Motif 2.1—Programmer's Reference

Desktop Product Documentation

The Open Group

Copyright © The Open Group, 1997.

All Rights Reserved

The information contained within this document is subject to change without notice.

This documentation and the software to which it relates are derived in part from copyrighted materials supplied by Digital Equipment Corporation, Hewlett-Packard Company, International Business Machines, Massachusetts Institute of Technology, Microsoft Corporation, Sun Microsystems Inc., and The Santa Cruz Operation Inc.

THE OPEN GROUP MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The Open Group shall not be liable for errors contained herein, or for any direct or indirect, incidental, special or consequential damages in connection with the furnishing, performance, or use of this material.

Desktop Product Documentation:

Motif 2.1—Programmer's Reference, Volume 1
ISBN 1-85912-119-5
Document Number M214A

Motif 2.1—Programmer's Reference, Volume 2
ISBN 1-85912-124-1
Document Number M214B

Motif 2.1—Programmer's Reference, Volume 3
ISBN 1-85912-164-0
Document Number M214C

Published in the U.K. by The Open Group, 1997

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:
OGPubs@opengroup.org

OTHER NOTICES

THIS DOCUMENT AND THE SOFTWARE DESCRIBED HEREIN ARE FURNISHED UNDER A LICENSE, AND MAY BE USED AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. TITLE TO AND OWNERSHIP OF THE DOCUMENT AND SOFTWARE REMAIN WITH THE OPEN GROUP OR ITS LICENSORS.

Certain portions of CDE known as "PBMPlus" are copyrighted © 1989, 1991 by Jef Poskanzer. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both the copyright notice and this permission notice appears in supporting documentation. This software is provided "as is" without express or implied warranty.

The code and documentation for the DtComboBox and DtSpinBox widgets were contributed by Interleaf, Inc. Copyright © 1993, Interleaf, Inc.

FOR U.S. GOVERNMENT CUSTOMERS REGARDING THIS DOCUMENTATION AND THE ASSOCIATED SOFTWARE

These notices shall be marked on any reproduction of this data, in whole or in part.

NOTICE:Notwithstanding any other lease or license that may pertain to, or accompany the delivery of, this computer software, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Section 52.227-19 of the FARS Computer Software-Restricted Rights clause.

RESTRICTED RIGHTS NOTICE:Use, duplication, or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013.

RESTRICTED RIGHTS LEGEND:Use, duplication or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the rights in Technical Data and Computer Software clause in DAR 7-104.9(a). This computer software is submitted with "restricted rights." Use, duplication or disclosure is subject to the restrictions as set forth in NASA FAR SUP 18-52.227-79 (April 1985) "Commercial Computer Software-Restricted Rights (April 1985)." If the contract contains the Clause at 18-52.227-74 "Rights in Data General" then the "Alternate III" clause applies.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

Unpublished - All rights reserved under the Copyright Laws of the United States.

This notice shall be marked on any reproduction of this data, in whole or in part.

Contents

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the IT DialTone. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world’s largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- consolidating, prioritizing, and communicating customer requirements to vendors
- conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- licensing and promoting the Open Brand, represented by the ‘‘X’’ mark, that designates vendor products which conform to Open Group Product Standards
- promoting the benefits of open systems to customers, vendors, and the public.

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

The Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of CAE and Preliminary Specifications through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product. There are currently two forms of Product

Standard, namely the Profile Definition and the Component Definition, although these will eventually be merged into one.

The ‘‘X’’ mark is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the X/Open Trade Mark Licence Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on specification development and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

CAE Specifications

CAE (Common Applications Environment) Specifications are the stable specifications that form the basis for our Product Standards, which are used to develop X/Open branded systems. These specifications are intended to be used widely within the industry for product development and procurement purposes.

Anyone developing products that implement a CAE Specification can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. CAE Specifications are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

Preliminary Specifications

Preliminary Specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is not a draft specification; rather, it is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

Preliminary Specifications are analogous to the trial-use standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a CAE Specification. While the intent is to progress Preliminary Specifications to corresponding CAE Specifications, the ability to do so depends on consensus among Open Group members.

