either a built-in argument name or a user-defined argument name that is specified with the **ARGUMENT** function.

If you use a built-in argument name as an arguments list entry in an object definition, the UIL compiler checks the argument name to be sure that it is supported by the type of object that you are defining. If the same argument name appears more than once in a given arguments list, the last entry that uses that argument name supersedes all previous entries with that name, and the compiler issues a message.

Some arguments, such as **XmNitems** and **XmNitemCount**, are coupled by the UIL compiler. When you specify one of the arguments, the compiler also sets the other. The coupled argument is not available to you.

The Motif Toolkit and the X Toolkit (intrinsics) support constraint arguments. A constraint argument is one that is passed to children of an object, beyond those arguments normally available. For example, the Form widget grants a set of constraint arguments to its children. These arguments control the position of the children within the Form.

Unlike the arguments used to define the attributes of a particular widget, constraint arguments are used exclusively to define additional attributes of the children of a particular widget. These attributes affect the behavior of the children within their parent. To supply constraint arguments to the children, you include the arguments in the arguments list for the child.

See Appendix B for information about which arguments are supported by which widgets. See Appendix C for information about what the valid value type is for each built-in argument.

Callbacks List Structure

Use a callbacks list to define which callback reasons are to be processed by a particular widget at run time. Callback lists have the following syntax:

LIST
list_name : **CALLBACKS** {
reason_name = **PROCEDURE** *procedure_name*
[([*value_expression*])]; |
reason_name = *procedure_list* ;
[...] }
[...]

For Motif Toolkit widgets, the reason name must be a built-in reason name. For a user-defined widget, you can use a reason name that you previously specified using the **REASON** function. If you use a built-in reason in an object definition, the UIL compiler ensures that reason is supported by the type of object you are defining. Appendix B shows which reasons each object supports.

If the same reason appears more than once in a callbacks list, the last entry referring to that name supersedes all previous entries using the same reason, and the UIL compiler issues a diagnostic message.

If you specify a named value for the procedure argument (callback tag), the data type of the value must match the type specified for the callback tag in the corresponding procedure declaration. When specifying a widget name as a procedure value expression you must also specify the type of the widget and a space before the name of the widget.

Because the UIL compiler produces a UID file rather than an object module (.o), the binding of the UIL name to the address of the entry point to the procedure is not done by the loader, but is established at run time with the MRM function **MrmRegisterNames**. You call this function before fetching any objects, giving it both the UIL names and the procedure addresses of each callback. The name you register with MRM in the application program must match the name you specified for the procedure in the UIL module.

Each callback procedure receives three arguments. The first two arguments have the same form for each callback. The form of the third argument varies from object to object.

The first argument is the address of the data structure maintained by the Motif Toolkit for this object instance. This address is called the widget ID for this object.

The second argument is the address of the value you specified in the callbacks list for this procedure. If you do not specify an argument, the address is NULL.

The third argument is the reason name you specified in the callbacks list.

Controls List Structure

A controls list defines which objects are children of, or controlled by, a particular object. Each entry in a controls list has the following syntax:

LIST
list_name **: CONTROLS {**
[*child_name*] [**MANAGED** | **UNMANAGED**] *object_definition*;
[...] }
[...]

If you specify the keyword **MANAGED** at run time, the object is created and managed; if you specify **UNMANAGED** at run time, the object is only created. Objects are managed by default.

You can use *child_name* to specify resources for the automatically created children of a particular control. Names for automatically created children are formed by appending **Xm_** to the name of the child widget. This name is specified in the documentation for the parent widget.

Unlike the arguments list and the callbacks list, a controls list entry that is identical to a previous entry does not supersede the previous entry. At run time, each controls list entry causes a child to be created when the parent is created. If the same object definition is used for multiple children, multiple instances of the child are created at run time. See Appendix B for a list of which widget types can be controlled by which other widget types.

Procedures List Structure

You can specify multiple procedures for a callback reason in UIL by defining a procedures list. Just as with other list types, procedures lists can be defined in-line or in a list section and referenced by name.

If you define a reason more than once (for example, when the reason is defined both in a referenced procedures list and in the callbacks list for the object), previous definitions are overridden by the latest definition. The syntax for a procedures list is as follows:

LIST
list_name **: PROCEDURES {**
procedure_name [([*value_expression*])];
[...] }
[...]

When specifying a widget name as a procedure value expression you must also specify the type of the widget and a space before the name of the widget.

Object Sections

An object section consists of the keyword **OBJECT** followed by a sequence of object declarations. It has the following syntax:

OBJECT *object_name* **:**
[**EXPORTED** | **PRIVATE** | **IMPORTED**] *object_type*
[**PROCEDURE** *creation_function*]
[*object_name* [**WIDGET** | **GADGET**] | { *list_definitions* }]

Use an object declaration to define the objects that are to be stored in the UID file. You can reference the object name in declarations that occur elsewhere in the UIL module in any context where an object name can be used (for example, in a controls list, as a symbolic reference to a widget ID, or as the *tag_value* argument for a callback procedure). Objects can be forward referenced; that is, you can declare an object name after you reference it. All references to an object name must be consistent with the type of the object, as specified in the object declaration. You can specify an object as exported, imported, or private.

The object definition can contain a sequence of lists that define the arguments, hierarchy, and callbacks for the widget. You can specify only one list of each type for an object. When you declare a user-defined widget, you must include a reference to the widget creation function for the user-defined widget.

Use the **GADGET** or **WIDGET** keyword to specify the object type or to override the default variant for this object type. You can use the Motif Toolkit name of an object type that has a gadget variant (for example, **XmLabelGadget**) as an attribute of an object declaration. The *object_type* can be any object type, including gadgets. You need to specify the **GADGET** or **WIDGET** keyword only in the declaration of an object, not when you reference the object. You cannot specify the **GADGET** or **WIDGET** keyword for a user-defined object; user-defined objects are always widgets.

Identifier sections

The identifier section allows you to define an identifier, a mechanism that achieves run-time binding of data to names that appear in a UIL module. The identifier section consists of the reserved keyword **IDENTIFIER**, followed by a list of names, each name followed by a semicolon.

IDENTIFIER *identifier_name*; [...;]

You can later use these names in the UIL module as either the value of an argument to a widget or the tag value to a callback procedure. At run time, you use the MRM functions **MrmRegisterNames** and **MrmRegisterNamesInHierarchy** to bind the identifier name with the data (or, in the case of callbacks, with the address of the data) associated with the identifier.

Each UIL module has a single name space; therefore, you cannot use a name you used for a value, object, or procedure as an identifier name in the same module.

The UIL compiler does not do any type checking on the use of identifiers in a UIL module. Unlike a UIL value, an identifier does not have a UIL type associated with it. Regardless of what particular type a widget argument or callback procedure tag is defined to be, you can use an identifier in that context instead of a value of the corresponding type.

To reference these identifier names in a UIL module, you use the name of the identifier wherever you want its value to be used.

Include directives

The include directive incorporates the contents of a specified file into a UIL module. This mechanism allows several UIL modules to share common definitions. The syntax for the include directive is as follows:

INCLUDE FILE *file_name* ;

The UIL compiler replaces the include directive with the contents of the include file and processes it as if these contents had appeared in the current UIL source file.

You can nest include files; that is, an include file can contain include directives. The UIL compiler can process up to 100 references (including the file containing the UIL module). Therefore, you can include up to 99 files in a single UIL module, including nested files. Each time a file is opened counts as a reference, so including the same file twice counts as two references.

The character expression is a file specification that identifies the file to be included. The rules for finding the specified file are similar to the rules for finding header, or **.h** files using the include directive, **#include**, with a quoted string in C. The UIL uses the **-I** option for specifying a search directory for include files.

- If you do not supply a directory, the UIL compiler searches for the include file in the directory of the main source file.
- If the compiler does not find the include file there, the compiler looks in the same directory as the source file.
- If you supply a directory, the UIL compiler searches only that directory for the file.

Names and Strings

Names can consist of any of the characters A to Z, a to z, 0 to 9, $ (dollar sign), and _ (underscore). Names cannot begin with a digit (0 to 9). The maximum length of a name is 31 characters.

UIL gives you a choice of either case-sensitive or case-insensitive names through a clause in the **MODULE** header. For example, if names are case sensitive, the names "sample" and "Sample" are distinct from each other. If names are case insensitive, these names are treated as the same name and can be used interchangeably. By default, UIL assumes names are case sensitive.

In **CASE-INSENSITIVE** mode, the compiler outputs all names in the UID file in uppercase form. In **CASE-SENSITIVE** mode, names appear in the UIL file exactly as they appear in the source.

The following table lists the reserved keywords, which are not available for defining programmer defined names.

Reserved Keywords			
ARGUMENTS	CALLBACKS	CONTROLS	END
EXPORTED	FALSE	GADGET	IDENTIFIER
INCLUDE	LIST	MODULE	OFF
ON	OBJECT	PRIVATE	PROCEDURE
PROCEDURES	TRUE	VALUE	WIDGET

The UIL unreserved keywords are described in the following list and table. These keywords can be used as programmer defined names; however, if you use any keyword as a name, you cannot use the UIL-supplied usage of that keyword.

- Built-in argument names (for example, **XmNx**, **XmNheight**)
- Built-in reason names (for example, **XmNactivateCallback**, **XmNhelpCallback**)
- Character set names (for example, **ISO_LATIN1**, **ISO_HEBREW_LR**)
- Constant value names (for example, **XmMENU_OPTION**, **XmBROWSE_SELECT**)
- Object types (for example, **XmPushButton**, **XmBulletinBoard**)

Unreserved Keywords		
ANY	ARGUMENT	ASCIZ_STRING_TABLE
ASCIZ_TABLE	BACKGROUND	BOOLEAN
CASE_INSENSITIVE	CASE_SENSITIVE	CHARACTER_SET
COLOR	COLOR_TABLE	COMPOUND_STRING
COMPOUND_STRING_TABLE	FILE	FLOAT
FONT	FONT_TABLE	FONTSET
FOREGROUND	ICON	IMPORTED
INTEGER	INTEGER_TABLE	KEYSYM
MANAGED	NAMES	OBJECTS
REASON	RGB	RIGHT_TO_LEFT
SINGLE_FLOAT	STRING	STRING_TABLE
TRANSLATION_TABLE	UNMANAGED	USER_DEFINED
VERSION	WIDE_CHARACTER	WIDGET
XBITMAPFILE		

String literals can be composed of the uppercase and lowercase letters, digits, and punctuation characters. Spaces, tabs, and comments are special elements in the language. They are a means of delimiting other elements, such as two names. One or more of these elements can appear before or after any other element in the language. However, spaces, tabs, and comments that appear in string literals are treated as character sequences rather than delimiters.