Consortium and Technology Specifications

The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as CAE Specifications, in which case the relevant Technology Specification is superseded by a CAE Specification.

In addition, The Open Group publishes:

Product Documentation

This includes product documentation—programmer's guides, user manuals, and so on—relating to the Prestructured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

Guides

These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the CAE Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

Technical Studies

Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

As with all live documents, CAE Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new Version indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it replaces the previous publication.
- A new Issue indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at *http://www.opengroup.org/public/pubs*.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at *http://www.opengroup.org/public/pubs*.

This Book

The *Motif 2.1—Programmer's Reference* contains the reference pages for all Motif programs, Xt widget classes, Xm widget classes, translations, Xm data types and functions, Mrm functions, Uil functions, and file formats.

Audience

This document is written for programmers who want to write applications by 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 2.1 of this document. It applies to Version 2.1 of the Motif software system.

Purpose

The purpose of this guide is to provide detailed information about all Motif 2.1 programs, widget classes, translations, data types, functions, and file formats for the application developer.

Organization

This document is organized into nine chapter and four appendixes:

- Chapter 1 contains the reference pages for Motif programs.
- Chapter 2 contains the reference pages for Xt widget classes.
- Chapter 3 contains the reference pages for Xm widget classes.
- Chapter 4 contains the reference pages for Motif translations.
- Chapter 5 contains the reference pages for Xm data types.
- Chapter 6 contains the reference pages for Xm functions.
- Chapter 7 contains the reference pages for Mrm functions.
- Chapter 8 contains the reference pages for Uil functions.
- Chapter 9 contains the reference pages for Motif file formats.
- Appendix A contains a list of the constraint arguments and automatically created children for widgets available within UIL (User Interface Language).
- 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.
- Appendix D contains a list of the UIL compiler diagnostics messages.

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 section gives practical examples for using the interface.

Return Values
This section lists the values returned by function interfaces.

Errors/Warnings
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 information on Motif and CDE style, refer to the following documents:

CDE 2.1/Motif 2.1—Style Guide and Glossary
Document Number M027 ISBN 1-85912-104-7

CDE 2.1/Motif 2.1—Style Guide Certification Checklist
Document Number M028 ISBN 1-85912-109-8

CDE 2.1/Motif 2.1—Style Guide Reference
Document Number M029 ISBN 1-85912-114-4

For additional information about Motif and CDE, refer to the following Desktop Documentation:

CDE 2.1/Motif 2.1—User's Guide
Document Number M021 ISBN 1-85912-173-X

CDE 2.1—System Manager's Guide
Document Number M022 ISBN 1-85912-178-0

CDE 2.1—Programmer's Overview and Guide
Document Number M023 ISBN 1-85912-183-7

CDE 2.1—Programmer's Reference, Volume 1
Document Number M024A ISBN 1-85912-188-8

CDE 2.1—Programmer's Reference, Volume 2
Document Number M024B ISBN 1-85912-193-4

CDE 2.1—Programmer's Reference, Volume 3
Document Number M024C ISBN 1-85912-174-8

CDE 2.1—Application Developer's Guide
Document Number M026 ISBN 1-85912-198-5

Motif 2.1—Programmer's Guide
Document Number M213 ISBN 1-85912-134-9

Motif 2.1—Widget Writer's Guide
Document Number M216 ISBN 1-85912-129-2

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

Xlib—C Language X Interface

X Toolkit Intrinsics—C Language Interface

Typographic and Keying Conventions

This book uses the following conventions.

DocBook SGML Conventions

This book is written in the Structured Generalized Markup Language (SGML) using the DocBook Document Type Definition (DTD). The following table describes the DocBook markup used for various semantic elements.