Data Types

UIL provides literals for several of the value types it supports. Some of the value types are not supported as literals (for example, pixmaps and string tables). You can specify values for these types by using functions described in the **Functions** section. UIL directly supports the following literal types:

- String literal
- Integer literal
- Boolean literal
- Floating-point literal

UIL also includes the data type **ANY**, which is used to turn off compile time checking of data types.

String Literals

A string literal is a sequence of zero or more 8-bit or 16-bit characters or a combination delimited by ' (single quotation marks) or " (double quotation marks). String literals can also contain multibyte characters delimited with double quotation marks. String literals can be no more than 2000 characters long.

A single-quoted string literal can span multiple source lines. To continue a single-quoted string literal, terminate the continued line with a \ (backslash). The literal continues with the first character on the next line.

Double-quoted string literals cannot span multiple source lines. (Because double-quoted strings can contain escape sequences and other special characters, you cannot use the backslash character to designate continuation of the string.) To build a string value that must span multiple source lines, use the concatenation operator described later in this section.

The syntax of a string literal is one of the following:

'[*character_string*]' [#*char_set*]"[*character_string*]"

Both string forms associate a character set with a string value. UIL uses the following rules to determine the character set and storage format for string literals:

- A string declared as '*string*' is equivalent to #*cur_charset*"*string*", where *cur_charset* will be the codeset portion of the value of the **LANG** environment variable if it is set or the value of **XmFALLBACK_CHARSET** if **LANG** is not set or has no codeset component. By default, **XmFALLBACK_CHARSET** is **ISO8859-1** (equivalent to **ISO_LATIN1**), but vendors may define a different default.
- A string declared as "*string*" is equivalent to #*char_set*"*string*" if you specified *char_set* as the default character set for the module. If no default character set has been specified for the module, then if the **-s** option is provided to the **uil** command or the **use_setlocale_flag** is set for the callable compiler, **Uil()**, the string will be interpreted to be a string in the current locale. This means that the string is parsed in the locale of the user by calling **setlocale**, its charset is **XmFONTLIST_DEFAULT_TAG**, and that if the string is converted to a compound string, it is stored as a locale encoded text segment. Otherwise, "*string*" is equivalent to #*cur_charset*"*string*", where *cur_charset* is interpreted as described for single quoted strings.
- A string of the form "*string*" or #*char_set*"*string*" is stored as a null-terminated string.

The following table lists the character sets supported by the UIL compiler for string literals. Note that several UIL names map to the same character set. In some cases, the UIL name influences how string literals are read. For example, strings identified by a UIL character set name ending in **_LR** are read left-to-right. Names that end in a different number reflect different fonts (for example, ISO_LATIN1 or ISO_LATIN6). All character sets in this table are represented by 8 bits.

Supported Character Sets	
UIL Name	**Description**
ISO_LATIN1	GL: ASCII, GR: Latin-1 Supplement
ISO_LATIN2	GL: ASCII, GR: Latin-2 Supplement
ISO_ARABIC	GL: ASCII, GR: Latin-Arabic Supplement
ISO_LATIN6	GL: ASCII, GR: Latin-Arabic Supplement
ISO_GREEK	GL: ASCII, GR: Latin-Greek Supplement
ISO_LATIN7	GL: ASCII, GR: Latin-Greek Supplement
ISO_HEBREW	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_LATIN8	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_HEBREW_LR	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_LATIN8_LR	GL: ASCII, GR: Latin-Hebrew Supplement
JIS_KATAKANA	GL: JIS Roman, GR: JIS Katakana

Following are the parsing rules for each of the character sets:

All character sets

Character codes in the range 00-1F, 7F, and 80-9F are control characters including both bytes of 16-bit characters. The compiler flags these as illegal characters.

ISO_LATIN1 ISO_LATIN2 ISO_ARABIC ISO_LATIN3 ISO_GREEK ISO_LATIN4

These sets are parsed from left to right. The escape sequences for null-terminated strings are also supported by these character sets.

ISO_HEBREW ISO_LATIN8

These sets are parsed from right to left; for example, the string **#ISO_HEBREW"012345"** generates a primitive string "543210" with character set **ISO_HEBREW**. A DDIS descriptor for such a string has this segment marked as being right-to-left. The escape sequences for null-terminated strings are also supported by these character sets, and the characters that compose the escape sequences are in left-to-right order. For example, you would enter \n, not n\.

ISO_HEBREW_LR ISO_LATIN8_LR

These sets are parsed from left to right; for example, the string **#ISO_HEBREW_LR"012345"** generates a primitive string "012345" with character set **ISO_HEBREW**. A DDIS descriptor

for such a string marks this segment as being left-to-right. The escape sequences for null-terminated strings are also supported by these character sets.

JIS_KATAKANA
This set is parsed from left to right. The escape sequences for null-terminated strings are also supported by this character set. Note that the \ (backslash) may be displayed as a yen symbol.

In addition to designating parsing rules for strings, character set information remains an attribute of a compound string. If the string is included in a string consisting of several concatenated segments, the character set information is included with that string segment. This gives the Motif Toolkit the information it needs to decipher the compound string and choose a font to display the string.

For an application interface displayed only in English, UIL lets you ignore the distinctions between the two uses of strings. The compiler recognizes by context when a string must be passed as a null-terminated string or as a compound string.

The UIL compiler recognizes enough about the various character sets to correctly parse string literals. The compiler also issues errors if you use a compound string in a context that supports only null-terminated strings.

Since the character set names are keywords, you must put them in lowercase if case-sensitive names are in force. If names are case insensitive, character set names can be uppercase, lowercase, or mixed case.

In addition to the built-in character sets recognized by UIL, you can define your own character sets with the **CHARACTER_SET** function. You can use the **CHARACTER_SET** function anywhere a character set can be specified.

String literals can contain characters with the eighth (high-order) bit set. You cannot type control characters (00-1F, 7F, and 80-9F) directly in a single-quoted string literal. However, you can represent these characters with escape sequences. The following list shows the escape sequences for special characters.

\b Backspace

\f Form-feed

\n Newline

\r Carriage return

\t Horizontal tab

\v Vertical tab

\' Single quotation mark

\"	Double quotation mark
\\	Backslash
integer\	Character whose internal representation is given by *integer* (in the range 0 to 255 decimal)

Note that escape sequences are processed literally in strings that are parsed in the current locale (localized strings).

The UIL compiler does not process newline characters in compound strings. The effect of a newline character in a compound string depends only on the character set of the string, and the result is not guaranteed to be a multiline string.

Compound String Literals

A compound string consists of a string of 8-bit, 16-bit, or multibyte characters, a named character set, and a writing direction. Its UIL data type is **compound_string**.

The writing direction of a compound string is implied by the character set specified for the string. You can explicitly set the writing direction for a compound string by using the **COMPOUND_STRING** function.

A compound string can consist of a sequence of concatenated compound strings, null-terminated strings, or a combination of both, each of which can have a different character set property and writing direction. Use the concatenation operator & (ampersand) to create a sequence of compound strings.

Each string in the sequence is stored, including the character set and writing direction information.

Generally, a string literal is stored in the UID file as a compound string when the literal consists of concatenated strings having different character sets or writing directions, or when you use the string to specify a value for an argument that requires a compound string value. If you want to guarantee that a string literal is stored as a compound string, you must use the **COMPOUND_STRING** function.

Data Storage Consumption for String Literals

The way a string literal is stored in the UID file depends on how you declare and use the string. The UIL compiler automatically converts a null-terminated string to a compound string if you use the string to specify the value of an argument that requires a compound string. However, this conversion is costly in terms of storage consumption.

PRIVATE, **EXPORTED**, and **IMPORTED** string literals require storage for a single allocation when the literal is declared; thereafter, storage is required for each reference to the literal. Literals declared in-line require storage for both an allocation and a reference.

The following table summarizes data storage consumption for string literals. The storage requirement for an allocation consists of a fixed portion and a variable portion. The fixed portion of an allocation is roughly the same as the storage requirement for a reference (a few bytes). The storage consumed by the variable portion depends on the size of the literal value (that is, the length of the string). To conserve storage space, avoid making string literal declarations that result in an allocation per use.

Data Storage Consumption for String Literals

Declaration	Data Type	Used As	Storage Requirements Per Use
In-line	Null-terminated	Null-terminated	An allocation and a reference (within the module)
Private	Null-terminated	Null-terminated	A reference (within the module)
Exported	Null-terminated	Null-terminated	A reference (within the UID hierarchy)
Imported	Null-terminated	Null-terminated	A reference (within the UID hierarchy)
In-line	Null-terminated	Compound	An allocation and a reference (within the module)
Private	Null-terminated	Compound	An allocation and a reference (within the module)
Exported	Null-terminated	Compound	A reference (within the UID hierarchy)
Imported	Null-terminated	Compound	A reference (within the UID hierarchy)
In-line	Compound	Compound	An allocation and a reference (within the module)
Private	Compound	Compound	A reference (within the module)
Exported	Compound	Compound	A reference (within the UID hierarchy)
Imported	Compound	Compound	A reference (within the UID hierarchy)

Integer Literals

An integer literal represents the value of a whole number. Integer literals have the form of an optional sign followed by one or more decimal digits. An integer literal must not contain embedded spaces or commas.

Integer literals are stored in the UID file as long integers. Exported and imported integer literals require a single allocation when the literal is declared; thereafter, a few bytes of storage are required for each reference to the literal. Private integer literals and those declared in-line require allocation and reference storage per use. To conserve storage space, avoid making integer literal declarations that result in an allocation per use.

The following table shows data storage consumption for integer literals.

Data Storage Consumption for Integer Literals	
Declaration	**Storage Requirements Per Use**
In-line	An allocation and a reference (within the module)
Private	An allocation and a reference (within the module)
Exported	A reference (within the UID hierarchy)
Imported	A reference (within the UID hierarchy)

Boolean Literal

A Boolean literal represents the value True (reserved keyword **TRUE** or **On**) or False (reserved keyword **FALSE** or **Off**). These keywords are subject to case-sensitivity rules.

In a UID file, **TRUE** is represented by the integer value 1 and **FALSE** is represented by the integer value 0 (zero).

Data storage consumption for Boolean literals is the same as that for integer literals.

Floating-Point Literal

A floating-point literal represents the value of a real (or float) number. Floating-point literals have the following form:

[+|-][*integer*].*integer*[**E**|**e**[+|-]*exponent*]

For maximum portability, a floating-point literal can represent values in the range 1.0E-37 to 1.0E+37 with at least 6 significant digits. On many machines this range will be wider, with more significant digits. A floating-point literal must not contain embedded spaces or commas.

Floating-point literals are stored in the UID file as double-precision, floating-point numbers. The following table gives examples of valid and invalid floating-point notation for the UIL compiler.

Floating Point Literals	
Valid Floating-Point Literals	**Invalid Floating-Point Literals**
1.0	1e1 (no decimal point)
.1	E-1 (no decimal point or digits)
3.1415E-2 (equals .031415)	2.87 e6 (embedded blanks)
-6.29e7 (equals -62900000)	2.0e100 (out of range)

Data storage consumption for floating-point literals is the same as that for integer literals.

The purpose of the **ANY** data type is to shut off the data-type checking feature of the UIL compiler. You can use the **ANY** data type for the following:

- Specifying the type of a callback procedure tag
- Specifying the type of a user-defined argument

You can use the **ANY** data type when you need to use a type not supported by the UIL compiler or when you want the data-type restrictions imposed by the compiler to be relaxed. For example, you might want to define a widget having an argument that can accept different types of values, depending on run-time circumstances.

If you specify that an argument takes an **ANY** value, the compiler does not check the type of the value specified for that argument; therefore, you need to take care when specifying a value for an argument of type **ANY**. You could get unexpected results at run time if you pass a value having a data type that the widget does not support for that argument.

Expressions

UIL includes compile-time value expressions. These expressions can contain references to other UIL values, but cannot be forward referenced.

The following table lists the set of operators in UIL that allow you to create integer, real, and Boolean values based on other values defined with the UIL module. In the table, a precedence of 1 is the highest.

Valid Operators			
Operator	**Operand Types**	**Meaning**	**Precedence**
~	Boolean	NOT	1
	integer	One's complement	
-	float	Negate	1
	integer	Negate	
+	float	NOP	1
	integer	NOP	
*	float,float	Multiply	2
	integer,integer	Multiply	
/	float,float	Divide	2
	integer,integer	Divide	
+	float,float	Add	3
	integer,integer	Add	
-	float,float	Subtract	3
	integer,integer	Subtract	
>>	integer,integer	Shift right	4
<<	integer,integer	Shift left	4
&	Boolean,Boolean	AND	5
	integer,integer	Bitwise AND	
	string,string	Concatenate	
|	Boolean,Boolean	OR	6
	integer,integer	Bitwise OR	
^	Boolean,Boolean	XOR	6
	integer,integer	Bitwise XOR	

A string can be either a single compound string or a sequence of compound strings. If the two concatenated strings have different properties (such as writing direction or character set), the result of the concatenation is a multisegment compound string.

The string resulting from the concatenation is a null-terminated string unless one or more of the following conditions exists:

- One of the operands is a compound string
- The operands have different character set properties
- The operands have different writing directions

Then the resulting string is a compound string. You cannot use imported or exported values as operands of the concatenation operator.

The result of each operator has the same type as its operands. You cannot mix types in an expression without using conversion routines.

You can use parentheses to override the normal precedence of operators. In a sequence of unary operators, the operations are performed in right-to-left order. For example, **- + -A** is equivalent to **-(+(-A))**. In a sequence of binary operators of the same precedence, the operations are performed in left-to-right order. For example, **A*B/C*D** is equivalent to **((A*B)/C)*D**.

A value declaration gives a value a name. You cannot redefine the value of that name in a subsequent value declaration. You can use a value containing operators and functions anywhere you can use a value in a UIL module. You cannot use imported values as operands in expressions.

Several of the binary operators are defined for multiple data types. For example, the operator for multiplication (*) is defined for both floating-point and integer operands.

For the UIL compiler to perform these binary operations, both operands must be of the same type. If you supply operands of different data types, the UIL compiler automatically converts one of the operands to the type of the other according to the following conversions rules:

- If the operands are an integer and a Boolean, the Boolean is converted to an integer.
- If the operands are an integer and a floating-point, the integer is converted to an floating-point.
- If the operands are a floating-point and a Boolean, the Boolean is converted to a floating-point.

You can also explicitly convert the data type of a value by using one of the conversion functions **INTEGER**, **FLOAT** or **SINGLE_FLOAT**.

Functions

UIL provides functions to generate the following types of values:

- Character sets
- Keysyms
- Colors
- Pixmaps
- Single-precision, floating-point numbers
- Double-precision, floating-point numbers
- Fonts

- Fontsets
- Font tables
- Compound strings
- Compound string tables
- ASCIZ (null-terminated) string tables
- Wide character strings
- Widget class names
- Integer tables
- Arguments
- Reasons
- Translation tables

Remember that all examples in the following sections assume case-insensitive mode. Keywords are shown in uppercase letters to distinguish them from user-specified names, which are shown in lowercase letters. This use of uppercase letters is not required in case-insensitive mode. In case-sensitive mode, keywords must be in lowercase letters.

CHARACTER_SET(*string_expression*[, property[, ...]])

You can define your own character sets with the **CHARACTER_SET** function. You can use the **CHARACTER_SET** function anywhere a character set can be specified.

The result of the **CHARACTER_SET** function is a character set with the name *string_expression* and the properties you specify. *string_expression* must be a null-terminated string. You can optionally include one or both of the following clauses to specify properties for the resulting character set:

RIGHT_TO_LEFT = *boolean_expression*
SIXTEEN_BIT = *boolean_expression*

The **RIGHT_TO_LEFT** clause sets the default writing direction of the string from right to left if *boolean_expression* is True, and right to left otherwise.

The **SIXTEEN_BIT** clause allows the strings associated with this character set to be interpreted as 16-bit characters if *boolean_expression* is True, and 8-bit characters otherwise.

KEYSYM(*string_literal*)

The **KEYSYM** function is used to specify a keysym for a mnemonic resource. *string_literal* must contain exactly one character.

COLOR(*string_expression*[,**FOREGROUND**|**BACKGROUND**])

The **COLOR** function supports the definition of colors. Using the **COLOR** function, you can designate a value to specify a color and then use that value for arguments requiring a color value. The string expression names the color you want to define; the optional keywords **FOREGROUND** and **BACKGROUND** identify how the color is to be displayed on a monochrome device when the color is used in the definition of a color table.

The UIL compiler does not have built-in color names. Colors are a server-dependent attribute of an object. Colors are defined on each server and may have different red-green-blue (RGB) values on each server. The string you specify as the color argument must be recognized by the server on which your application runs.

In a UID file, UIL represents a color as a character string. MRM calls X translation routines that convert a color string to the device-specific pixel value. If you are running on a monochrome server, all colors translate to black or white. If you are on a color server, the color names translate to their proper colors if the following conditions are met:

- The color is defined.
- The color map is not yet full.

If the color map is full, even valid colors translate to black or white (foreground or background).

Interfaces do not, in general, specify colors for widgets, so that the selection of colors can be controlled by the user through the **.Xdefaults** file.

To write an application that runs on both monochrome and color devices, you need to specify which colors in a color table (defined with the **COLOR_TABLE** function) map to the background and which colors map to the foreground. UIL lets you use the **COLOR** function to designate this mapping in the definition of the color. The following example shows how to use the **COLOR** function to map the color red to the background color on a monochrome device:

```
VALUE c: COLOR ( 'red',BACKGROUND );
```

The mapping comes into play only when the MRM is given a color and the application is to be displayed on a monochrome device. In this case, each color is considered to be in one of the following three categories:

- The color is mapped to the background color on the monochrome device.
- The color is mapped to the foreground color on the monochrome device.
- Monochrome mapping is undefined for this color.

If the color is mapped to the foreground or background color, MRM substitutes the foreground or background color, respectively. If you do not specify the monochrome mapping for a color, MRM passes the color string to the Motif Toolkit for mapping to the foreground or background color.

RGB(*red_integer, green_integer, blue_integer*)

The three integers define the values for the red, green, and blue components of the color, in that order. The values of these components can range from 0 to 65,535, inclusive.

In a UID file, UIL represents an **RGB** value as three integers. MRM calls X translation routines that convert the integers to the device-specific pixel value. If you are running on a monochrome server, all colors translate to black or white. If you are on a color server, **RGB** values translate to their proper colors if the colormap is not yet full. If the colormap is full, values translate to black or white (foreground or background).

COLOR_TABLE(*color_expression='character'*[,...])

The color expression is a previously defined color, a color defined in line with the **COLOR** function, or the phrase **BACKGROUND COLOR** or **FOREGROUND COLOR**. The character can be any valid UIL character.

The **COLOR_TABLE** function provides a device-independent way to specify a set of colors. The **COLOR_TABLE** function accepts either previously defined UIL color names or in-line color definitions (using the **COLOR** function). A color table must be private because its contents must be known by the UIL compiler to construct an icon. The colors within a color table, however, can be imported, exported, or private.

The single letter associated with each color is the character you use to represent that color when creating an icon. Each letter used to represent a color must be unique within the color table.