Markup Appearance	**Semantic Element(s)**	**Example**
AaBbCc123	The names of commands.	Use the **ls** command to list files.
AaBbCc123	The names of command options.	Use **ls –a** to list all files.
AaBbCc123	Command-line placeholder: replace with a real name or value.	To delete a file, type **rm** *filename*.
AaBbCc123	The names of files and directories.	Edit your **.login** file.
AaBbCc123	Book titles, new words or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Terminology Conventions

Components of the user interface are represented by uppercase letters for each major word in the name of the component, such as PushButton. In addition, this book uses the term *primitive* to mean any subclass of **XmPrimitive** and the term *manager* to mean any subclass of **XmManager**. Note that both of these terms are in lowercase.

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 guide describes keys that use 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 Chapter 13 of the *Motif 2.1—Programmer's Guide* for information on the mechanism for binding virtual keys to actual keys. For details, see the **VirtualBindings**(3) reference page in this manual.

Mouse Conventions

Mouse buttons are described in this reference by 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**(3) reference page in this document.

Problem Reporting

If you have any problems with the software or vendor-supplied documentation, contact your software vendor's customer service department. Comments relating to this Open Group document, however, should be sent to the addresses provided on the copyright page.

Trademarks

Motif® OSF/1®, and UNIX® are registered trademarks and the IT DialTone™, The Open Group™, and the ''X Device''™ are trademarks of The Open Group.

AIX is a trademark of International Business Machines Corp.

HP/UX is a trademark of Hewlett Packard Company.

Solaris is a trademark of Sun Microsystems, Inc.

UnixWare is a trademark of Novell, Inc.

Microsoft Windows is a trademark of Microsoft.

OS/2 is a trademark of International Business Machines Corp.

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

Chapter 1

Programs

mwm

Purpose

The Motif Window Manager

Synopsis

mwm [*options*]

Description

The **mwm** window manager 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.

The stand-alone window manager is not an integral part of CDE and does not support communication with other components in the CDE environment, such as the Style Manager and the Session Manager.

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. Since **mwm** does this by default, this option is of limited use. See the *multiScreen* resource for information on managing 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* scr0 scr1. 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, press button 1 and drag 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		Restores the window to its size before minimizing or maximizing.
Move		Allows the window to be moved with keys or mouse.
Size		Allows the window to be resized.
Minimize		Turns the window into an icon.
Maximize		Makes the window fill the screen.
Lower		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 (see the *matteWidth* resource). 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 + release without moving mouse = click) causes the menu to stay posted. The menu contains the following selections:

Icon Window Menu

Selection	Accelerator	Description
Restore		Opens the associated window.
Move		Allows the icon to be moved with keys.
Size		Inactive (not an option for icons).
Minimize		Inactive (not an option for icons).
Maximize		Opens the associated window and makes it fill the screen.
Lower		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, double-clicking button 1 restores it to its maximized state. Double-clicking button 1 on the icon box’s icon opens the icon box and allows access to the contained icons. (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 a **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.

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.

Pressing mouse button 3 when the pointer is over an icon causes the menu for that icon to pop up.

Icon Menu for the Icon Box

Selection	Accelerator	Description
Restore		Opens the associated window (if not already open).
Move		Allows the icon to be moved with keys.
Size		Inactive.
Minimize		Inactive.
Maximize		Opens the associated window (if not already open) and maximizes its size.
Lower		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

The **mwm** window manager 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, by pressing button 1 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 a 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 arguments 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 arguments *within* and *freeFamily* can also be used with **f.raise** and **f.raise_lower**.

Session Management

The window manager is an X Session Management Protocol aware client. It responds to SaveYourself (and other associated messages) by saving the geometries of its clients to a state file. **mwm** can then be restarted by the XSMP session manager. The default location for the state file is **$HOME/.mwmclientdb**. This location can be overriden with the resource **sessionClientDB**.

X Resources

The **mwm** command 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 file names **/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".

The **mwm** command 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. They apply to all screens and workspaces.

Screen Specific Appearance and Behavior Resources:
These resources specify the appearance and behavior of **mwm** elements that are settable on a per-screen basis.