ICON([**COLOR_TABLE=***color_table_name*,] *row*[,...)

color-table-name must refer to a previously defined color table, and *row* is a character expression giving one row of the icon.

The **ICON** function describes a rectangular icon that is *x* pixels wide and *y* pixels high. The strings surrounded by single quotation marks describe the icon. Each string represents a row in the icon; each character in the string represents a pixel.

The first row in an icon definition determines the width of the icon. All rows must have the same number of characters as the first row. The height of the icon is dictated by the number of rows.

The first argument of the **ICON** function (the color table specification) is optional and identifies the colors that are available in this icon. By using the single letter associated with each color, you can specify the color of each pixel in the icon. The icon must be constructed of characters defined in the specified color table.

A default color table is used if you omit the argument specifying the color table. To make use of the default color table, the rows of your icon must contain only spaces and asterisks. The default color table is defined as follows:

```
COLOR_TABLE( BACKGROUND COLOR = ' ', FOREGROUND COLOR = '*' )
```

You can define other characters to represent the background color and foreground color by replacing the space and asterisk in the **BACKGROUND COLOR** and **FOREGROUND COLOR** clauses shown in the previous statement. You can specify icons as private, imported, or exported. Use the MRM function **MrmFetchIconLiteral** to retrieve an exported icon at run time.

XBITMAPFILE(*string_expression*)

The **XBITMAPFILE** function is similar to the **ICON** function in that both describe a rectangular icon that is *x* pixels wide and *y* pixels high. However, **XBITMAPFILE** allows you to specify an external file containing the definition of an X bitmap, whereas all **ICON** function definitions must be coded directly within UIL. X bitmap files can be generated by many different X applications. UIL reads these files through the **XBITMAPFILE** function, but does not support creation of these files. The X bitmap file specified as the argument to the **XBITMAPFILE** function is read at application run time by MRM.

The **XBITMAPFILE** function returns a value of type **pixmap** and can be used anywhere a pixmap data type is expected.

SINGLE_FLOAT(*real_number_literal*)

The **SINGLE_FLOAT** function lets you store floating-point literals in UIL files as single-precision, floating-point numbers. Single-precision floating-point numbers can often be stored using less memory than double-precision, floating-point numbers. The *real_number_literal* can be either an integer

literal or a floating-point literal. A value defined using this function cannot be used in an arithmetic expression.

FLOAT(*real_number_literal*)

The **FLOAT** function lets you store floating-point literals in UIL files as double-precision, floating-point numbers. The *real_number_literal* can be either an integer literal or a floating-point literal.

FONT(*string_expression*[, **CHARACTER_SET**=*char_set*])

You define fonts with the **FONT** function. Using the **FONT** function, you designate a value to specify a font and then use that value for arguments that require a font value. The UIL compiler has no built-in fonts.

Each font makes sense only in the context of a character set. The **FONT** function has an additional parameter to let you specify the character set for the font. This parameter is optional; if you omit it, the default character set depends on the value of the **LANG** environment variable if it is set, or on the value of **XmFALLBACK_CHARSET** if **LANG** is not set.

string_expression specifies the name of the font and the clause **CHARACTER_SET** = *char_set* specifies the character set for the font. The string expression used in the **FONT** function cannot be a compound string.

FONTSET(*string_expression*[,...][, **CHARACTER_SET**=*charset*])

You define fontsets with the **FONTSET** function. Using the **FONTSET** function, you designate a set of values to specify fonts and then use those values for arguments that require a fontset. The UIL compiler has no built-in fonts.

Each font makes sense only in the context of a character set. The **FONTSET** function has an additional parameter to let you specify the character set for the font. This parameter is optional; if you omit it, the default character set depends on the value of the **LANG** environment variable if it is set, or on the value of **XmFALLBACK_CHARSET** if **LANG** is not set.

The string expression specifies the name of the font and the clause **CHARACTER_SET** = *char_set* specifies the character set for the font. The string expression used in the **FONTSET** function cannot be a compound string.

FONT_TABLE(*font_expression*[,...])

A font table is a sequence of pairs of fonts and character sets. At run time, when an object needs to display a string, the object scans the font table for the character set that matches the character set of the string to be displayed. UIL provides the **FONT_TABLE** function to let you supply such an argument. *font_expression* is created with the **FONT** and **FONTSET** functions.

If you specify a single font value to specify an argument that requires a font table, the UIL compiler automatically converts a font value to a font table.

COMPOUND_STRING(*string_expression*[,*property*[,...]])

Use the **COMPOUND_STRING** function to set properties of a null-terminated string and to convert it into a compound string. The properties you can set are the character set, writing direction, and separator.

The result of the **COMPOUND_STRING** function is a compound string with the string expression as its value. You can optionally include one or more of the following clauses to specify properties for the resulting compound string:

CHARACTER_SET = *character_set*
RIGHT_TO_LEFT = *boolean_expression*
SEPARATE = *boolean_expression*

The **CHARACTER_SET** clause specifies the character set for the string. If you omit the **CHARACTER_SET** clause, the resulting string has the same character set as *string_expression.*

The **RIGHT_TO_LEFT** clause sets the writing direction of the string from right to left if *boolean_expression* is True, and left to right otherwise. Specifying this argument does not cause the value of the string expression to change. If you omit the **RIGHT_TO_LEFT** argument, the resulting string has the same writing direction as *string_expression.*

The **SEPARATE** clause appends a separator to the end of the compound string if *boolean_expression* is True. If you omit the **SEPARATE** clause, the resulting string does not have a separator.

You cannot use imported or exported values as the operands of the **COMPOUND_STRING** function.

COMPOUND_STRING_TABLE(*string_expression*[,...])

A compound string table is an array of compound strings. Objects requiring a list of string values, such as the **XmNitems** and **XmNselectedItems** arguments for the list widget, use string table values. The **COMPOUND_STRING_TABLE** function builds the values for these two arguments of the list widget. The **COMPOUND_STRING_TABLE** function generates a value of type **string_table**. The name **STRING_TABLE** is a synonym for **COMPOUND_STRING_TABLE**.

The strings inside the string table can be simple strings, which the UIL compiler automatically converts to compound strings.

ASCIZ_STRING_TABLE(*string_expression*[,...])
An ASCIZ string table is an array of ASCIZ (null-terminated) string values separated by commas. This function allows you to pass more than one ASCIZ string as a callback tag value. The **ASCIZ_STRING_TABLE** function generates a value of type **asciz_table**. The name **ASCIZ_TABLE** is a synonym for **ASCIZ_STRING_TABLE**.

WIDE_CHARACTER(*string_expression*)
Use the **WIDE_CHARACTER** function to generate a wide character string from null-terminated string in the current locale.

CLASS_REC_NAME(*string_expression*)
Use the **CLASS_REC_NAME** function to generate a widget class name. For a widget class defined by the toolkit, the string argument is the name of the class. For a user-defined widget, the string argument is the name of the creation routine for the widget.

INTEGER_TABLE(*integer_expression*[,...])
An integer table is an array of integer values separated by commas. This function allows you to pass more than one integer per callback tag value. The **INTEGER_TABLE** function generates a value of type **integer_table**.

ARGUMENT(*string_expression*[, *argument_type*])
The **ARGUMENT** function defines the arguments to a user-defined widget. Each of the objects that can be described by UIL permits a set of arguments, listed in Appendix B. For example, **XmNheight** is an argument to most objects and has an integer data type. To specify height for a user-defined widget, you can use the built-in argument name **XmNheight**, and specify an integer value when you declare the user-defined widget. You do not use the **ARGUMENT** function to specify arguments that are built into the UIL compiler.

The *string_expression* name is the name the UIL compiler uses for the argument in the UID file. *argument_type* is the type of value that can be associated with the argument. If you omit the second argument, the default type is **ANY** and no value type checking occurs. Use one of the following keywords to specify the argument type:

- ANY
- ASCIZ_TABLE
- BOOLEAN
- COLOR

- COLOR_TABLE
- COMPOUND_STRING
- FLOAT
- FONT
- FONT_TABLE
- FONTSET
- ICON
- INTEGER
- INTEGER_TABLE
- REASON
- SINGLE_FLOAT
- STRING
- STRING_TABLE
- TRANSLATION_TABLE
- WIDE_CHARACTER
- WIDGET

You can use the **ARGUMENT** function to allow the UIL compiler to recognize extensions to the Motif Toolkit. For example, an existing widget may accept a new argument. Using the **ARGUMENT** function, you can make this new argument available to the UIL compiler before the updated version of the compiler is released.

REASON(*string_expression*)

The **REASON** function is useful for defining new reasons for user-defined widgets.

Each of the objects in the Motif Toolkit defines a set of conditions under which it calls a user-defined function. These conditions are known as callback reasons. The user-defined functions are termed callback procedures. In a UIL module, you use a callbacks list to specify which user-defined functions are to be called for which reasons.

Appendix B lists the callback reasons supported by the Motif Toolkit objects.

When you declare a user-defined widget, you can define callback reasons for that widget using the **REASON** function. The string expression specifies the argument name stored in the UID file for the reason. This reason name is supplied to the widget creation routine at run time.

TRANSLATION_TABLE(*string_expression*[,...])

Each of the Motif Toolkit widgets has a translation table that maps X events (for example, mouse button 1 being pressed) to a sequence of actions. Through widget arguments, such as the common translations argument, you can specify an alternate set of events or actions for a particular widget. The **TRANSLATION_TABLE** function creates a translation table that can be used as the value of an argument that is of the data type **translation_table**.

You can use one of the following translation table directives with the **TRANSLATION_TABLE** function: **#override**, **#augment**, or **#replace**. The default is **#replace**. If you specify one of these directives, it must be the first entry in the translation table.

The **#override** directive causes any duplicate translations to be ignored. For example, if a translation for **<Btn1Down>** is already defined in the current translations for a PushButton, the translation defined by *new_translations* overrides the current definition. If the **#augment** directive is specified, the current definition takes precedence. The **#replace** directive replaces all current translations with those specified in the **XmNtranslations** resource.

Related Information

uil(1X), **Uil(3X)**

WML—The widget meta-language file format for creating UIL compilers

Description

The widget meta-language facility (WML) is used to generate the components of the user interface language (UIL) compiler that can change depending on the widget set. Using WML you can add support in UIL for new widgets to the OSF/Motif widget set or for a totally new widget set.

File Format

WML files are ASCII files that you can modify with any standard text editor. They are accessed in the **tools/wml** directory by WML. By convention WML files have the suffix **.wml**. The Motif widget set is described in the **motif.wml** file. This is also the default WML file when using the WML facility.

When adding new widgets or changing widget characteristics, you should start with a copy of the **motif.wml** file. If you are creating a new widget set for use with UIL, you should start from scratch. In either case the **motif.wml** file is a good example of WML syntax, and you should familiarize yourself with it before writing your own WML file.

WML files have a simple syntax, similar in structure to UIL. It is made up of the following elements:

- Comments
- Data Type Definitions
- Character Set Definitions
- Enumeration Set Definitions
- Control List Definitions
- Class Definitions
- Child Definitions
- Resource Definitions

You can use space, tabs, or newlines anywhere in the syntax, as long as you do not split up keywords or strings, except that comments end at a newline. The order of elements is not important to the syntax.

This description uses the following additional conventions to describe the syntax of the widget meta-language:

[ ]	Indicates optional elements
...	Indicates where an element of syntax can be repeated
|	Indicates a choice among multiple items

Comments

You can include comments in the WML file. Comments have the following syntax:

[*any.element*]!*any.comment*

Comments begin with an exclamation point and extend to the end of the line. A comment can begin on a line by itself or follow any part of another element. A comment does not change the meaning of any other element. For example:

```
!This is a comment
!  that spans two lines.
DataType    !This is a comment following code.
```

Data Type Definitions

Data type definitions register all the resource data types used in the file. You must register all the data types used in your WML file. Data type definitions have the following syntax:

DataType
any.datatype [{ **InternalLiteral** = *internal.name* |
DocName = "*string*"; [...]}];
[...]

A data type definition begins with the keyword **DataType**. Following the **DataType** keyword is a list of data types that can be further modified with

InternalLiteral
This forces the value of the internal symbol table literal definition of the data type name. This modifier is only used to get around symbol table definitions hard coded into the UIL compiler. It should rarely be used.

DocName This gives an arbitrary string for use in the documentation. This string is meant to supply a different name for the data type for use in the documentation, or a single name for the data type if the data type has aliases.

For example:

```
DataType OddNumber {DocName="OddNumber";};
        NewString;
```

Character Set Definitions

Character set definitions register the Motif Toolkit name and other information for the character set names used in UIL. Character set definitions have the following syntax:

CharacterSet
any.character.set
{ [**FontListElementTag** | **XmStringCharsetName**] = "*string*" ;
[**Alias** = "*string*" ... ; |
Direction = [**LeftToRight** | **RightToLeft**] ; |
ParseDirection = [**LeftToRight** | **RightToLeft**] ; |
CharacterSize = [**OneByte** | **TwoByte**] ;]
[...] } ;
[...]

A character set definition begins with the keyword **CharacterSet**. Following the **CharacterSet** keyword is a list of character sets that can be further modified with

FontListElementTag | XmStringCharsetName
Specifies the name of the character set, which will become the character set component of a compound string segment created using this character set. This modifier is required.

Alias Specifies one or more aliases for the character set name. Each alias can be used within UIL to refer to the same character set.

Direction Specifies the direction of a compound string segment created using this character set. The default is **LeftToRight**.

ParseDirection
Specifies the direction in which an input string is parsed when a compound string segment is created using this character set. The default is whatever **Direction** is specified.

CharacterSize
Specifies the number of bytes in each character of a compound string segment created using this character set. The default is **OneByte**.

For example:

```
CharacterSet
  iso_latin1
    { XmStringCharsetName = "ISO8859-1";
      Alias = "ISOLatin1"; };
  iso_hebrew_lr
    { XmStringCharsetName = "ISO8859-8";
      Alias = "iso_latin8_lr";
      Direction = RightToLeft;
      ParseDirection = LeftToRight; };
  ksc_korean
    { XmStringCharsetName = "KSC5601.1987-0";
      CharacterSize = TwoByte; };
```

Enumeration Set Definitions

Enumeration set definitions register the named constants used in the Motif Toolkit to specify some resource values. Enumeration set definitions have the following syntax:

EnumerationSet
resource.name : *resource.type*
{ *enum.value.name* ; [...] } ;

An enumeration set definition begins with the keyword **EnumerationSet**. For each enumeration set defined, the name and type of the resource are listed. The resource name is the Motif Toolkit resource name, with the beginning **XmN** removed and with the initial letter capitalized. For example, the name of the Motif Toolkit resource **XmNrowColumnType** is **RowColumnType**. The resource type is the data type for the resource; for most resources, this is **integer**. Following the resource name and type is a list of names of enumeration values that can be used as settings for the resource. These names are the same as those in the Motif Toolkit.

For example:

```
EnumerationSet
  RowColumnType: integer
    { XmWORK_AREA; XmMENU_BAR; XmMENU_POPUP;
      XmMENU_PULLDOWN; XmMENU_OPTION; };
```

Control List Definitions

Control list definitions assign a name to groups of controls. You can use these control lists later in class definitions to simplify the structure of your WML file. Control list definitions have the following syntax:

ControlList
any.control.list [{ *any.control*; [...]}];

A control list definition starts with the **ControlList** keyword. Following the **ControlList** keyword are any number of control list definitions. Control list definitions are made up of a control list name followed by the set of controls it represents. For example:

```
ControlList
        Buttons {PushButton;
                 RadioButton;
                 CascadeButton;
                 NewCascadebutton;};
```

Each control specified in the control list must be defined as a class in the file.

Class Definitions

Class definitions describe a particular widget class including its position in the class hierarchy, toolkit convenience function, resources, and controls. There should be one class definition for each widget or gadget in the widget set you want to support in UIL. Class definitions have the following syntax:

Class *class.name* **: MetaClass | Widget | Gadget**
[{[
SuperClass = *class.name*; |
ParentClass = *parent.class.name*; |
InternalLiteral = *internal.name*; |
Alias = *alias*; |
ConvenienceFunction = *convenience.function*; |
WidgetClass = *widget.class*; |
DocName = "*string*"; |
DialogClass = **True** | **False**; |
Resources { *any.resource.name* [{
Default = *new.default.value*; |
Exclude = **True** |
False;
[...]}];
[...]}; |
Controls { *any.control.name*; [...]};
Children { *any.child.name*; [...] };
[...]
]}];

Class definitions start with the **Class** keyword. For each class defined, the name of the class and whether the class is a metaclass, widget, or gadget is listed. Each class definition can be further modified with the keywords described in the following list.

SuperClass This indicates the name of the parent class. Only the root of the hierarchy does not specify a SuperClass.

ParentClass This indicates the name of the widget's automatically created parent class if one exists. This allows resources for that automatically created class to be used in instances of this class. For example, **XmBulletinBoardDialog** creates both an **XmBulletinBoard** and an **XmDialogShell**. To access the resources of the **XmDialogShell** parent class it must be specified here.

InternalLiteral
This forces the value of the internal symbol table literal definition of the class name. This modifier is only used to get around symbol table definitions hard coded into the UIL compiler. It should rarely be used.

Alias This indicates alternate names for the class for use in a UIL specification.

ConvenienceFunction
This indicates the name of the creation convenience function for this class. All widget and gadget classes must have a **ConvenienceFunction.**

WidgetClass This indicates the associated widget class of gadget type classes. Presently, nothing is done with this value.

DocName This defines an arbitrary string for use in the documentation. Presently, nothing is done with this value.

DialogClass This indicates whether the class is a dialog class. Presently, nothing is done with this value.

Resources This lists the resources of the widget class. This keyword can be further modified with

Default This specifies a new default value for this resource. Resource default values are usually set in the resource definition. If an inherited resource's default value is changed by the class, the new default value should be noted here.

Exclude This specifies whether an inherited resource should be excluded from the resource list of the class. **Exclude** is False by default.

Children This lists the names of the automatically created children of this class, so that those children can be accessed in the UIL file.

Controls This lists the controls that the widget class allows. The controls can be other classes or a control list from the control list definition.

The following example uses the examples from the data type definitions and control list definitions above.

```
Class
     TopLevelWidget : MetaClass
          {
          Resources
               {
               XtbNfirstResource;
               XtbNsecondResource;
               };
          };
     NewWidget : Widget
          {
          SuperClass = TopLevelWidget;
          ConvenienceFunction =
              XtbCreateNewWidget;
          Resources
               {
               XtbNnewResource;
               XtbNfirstResource
                  {Default="XtbNEW_VALUE";};
               XtbNsecondResource
                  {Exclude=True;};
               };
          Controls
               {
               NewWidget;
               Buttons;
               };
          };
```

Child Definitions

Child definitions register the classes of automatically created children. Automatically created children are referenced elsewhere in a **uil** file using the **Children** keyword within a class definition. Child definitions have the following syntax:

Child
child.name **:** *class.name*;
[...]

Where *child.name* is the name of the automatically created child and *class.name* is the name of the class of that child.

Resource Definitions

Resource definitions describe a particular resource including its type, and default value. There should be a resource definition for each new resource referenced in the class definitions. Resource definitions have the following syntax:

Resource
resource.name **: Argument | Reason | Constraint | SubResource**
[{[
Type = *type* ; |
ResourceLiteral = *resource.literal* ; |
InternalLiteral = *internal.name*; |
Alias = *alias* ; |
Related = *related* ; |
Default = *default* ; |
DocName = *doc.name* ;]
[...]}]
[...]

Resource definitions start with the **Resource** keyword. For each resource definition, the name of the resource and whether the resource is an argument, reason, constraint or subresource is listed.

Argument Indicates a standard resource

Reason Indicates a callback resource

Constraint Indicates a constraint resource

SubResource Presently, nothing is done with this value

The resource definition can be further modified with the following keywords:

Type This indicates the data type of the resource. It must be listed in the data type definition.