Client Specific Resources:
These **mwm** resources can be set for a particular client window or class of client windows. 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 "~/", 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 comprise 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 man 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†
backgroundPixmap	BackgroundPixmap	string††	varies†
bottomShadowColor	Foreground	color	varies†
bottomShadowPixmap	Foreground	string††	varies†
fontList	FontList	string†††	"fixed"
foreground	Foreground	color	varies†
saveUnder	SaveUnder	T/F	F

topShadowColor	Background	color	varies†
topShadowPixmap	TopShadowPixmap	string††	varies†

† The default is chosen based on the visual type of the screen.
†† Image name. See **XmInstallImage**(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†
activeBackgroundPixmap	BackgroundPixmap	string††	varies†
activeBottomShadowColor	Foreground	color	varies†
activeBottomShadowPixmap	BottomShadowPixmap	string††	varies†
activeForeground	Foreground	color	varies†
activeTopShadowColor	Background	color	varies†
activeTopShadowPixmap	TopShadowPixmap	string††	varies†

† The default is chosen based on the visual type of the screen.
†† See **XmInstallImage**(3).

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
clientAutoPlace	ClientAutoPlace	T/F	T
colormapFocusPolicy	ColormapFocusPolicy	string	keyboard
configFile	ConfigFile	file	$HOME/mwmrc
deiconifyKeyFocus	DeiconifyKeyFocus	T/F	T
doubleClickTime	DoubleClickTime	millisec.	multi-clicktime†
enableWarp	enableWarp	T/F	T
enforceKeyFocus	EnforceKeyFocus	T/F	T
frameStyle	FrameStyle	string	recessed
iconAutoPlace	IconAutoPlace	T/F	T
iconClick	IconClick	T/F	T
interactivePlacement	InteractivePlacement	T/F	T
keyboardFocusPolicy	KeyboardFocusPolicy	string	explicit
lowerOnIconify	LowerOnIconify	T/F	T
moveThreshold	MoveThreshold	pixels	4
multiScreen	MultiScreen	T/F	T
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
refreshByClearing	RefreshByClearing	T/F	T
rootButtonClick	RootButtonClick	T/F	F

Name	Class	Value Type	Default
screens	Screens	string	varies
sessionClientDB	SessionClientDB	string	**$HOME/.mwmclientd**
showFeedback	ShowFeedback	string	all
startupKeyFocus	StartupKeyFocus	T/F	T
wMenuButtonClick	WMenuButtonClick	T/F	T
wMenuButtonClick2	WMenuButtonClick2	T/F	T

† The resource doubleClickTime is included for backward compatibility. Use of the Xt resource multiClickTime is preferred.

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 BR /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.

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 a **mwm** resource description file. If the pathname begins with "~/", **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 "~/" or "/", **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. The order of the search is shown below:

$HOME/.dt/$LANG/mwmrc
$HOME/.dt/mwmrc
/etc/dt/config/$LANG/sys.mwmrc†
/etc/dt/config/sys.mwmrc†
/usr/dt/config/$LANG/sys.mwmrc†
/usr/dt/config/sys.mwmrc†

Paths marked with '†' are implementation dependent.

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 use of this resource is deprecated. Use the Xt resource *multiClickTime* instead. The value of *doubleClickTime* dynamically defaults to the value of *multiClickTime*.

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.

frameStyle (class *frameStyle*)

If this resource is given a value of "slab", the the window manager frame is drawn such that the client area appears to be at the same height as the top of the window frame. If the resource is set to "recessed", the window frame is drawn such that the client area appears lower than the top of the window frame. The default value is "recessed".

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.

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.

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 hand 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.

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.

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.

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 True.