ResourceLiteral This indicates the keyword used in the UIL file to reference the resource. In Motif, the resource name is the same as the **ResourceLiteral**.

InternalLiteral This forces the value of the internal symbol table literal definition of the resource name. This modifier is only used to get around symbol table definitions hard coded into the UIL compiler. It should rarely be used.

Alias This indicates alternate names for the resource for use in a UIL specification.

Related This is a special purpose field that allows resources that act as a counter for the current resources to be related to the resource. UIL automatically sets the value of this related resource to the number of items in the compiled instance of type *resource.name*.

Default This indicates the default value of the resource.

DocName This defines an arbitrary string for use in the documentation. Presently, nothing is done with this value.

The following example uses the examples from the data type definitions, control list definitions and class definitions above.

```
Resource
     XtbNfirstResource : Argument
          { Type = OddNumber;
            Default = "XtbOLD_VALUE";};
     XtbNsecondResource : Argument
          { Type = NewString;
            Default = "XtbNEW_STRING"; };
     XtbNnewResource : Argument
          { Type = OddNumber;
            Default = "XtbODD_NUMBER"; };
```

Appendix A

Constraint Arguments and Automatically Created Children

The following tables list the constraint arguments and automatically created children for widgets available within UIL. The constraints are available for children of the listed widget. For more information about constraint arguments see the *OSF/Motif Programmer's Guide*.

XmForm and XmFormDialog Constraint Arguments	
XmNbottomAttachment	XmNrightAttachment
XmNbottomOffset	XmNrightOffset
XmNbottomPosition	XmNrightPosition
XmNbottomWidget	XmNrightWidget
XmNleftAttachment	XmNtopAttachment
XmNleftOffset	XmNtopOffset
XmNleftPosition	XmNtopPosition
XmNleftWidget	XmNtopWidget
XmNresizable	

XmPanedWindow Constraint Arguments	
XmNallowResize	XmNpaneMinimum
XmNpaneMaximum	XmNskipAdjust

XmFrame Constraint Arguments	
XmNchildHorizontalAlignment	XmNchildType
XmNchildHorizontalSpacing	XmNchildVerticalAlignment

XmSelectionBox Constraint Arguments
XmNchildPlacement

XmScale Automatically Created Children	
Name	**Class**
Xm_Title	XmLabel

XmScrolledWindow Automatically Created Children	
Name	**Class**
Xm_VertScrollBar	XmScrollBar
Xm_HorScrollBar	XmScrollBar

XmOptionMenu Automatically Created Children	
Name	**Class**
Xm_OptionLabel	XmLabelGadget
Xm_OptionButton	XmCascadeButtonGadget

XmPopup and XmPulldownMenu Automatically Created Children	
Name	**Class**
Xm_TearOffControl	XmTearOffButton

XmMainWindow Automatically Created Children	
Name	**Class**
Xm_Separator1	XmSeparator
Xm_Separator2	XmSeparator
Xm_Separator3	XmSeparator

XmMessageBox Automatically Created Children	
Name	**Class**
Xm_Symbol	XmLabel
Xm_Separator	XmSeparator
Xm_Message	XmLabel
Xm_OK	XmPushButton
Xm_Cancel	XmPushButton
Xm_Help	XmPushButton

XmSelectionBox Automatically Created Children	
Name	**Class**
Xm_Items	XmLabel
Xm_ItemsList	XmScrolledList
Xm_Selection	XmLabel
Xm_Text	XmText
Xm_Separator	XmSeparator
Xm_OK	XmPushButton
Xm_Cancel	XmPushButton
Xm_Help	XmPushButton
Xm_Apply	XmPushButton

XmFileSelectionBox Automatically Created Children	
Name	**Class**
Xm_Items	XmLabel
Xm_ItemsList	XmScrolledList
Xm_Separator	XmSeparator
Xm_OK	XmPushButton
Xm_Cancel	XmPushButton
Xm_Help	XmPushButton
Xm_FilterLabel	XmLabel
Xm_FilterText	XmText
Xm_DirList	XmScrolledList
Xm_Dir	XmLabel
Xm_Filter	XmPushButton

Appendix B

UIL Built-In Tables

This appendix contains a listing of part of the UIL built-in tables used during compilation to check that your UIL specification is consistent with the Motif Toolkit.

For each object in the Motif Toolkit, this appendix contains a table that lists the reasons and controls (children) supported by UIL for that object. The arguments supported by UIL for each object are the same as the Motif Toolkit resources for that object. Appendix C lists the name and UIL data type of each UIL argument. For information on which arguments are supported for which objects and for the default values of arguments, see the widget reference pages.

XmArrowButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNactivateCallback XmNarmCallback XmNdestroyCallback XmNdisarmCallback XmNhelpCallback

XmArrowButtonGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback XmNactivateCallback XmNarmCallback XmNdestroyCallback XmNdisarmCallback XmNhelpCallback

XmBulletinBoard	
Controls	**Reasons**
XmArrowButton XmArrowButtonGadget XmBulletinBoard XmBulletinBoardDialog XmCascadeButton XmCascadeButtonGadget XmCommand XmDialogShell XmDrawingArea XmDrawnButton XmErrorDialog XmFileSelectionBox XmFileSelectionDialog XmForm XmFormDialog	MrmNcreateCallback XmNdestroyCallback XmNfocusCallback XmNhelpCallback XmNmapCallback XmNunmapCallback

XmBulletinBoard	
Controls	**Reasons**
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	

XmBulletinBoard	
Controls	**Reasons**
XmWorkArea	
XmWorkingDialog	
user_defined	

XmBulletinBoardDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNfocusCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNpopdownCallback
XmCommand	XmNpopupCallback
XmDialogShell	XmNunmapCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	

XmBulletinBoardDialog	
Controls	**Reasons**
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmCascadeButton	
Controls	**Reasons**
XmPopupMenu XmPulldownMenu	MrmNcreateCallback XmNactivateCallback XmNcascadingCallback XmNdestroyCallback XmNhelpCallback

XmCascadeButtonGadget	
Controls	**Reasons**
XmPulldownMenu	MrmNcreateCallback XmNactivateCallback XmNcascadingCallback XmNdestroyCallback XmNhelpCallback

XmCommand	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNcommandChangedCallback XmNcommandEnteredCallback XmNdestroyCallback XmNfocusCallback XmNhelpCallback XmNmapCallback XmNunmapCallback

XmDialogShell	
Controls	**Reasons**
XmBulletinBoard	MrmNcreateCallback
XmDrawingArea	XmNdestroyCallback
XmFileSelectionBox	XmNpopdownCallback
XmForm	XmNpopupCallback
XmFrame	
XmMessageBox	
XmPanedWindow	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrolledWindow	
XmSelectionBox	
XmWorkArea	

XmDrawingArea	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNexposeCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNinputCallback
XmCascadeButtonGadget	XmNresizeCallback
XmCommand	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	

XmDrawingArea	
Controls	**Reasons**
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmDrawnButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNexposeCallback
	XmNhelpCallback
	XmNresizeCallback

XmErrorDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	

XmErrorDialog	
Controls	**Reasons**
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFileSelectionBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCommand	XmNmapCallback
XmDialogShell	XmNnoMatchCallback
XmDrawingArea	XmNokCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmFileSelectionBox	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFileSelectionDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCommand	XmNmapCallback
XmDialogShell	XmNnoMatchCallback
XmDrawingArea	XmNokCallback
XmDrawnButton	XmNpopdownCallback
XmErrorDialog	XmNpopupCallback
XmFileSelectionBox	XmNunmapCallback
XmFileSelectionDialog	

XmFileSelectionDialog	
Controls	**Reasons**
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	

XmFileSelectionDialog	
Controls	**Reasons**
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmForm	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNfocusCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNunmapCallback
XmCommand	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	

XmForm	
Controls	**Reasons**
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFormDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNfocusCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNpopdownCallback
XmCommand	XmNpopupCallback
XmDialogShell	XmNunmapCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmFormDialog	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmFrame	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNtraverseObscuredCallback
XmCascadeButton	
XmCascadeButtonGadget	
XmCommand	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	

XmFrame	
Controls	**Reasons**
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	

XmFrame	
Controls	**Reasons**
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmInformationDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	

XmInformationDialog	
Controls	**Reasons**
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmLabel	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNdestroyCallback
	XmNhelpCallback

XmLabelGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback XmNdestroyCallback XmNhelpCallback

XmList	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNbrowseSelectionCallback XmNdefaultActionCallback XmNdestroyCallback XmNextendedSelectionCallback XmNhelpCallback XmNmultipleSelectionCallback XmNsingleSelectionCallback

XmMainWindow	
Controls	**Reasons**
XmArrowButton XmArrowButtonGadget XmBulletinBoard XmBulletinBoardDialog XmCascadeButton XmCascadeButtonGadget XmCommand XmDialogShell XmDrawingArea XmDrawnButton XmErrorDialog XmFileSelectionBox XmFileSelectionDialog XmForm XmFormDialog XmFrame	MrmNcreateCallback XmNdestroyCallback XmNhelpCallback XmNtraverseObscuredCallback

XmMainWindow	
Controls	**Reasons**
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	

XmMainWindow	
Controls	**Reasons**
XmWorkingDialog user_defined	

XmMenuBar	
Controls	**Reasons**
XmCascadeButton	MrmNcreateCallback
XmCascadeButtonGadget	XmNdestroyCallback
XmDrawnButton	XmNentryCallback
XmLabel	XmNhelpCallback
XmLabelGadget	XmNmapCallback
XmPopupMenu	XmNtearOffMenuActivateCallback
XmPushButton	XmNtearOffMenuDeactivateCallback
XmPushButtonGadget	XmNunmapCallback
XmSeparator	
XmSeparatorGadget	
XmToggleButton	
XmToggleButtonGadget	
user_defined	

XmMenuShell	
Controls	**Reasons**
XmRowColumn	MrmNcreateCallback XmNdestroyCallback XmNpopdownCallback XmNpopupCallback

XmMessageBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNunmapCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmMessageBox	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmMessageDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmMessageDialog	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmOptionMenu	
Controls	**Reasons**
XmPulldownMenu	MrmNcreateCallback
	XmNdestroyCallback
	XmNentryCallback
	XmNhelpCallback
	XmNmapCallback
	XmNtearOffMenuActivateCallback
	XmNtearOffMenuDeactivateCallback
	XmNunmapCallback

XmPanedWindow	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	
XmCascadeButton	
XmCascadeButtonGadget	
XmCommand	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmPanedWindow	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmPopupMenu	
Controls	**Reasons**
XmCascadeButton	MrmNcreateCallback
XmCascadeButtonGadget	XmNdestroyCallback
XmDrawnButton	XmNentryCallback
XmLabel	XmNhelpCallback
XmLabelGadget	XmNmapCallback
XmPushButton	XmNpopdownCallback
XmPushButtonGadget	XmNpopupCallback
XmSeparator	XmNtearOffMenuActivateCallback
XmSeparatorGadget	XmNtearOffMenuDeactivateCallback
XmToggleButton	XmNunmapCallback
XmToggleButtonGadget	
user_defined	

XmPromptDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCommand	XmNmapCallback
XmDialogShell	XmNnoMatchCallback
XmDrawingArea	XmNokCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	

XmPromptDialog	
Controls	**Reasons**
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmPulldownMenu	
Controls	**Reasons**
XmCascadeButton	MrmNcreateCallback
XmCascadeButtonGadget	XmNdestroyCallback
XmDrawnButton	XmNentryCallback
XmLabel	XmNhelpCallback
XmLabelGadget	XmNmapCallback
XmPushButton	XmNpopdownCallback
XmPushButtonGadget	XmNpopupCallback
XmSeparator	XmNtearOffMenuActivateCallback
XmSeparatorGadget	XmNtearOffMenuDeactivateCallback
XmToggleButton	XmNunmapCallback
XmToggleButtonGadget	
user_defined	

XmPushButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback

XmPushButtonGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback

XmQuestionDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmQuestionDialog	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmRadioBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNentryCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNtearOffMenuActivateCallback
XmCommand	XmNtearOffMenuDeactivateCallback
XmDialogShell	XmNunmapCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmRadioBox	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmRowColumn	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNentryCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNtearOffMenuActivateCallback
XmCommand	XmNtearOffMenuDeactivateCallback
XmDialogShell	XmNunmapCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmRowColumn	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmScale	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNdragCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNvalueChangedCallback
XmCascadeButtonGadget	
XmCommand	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmScale	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmScrollBar	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNdecrementCallback XmNdestroyCallback XmNdragCallback XmNhelpCallback XmNincrementCallback XmNpageDecrementCallback XmNpageIncrementCallback XmNtoBottomCallback XmNtoTopCallback XmNvalueChangedCallback

XmScrolledList	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNbrowseSelectionCallback XmNdefaultActionCallback XmNdestroyCallback XmNextendedSelectionCallback XmNhelpCallback XmNmultipleSelectionCallback XmNsingleSelectionCallback

XmScrolledText	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNactivateCallback XmNdestroyCallback XmNfocusCallback XmNgainPrimaryCallback XmNhelpCallback XmNlosePrimaryCallback XmNlosingFocusCallback XmNmodifyVerifyCallback XmNmodifyVerifyCallbackWcs XmNmotionVerifyCallback XmNvalueChangedCallback

XmScrolledWindow	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNhelpCallback
XmBulletinBoardDialog	XmNtraverseObscuredCallback
XmCascadeButton	
XmCascadeButtonGadget	
XmCommand	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	

XmScrolledWindow	
Controls	**Reasons**
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmSelectionBox	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCommand	XmNmapCallback
XmDialogShell	XmNnoMatchCallback
XmDrawingArea	XmNokCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmSelectionBox	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmSelectionDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNapplyCallback
XmBulletinBoard	XmNcancelCallback
XmBulletinBoardDialog	XmNdestroyCallback
XmCascadeButton	XmNfocusCallback
XmCascadeButtonGadget	XmNhelpCallback
XmCommand	XmNmapCallback
XmDialogShell	XmNnoMatchCallback
XmDrawingArea	XmNokCallback
XmDrawnButton	XmNpopdownCallback
XmErrorDialog	XmNpopupCallback
XmFileSelectionBox	XmNunmapCallback
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmSelectionDialog	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmSeparator	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNdestroyCallback XmNhelpCallback

XmSeparatorGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback XmNdestroyCallback XmNhelpCallback

XmTearOffButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback
	XmNactivateCallback
	XmNarmCallback
	XmNdestroyCallback
	XmNdisarmCallback
	XmNhelpCallback

XmTemplateDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	

XmTemplateDialog	
Controls	**Reasons**
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmText	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNactivateCallback XmNdestroyCallback XmNfocusCallback XmNgainPrimaryCallback XmNhelpCallback XmNlosePrimaryCallback XmNlosingFocusCallback XmNmodifyVerifyCallback XmNmodifyVerifyCallbackWcs XmNmotionVerifyCallback XmNvalueChangedCallback

XmTextField	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNactivateCallback XmNdestroyCallback XmNfocusCallback XmNgainPrimaryCallback XmNhelpCallback XmNlosePrimaryCallback XmNlosingFocusCallback XmNmodifyVerifyCallback XmNmodifyVerifyCallbackWcs XmNmotionVerifyCallback XmNvalueChangedCallback

XmToggleButton	
Controls	**Reasons**
XmPopupMenu	MrmNcreateCallback XmNarmCallback XmNdestroyCallback XmNdisarmCallback XmNhelpCallback XmNvalueChangedCallback

XmToggleButtonGadget	
Controls	**Reasons**
No children are supported	MrmNcreateCallback XmNarmCallback XmNdestroyCallback XmNdisarmCallback XmNhelpCallback XmNvalueChangedCallback

XmWarningDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	

XmWarningDialog	
Controls	**Reasons**
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	

XmWarningDialog	
Controls	**Reasons**
XmWorkArea	
XmWorkingDialog	
user_defined	

XmWorkArea	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNdestroyCallback
XmBulletinBoard	XmNentryCallback
XmBulletinBoardDialog	XmNhelpCallback
XmCascadeButton	XmNmapCallback
XmCascadeButtonGadget	XmNtearOffMenuActivateCallback
XmCommand	XmNtearOffMenuDeactivateCallback
XmDialogShell	XmNunmapCallback
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	

XmWorkArea	
Controls	**Reasons**
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

XmWorkingDialog	
Controls	**Reasons**
XmArrowButton	MrmNcreateCallback
XmArrowButtonGadget	XmNcancelCallback
XmBulletinBoard	XmNdestroyCallback
XmBulletinBoardDialog	XmNfocusCallback
XmCascadeButton	XmNhelpCallback
XmCascadeButtonGadget	XmNmapCallback
XmCommand	XmNokCallback
XmDialogShell	XmNpopdownCallback
XmDrawingArea	XmNpopupCallback
XmDrawnButton	XmNunmapCallback
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

XmWorkingDialog	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

user_defined	
Controls	**Reasons**
XmArrowButton	
XmArrowButtonGadget	
XmBulletinBoard	
XmBulletinBoardDialog	
XmCascadeButton	
XmCascadeButtonGadget	
XmCommand	
XmDialogShell	
XmDrawingArea	
XmDrawnButton	
XmErrorDialog	
XmFileSelectionBox	
XmFileSelectionDialog	
XmForm	
XmFormDialog	
XmFrame	
XmInformationDialog	
XmLabel	
XmLabelGadget	
XmList	
XmMainWindow	
XmMenuBar	
XmMenuShell	
XmMessageBox	
XmMessageDialog	
XmOptionMenu	
XmPanedWindow	
XmPopupMenu	
XmPromptDialog	
XmPulldownMenu	
XmPushButton	
XmPushButtonGadget	
XmQuestionDialog	
XmRadioBox	
XmRowColumn	

user_defined	
Controls	**Reasons**
XmScale	
XmScrollBar	
XmScrolledList	
XmScrolledText	
XmScrolledWindow	
XmSelectionBox	
XmSelectionDialog	
XmSeparator	
XmSeparatorGadget	
XmTemplateDialog	
XmText	
XmTextField	
XmToggleButton	
XmToggleButtonGadget	
XmWarningDialog	
XmWorkArea	
XmWorkingDialog	
user_defined	

Appendix C

UIL Arguments

This appendix provides an alphabetical listing of the UIL arguments and their data types. Each argument name is the same as the corresponding Motif Toolkit resource name. For information on which arguments are supported for which objects and for the default values of arguments, see the widget reference pages.