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.

refreshByClearing (class *RefreshByClearing*)
This resource determines the mechanism used to refresh a window (or the screen) when the **f.refresh_win** (**f.refresh**) function is executed. When set to True, an XClearArea is performed over the window for **f.refresh_win**. When set to False, a covering window is created and

destroyed over the top of the window to be refreshed. If the function is **f.refresh** and this resource is set to True, then an XClearArea is performed over every window on the screen. If the resource is set to False, then one large window covering the entire screen is created and destroyed. The default value of this resource is True.

rootButtonClick (class *RootButtonClick*)

The *rootButtonClick* resource controls whether the a click on the root window will post the root menu in a "sticky" mode. If this resource is set to True, a button click on the root window will post the menu bound to the button down event for that button in a "sticky" fashion. If this resource is set to False, then the same button click would only cause the menu to flash as it would be unposted once the button up event is seen. The criterion used to determine if it is a button click is if the pointer doesn't move between the button down and button up events. The default value for this resource 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.

sessionClientDB (class *SessionClientDB*)

This resource identifies a file name to use as a root when saving state at the request of an XSMP session manager. When the session is saved, the window manager will then reuse the file name by automatically incrementing a suffix.

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 below:

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 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.

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*.

Screen Specific Appearance and Behavior Resources

*Mwm****screen_name***resource_id** is the syntax for specifying screen-specific resources. For example, **Mwm*1*keyBindings** specifies the key bindings to use for screen "1".

Screen Specific Resources

Name	Class	Value Type	Default
buttonBindings	ButtonBindings	string	DefaultButtonBindings
cleanText	CleanText	T/F	T
fadeNormalIcon	FadeNormalIcon	T/F	F
feedbackGeometry	FeedbackGeometry	string	center on screen
frameBorderWidth	FrameBorderWidth	pixels	varies
iconBoxGeometry	IconBoxGeometry	string	6x1+0-0
iconBoxName	IconBoxName	string	iconbox
iconBoxSBDisplayPolicy	IconBoxSBDisplayPolicy	string	all
iconBoxTitle	IconBoxTitle	XmString	Icons
iconDecoration	IconDecoration	string	varies
iconImageMaximum	IconImageMaximum	wxh	48x48
iconImageMinimum	IconImageMinimum	wxh	16x16
iconPlacement	IconPlacement	string	left bottom
iconPlacementMargin	IconPlacementMargin	pixels	varies
keyBindings	KeyBindings	string	DefaultKeyBindings
limitResize	LimitResize	T/F	T
maximumMaximumSize	MaximumMaximumSize	wxh (pixels)	2X screen w&h
moveOpaque	MoveOpaque	T/F	F
resizeBorderWidth	ResizeBorderWidth	pixels	varies
resizeCursors	ResizeCursors	T/F	T

transientDecoration	TransientDecoration	string	menu title
transientFunctions	TransientFunctions	string	–minimize–maximize
useIconBox	UseIconBox	T/F	F

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 backgroundPixmap 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.

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.

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.

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 stand alone 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, don't 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).

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 **mwm** resource description file. The default value for this resource is "DefaultKeyBindings".

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.

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.

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 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.

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**(3) 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**(3) 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).

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 don't 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	BottomShadowPixmap	color	icon bottom shadow pixmap
iconImageForeground	Foreground	color	varies
iconImageTopShadowColor	Background	color	icon top shadow color
iconImageTopShadowPixmap	TopShadowPixmap	color	icon top shadow pixmap
matteBackground	Background	color	background
matteBottomShadowColor	Foreground	color	bottom shadow color
matteBottomShadowPixmap	BottomShadowPixmap	color	bottom shadow pixmap
matteForeground	Foreground	color	foreground
matteTopShadowColor	Background	color	top shadow color
matteTopShadowPixmap	TopShadowPixmap	color	top shadow pixmap

matteWidth	MatteWidth	pixels	0
maximumClientSize	MaximumClientSize	wxh vertical horizontal	fill the screen
useClientIcon	UseClientIcon	T/F	T
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**(3) 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**(3) 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†.

†See **mwmrc**(4).

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 pixmap or 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 of this resource 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 True, giving the client-supplied icon image higher precedence than the user-supplied icon image.

usePPosition (class *UsePPosition*)

This resource specifies whether Mwm honors program specified position **PPosition** specified in the *WM_NORMAL_HINTS* property in the absence of an 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 cause **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 creating custom menus in your **mwmrc** file (see **mwmrc**(4) and specifying resources to activate the custom menus. The resources have the form *Mwm** **client_name_or_class****windowMenu*. 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 resource manager property (see **xrdb**(1) and 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.

The **mwm** resource description file is described in **mwmrc**(4).

Environment

The **mwm** window manager uses the environment variable **HOME** specifying the user's home directory.

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

The **mwm** window uses the environment variable **XFILESEARCHPATH**, **XUSERFILESEARCHPATH**, **XAPPLRESDIR**, **XENVIRONMENT**, **LANG**, and **HOME** in determining search paths for resource defaults files. The **mwm** window manager may also us **XBMLANGPATH** to search for bitmap files.

The **mwm** window manager 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

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

Files

/usr/lib/X11/$LANG/system.mwmrc

/usr/lib/X11/system.mwmrc

/usr/lib/X11/app-defaults/Mwm

$HOME/Mwm

$HOME/$LANG/.mwmrc

$HOME/.mwmrc

Related Information

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

uil

Purpose

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. UIDs are portable only across same-size machine architectures.
- **–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.

Related Information

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

xmbind

Purpose

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**(3).

Options

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

Related Information

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

Chapter 2
Xt Widget Classes

ApplicationShell

Purpose

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**	**Type**	**Default**	**Access**
XmNargc	XmCArgc	int	0	CSG
XmNargv	XmCArgv	String *	NULL	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. When **XtGetValues** is called on this resource, the returned value is a pointer to the actual resource value and should not be freed.

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**	**Type**	**Default**	**Access**
XmNiconic	XmCIconic	Boolean	False	CSG
XmNiconName	XmCIconName	String	NULL	CSG
XmNiconNameEncoding	XmCIconNameEncoding	Atom	dynamic	CSG

VendorShell Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaudibleWarning	XmCAudibleWarning	unsigned char	XmBELL	CSG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNdeleteResponse	XmCDeleteResponse	unsigned char	XmDESTROY	CSG
XmNinputMethod	XmCInputMethod	String	NULL	CSG

XmNinputPolicy	XmCInputPolicy	XmInputPolicy	XmPER_SHELL	CSG
XmNkeyboardFocusPolicy	XmCKeyboardFocusPolicy	unsigned char	XmEXPLICIT	CSG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	XmLEFT_TO_RIGHT	CG
XmNmwmDecorations	XmCMwmDecorations	int	-1	CG
XmNmwmFunctions	XmCMwmFunctions	int	-1	CG
XmNmwmInputMode	XmCMwmInputMode	int	-1	CG
XmNmwmMenu	XmCMwmMenu	String	NULL	CG
XmNpreeditType	XmCPreeditType	String	dynamic	CSG
XmNshellUnitType	XmCShellUnitType	unsigned char	XmPIXELS	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNunitType	XmCUnitType	unsigned char	XmPIXELS	CSG
XmNuseAsyncGeometry	XmCUseAsyncGeometry	Boolean	False	CSG

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

XmNmaxWidth	XmCMaxWidth	int	XtUnspecifiedShellInt	CSG
XmNminAspectX	XmCMinAspectX	int	XtUnspecifiedShellInt	CSG
XmNminAspectY	XmCMinAspectY	int	XtUnspecifiedShellInt	CSG
XmNminHeight	XmCMinHeight	int	XtUnspecifiedShellInt	CSG
XmNminWidth	XmCMinWidth	int	XtUnspecifiedShellInt	CSG
XmNtitle	XmCTitle	String	dynamic	CSG
XmNtitleEncoding	XmCTitleEncoding	Atom	dynamic	CSG
XmNtransient	XmCTransient	Boolean	False	CSG
XmNwaitForWm	XmCWaitForWm	Boolean	True	CSG
XmNwidthInc	XmCWidthInc	int	XtUnspecifiedShellInt	CSG
XmNwindowGroup	XmCWindowGroup	Window	dynamic	CSG
XmNwinGravity	XmCWinGravity	int	dynamic	CSG
XmNwmTimeout	XmCWmTimeout	int	5000 ms	CSG

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

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmapped-WhenManaged	XmCMapped-WhenManaged	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for ApplicationShell.