UIL Argument Name	Argument Type
XmNaccelerator	string
XmNacceleratorText	compound_string
XmNaccelerators	translation_table
XmNadjustLast	boolean
XmNadjustMargin	boolean
XmNalignment	integer
XmNallowOverlap	boolean
XmNallowResize	boolean
XmNallowShellResize	boolean
XmNancestorSensitive	boolean
XmNapplyLabelString	compound_string

UIL Argument Name	Argument Type
XmNarmColor	color
XmNarmPixmap	pixmap
XmNarrowDirection	integer
XmNaudibleWarning	integer
XmNautoShowCursorPosition	boolean
XmNautoUnmanage	boolean
XmNautomaticSelection	boolean
XmNbackground	color
XmNbackgroundPixmap	pixmap
XmNbaseHeight	integer
XmNbaseWidth	integer
XmNblinkRate	integer
XmNborderColor	color
XmNborderPixmap	pixmap
XmNborderWidth	integer
XmNbottomAttachment	integer
XmNbottomOffset	integer
XmNbottomPosition	integer
XmNbottomShadowColor	color
XmNbottomShadowPixmap	pixmap
XmNbottomWidget	widget_ref
XmNbuttonFontList	font_table
XmNcancelButton	widget_ref
XmNcancelLabelString	compound_string
XmNcascadePixmap	pixmap
XmNchildHorizontalAlignment	integer
XmNchildHorizontalSpacing	integer
XmNchildPlacement	integer
XmNchildType	integer
XmNchildVerticalAlignment	integer
XmNcolormap	identifier
XmNcolumns	integer
XmNcommand	compound_string
XmNcommandWindow	widget_ref
XmNcommandWindowLocation	integer
XmNcreatePopupChildProc	any

UIL Argument Name	Argument Type
XmNcursorPosition	integer
XmNcursorPositionVisible	boolean
XmNdecimalPoints	integer
XmNdefaultButton	widget_ref
XmNdefaultButtonShadowThickness	integer
XmNdefaultButtonType	integer
XmNdefaultFontList	font_table
XmNdefaultPosition	boolean
XmNdeleteResponse	integer
XmNdepth	identifier
XmNdialogStyle	integer
XmNdialogTitle	compound_string
XmNdialogType	integer
XmNdirListItemCount	integer
XmNdirListItems	string_table
XmNdirListLabelString	compound_string
XmNdirMask	compound_string
XmNdirSearchProc	any
XmNdirSpec	compound_string
XmNdirectory	compound_string
XmNdoubleClickInterval	integer
XmNeditMode	integer
XmNeditable	boolean
XmNentryAlignment	integer
XmNentryBorder	integer
XmNentryClass	class_rec_name
XmNentryVerticalAlignment	integer
XmNfileListItemCount	integer
XmNfileListItems	string_table
XmNfileListLabelString	compound_string
XmNfileSearchProc	any
XmNfileTypeMask	integer
XmNfillOnArm	boolean
XmNfillOnSelect	boolean
XmNfilterLabelString	compound_string
XmNfontList	font_table

UIL Argument Name	Argument Type
XmNforeground	color
XmNfractionBase	integer
XmNgeometry	string
XmNheight	integer
XmNheightInc	integer
XmNhelpLabelString	compound_string
XmNhighlightColor	color
XmNhighlightOnEnter	boolean
XmNhighlightPixmap	pixmap
XmNhighlightThickness	integer
XmNhistoryItemCount	integer
XmNhistoryItems	string_table
XmNhistoryMaxItems	integer
XmNhistoryVisibleItemCount	integer
XmNhorizontalScrollBar	widget_ref
XmNhorizontalSpacing	integer
XmNiconMask	pixmap
XmNiconPixmap	pixmap
XmNiconWindow	any
XmNiconX	integer
XmNiconY	integer
XmNincrement	integer
XmNindicatorOn	boolean
XmNindicatorSize	integer
XmNindicatorType	integer
XmNinitialDelay	integer
XmNinitialFocus	widget_ref
XmNinitialResourcesPersistent	boolean
XmNinitialState	integer
XmNinput	boolean
XmNinputMethod	string
XmNinsertPosition	identifier
XmNisAligned	boolean
XmNisHomogeneous	boolean
XmNitemCount	integer
XmNitems	string_table

UIL Argument Name	Argument Type
XmNkeyboardFocusPolicy	integer
XmNlabelFontList	font_table
XmNlabelInsensitivePixmap	pixmap
XmNlabelPixmap	pixmap
XmNlabelString	compound_string
XmNlabelType	integer
XmNleftAttachment	integer
XmNleftOffset	integer
XmNleftPosition	integer
XmNleftWidget	widget_ref
XmNlistItemCount	integer
XmNlistItems	string_table
XmNlistLabelString	compound_string
XmNlistMarginHeight	integer
XmNlistMarginWidth	integer
XmNlistSizePolicy	integer
XmNlistSpacing	integer
XmNlistUpdated	boolean
XmNlistVisibleItemCount	integer
XmNmainWindowMarginHeight	integer
XmNmainWindowMarginWidth	integer
XmNmappedWhenManaged	boolean
XmNmappingDelay	integer
XmNmargin	integer
XmNmarginBottom	integer
XmNmarginHeight	integer
XmNmarginLeft	integer
XmNmarginRight	integer
XmNmarginTop	integer
XmNmarginWidth	integer
XmNmaxAspectX	integer
XmNmaxAspectY	integer
XmNmaxHeight	integer
XmNmaxLength	integer
XmNmaxWidth	integer
XmNmaximum	integer

UIL Argument Name	Argument Type
XmNmenuAccelerator	string
XmNmenuBar	widget_ref
XmNmenuHelpWidget	widget_ref
XmNmenuHistory	widget_ref
XmNmenuPost	string
XmNmessageAlignment	integer
XmNmessageString	compound_string
XmNmessageWindow	widget_ref
XmNminAspectX	integer
XmNminAspectY	integer
XmNminHeight	integer
XmNminWidth	integer
XmNminimizeButtons	boolean
XmNminimum	integer
XmNmnemonic	keysym
XmNmnemonicCharSet	string
XmNmultiClick	integer
XmNmustMatch	boolean
XmNmwmDecorations	integer
XmNmwmFunctions	integer
XmNmwmInputMode	integer
XmNmwmMenu	string
XmNnavigationType	integer
XmNnoMatchString	compound_string
XmNnoResize	boolean
XmNnumColumns	integer
XmNokLabelString	compound_string
XmNorientation	integer
XmNoverrideRedirect	boolean
XmNpacking	integer
XmNpageIncrement	integer
XmNpaneMaximum	integer
XmNpaneMinimum	integer
XmNpattern	compound_string
XmNpendingDelete	boolean
XmNpopupEnabled	boolean

UIL Argument Name	Argument Type
XmNpreeditType	string
XmNprocessingDirection	integer
XmNpromptString	compound_string
XmNpushButtonEnabled	boolean
XmNqualifySearchDataProc	any
XmNradioAlwaysOne	boolean
XmNradioBehavior	boolean
XmNrecomputeSize	boolean
XmNrefigureMode	boolean
XmNrepeatDelay	integer
XmNresizable	boolean
XmNresizeHeight	boolean
XmNresizePolicy	integer
XmNresizeWidth	boolean
XmNrightAttachment	integer
XmNrightOffset	integer
XmNrightPosition	integer
XmNrightWidget	widget_ref
XmNrowColumnType	integer
XmNrows	integer
XmNrubberPositioning	boolean
XmNsashHeight	integer
XmNsashIndent	integer
XmNsashShadowThickness	integer
XmNsashWidth	integer
XmNsaveUnder	boolean
XmNscaleHeight	integer
XmNscaleMultiple	integer
XmNscaleWidth	integer
XmNscreen	identifier
XmNscrollBarDisplayPolicy	integer
XmNscrollBarPlacement	integer
XmNscrollHorizontal	boolean
XmNscrollLeftSide	boolean
XmNscrollTopSide	boolean
XmNscrollVertical	boolean

UIL Argument Name	Argument Type
XmNscrolledWindowMarginHeight	integer
XmNscrolledWindowMarginWidth	integer
XmNscrollingPolicy	integer
XmNselectColor	color
XmNselectInsensitivePixmap	pixmap
XmNselectPixmap	pixmap
XmNselectThreshold	integer
XmNselectedItemCount	integer
XmNselectedItems	string_table
XmNselectionArray	any
XmNselectionArrayCount	integer
XmNselectionLabelString	compound_string
XmNselectionPolicy	integer
XmNsensitive	boolean
XmNseparatorOn	boolean
XmNseparatorType	integer
XmNset	boolean
XmNshadowThickness	integer
XmNshadowType	integer
XmNshellUnitType	integer
XmNshowArrows	boolean
XmNshowAsDefault	integer
XmNshowSeparator	boolean
XmNshowValue	boolean
XmNskipAdjust	boolean
XmNsliderSize	integer
XmNsource	any
XmNspacing	integer
XmNstringDirection	integer
XmNsubMenuId	widget_ref
XmNsymbolPixmap	pixmap
XmNtearOffModel	integer
XmNtextAccelerators	translation_table
XmNtextColumns	integer
XmNtextFontList	font_table
XmNtextString	compound_string

UIL Argument Name	Argument Type
XmNtextTranslations	translation_table
XmNtitle	string
XmNtitleEncoding	any
XmNtitleString	compound_string
XmNtopAttachment	integer
XmNtopCharacter	integer
XmNtopItemPosition	integer
XmNtopOffset	integer
XmNtopPosition	integer
XmNtopShadowColor	color
XmNtopShadowPixmap	pixmap
XmNtopWidget	widget_ref
XmNtransient	boolean
XmNtransientFor	widget_ref
XmNtranslations	translation_table
XmNtraversalOn	boolean
XmNtroughColor	color
XmNunitType	integer
XmNunpostBehavior	integer
XmNuseAsyncGeometry	boolean
XmNuserData	any
XmNvalue	any
XmNvalueWcs	wide_character
XmNverifyBell	boolean
XmNverticalScrollBar	widget_ref
XmNverticalSpacing	integer
XmNvisibleItemCount	integer
XmNvisibleWhenOff	boolean
XmNvisual	any
XmNvisualPolicy	integer
XmNwaitForWm	boolean
XmNwhichButton	integer
XmNwidth	integer
XmNwidthInc	integer
XmNwinGravity	integer
XmNwindowGroup	any

UIL Argument Name	Argument Type
XmNwmTimeout	integer
XmNwordWrap	boolean
XmNworkWindow	widget_ref
XmNx	integer
XmNy	integer

Index

Symbols

A

B

C

D

E

F

I

K

L

M

O

P

R

S

T

U

V

W

X

Notes

OSF/Motif™ Release 1.2

Programmer's Reference

TITLES IN THE OSF/Motif SERIES:

OSF/Motif Programmer's Guide

OSF/Motif Programmer's Reference

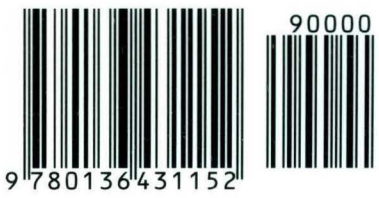
OSF/Motif Style Guide

OSF/Motif User's Guide

Application Environment Specification (AES)
User Environment Volume

Printed in U.S.A.

ISBN 0-13-643115-1



Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142

Prentice-Hall, Inc.