Related Information

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

Composite

Purpose

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**	**Type**	**Default**	**Access**
XmNchildren	XmCReadOnly	WidgetList	NULL	G
XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	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 *w*);

w Specifies the widget.

Composite widgets that allow clients to order their children (usually homogeneous boxes) can call their widget instance's **XmNinsertPosition** 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 **XmNinsertPosition** procedure when it creates each composite widget instance.

The return value of the **XmNinsertPosition** 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 the value of *XmNumChildren* indicates that it should go after all other children. By default, unless a subclass or an application provides an **XmNinsertPosition** procedure, each child is inserted at the end of the **XmNchildren** list. The **XmNinsertPosition** procedure 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**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	1	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmapped-WhenManaged	XmCMapped-WhenManaged	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for Composite.

Related Information

Core(3).

Constraint

Purpose

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**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	1	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for Constraint.

Related Information

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

Core

Purpose

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	Type	Default	Access
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG

XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	1	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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. Note that the default accelerators for any widget will always be installed, no matter whether this resource is specified or not.

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 0 (zero) means that no border shows. Note that you should use resources like **XmNshadowThickness** and **XmNhighlightThickness** instead of **XmNborderWidth** to specify border widths.

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. Note that the default translations for any widget will always be installed, no matter whether this resource is specified or not.

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(3) and **RectObj**(3).

Object

Purpose

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**	**Type**	**Default**	**Access**
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C

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

Translations

There are no translation for Object.

OverrideShell

Purpose

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**	**Type**	**Default**	**Access**
XmNallowShell-Resize	XmCAllowShell-Resize	Boolean	False	CG
XmNcreatePopup-ChildProc	XmCCreatePopup-ChildProc	XtCreatePopup-ChildProc	NULL	CSG
XmNgeometry	XmCGeometry	String	NULL	CSG
XmNoverride-Redirect	XmCOverride-Redirect	Boolean	True	CSG
XmNpopdown-Callback	XmCCallback	XtCallbackList	NULL	C
XmNpopup-Callback	XmCCallback	XtCallbackList	NULL	C
XmNsaveUnder	XmCSaveUnder	Boolean	True	CSG
XmNvisual	XmCVisual	Visual *	CopyFromParent	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG

XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmapped-WhenManaged	XmCMapped-WhenManaged	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for OverrideShell.

Related Information

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

RectObj

Purpose

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**	**Type**	**Default**	**Access**
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNborderWidth	XmCBorderWidth	Dimension	1	CSG

XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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**	**Type**	**Default**	**Access**
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C

Translations

There are no translations for RectObj.

Related Information

Object(3).

Shell

Purpose

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**	**Type**	**Default**	**Access**
XmNallowShell-Resize	XmCAllowShell-Resize	Boolean	False	CG
XmNcreatePopup-ChildProc	XmCCreatePopup-ChildProc	XtCreatePopup-ChildProc	NULL	CSG
XmNgeometry	XmCGeometry	String	NULL	CSG
XmNoverride-Redirect	XmCOverride-Redirect	Boolean	False	CSG
XmNpopdown-Callback	XmCCallback	XtCallbackList	NULL	C
XmNpopup-Callback	XmCCallback	XtCallbackList	NULL	C
XmNsaveUnder	XmCSaveUnder	Boolean	False	CSG
XmNvisual	XmCVisual	Visual *	CopyFrom-Parent	CSG

XmNallowShellResize

Specifies that if this resource is False, the Shell widget instance returns **XtGeometryNo** to all geometry requests from its children. All Motif convenience create dialog functions override this default value and set **XmNallowShellResize** to True.

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. When **XtGetValues** is called on this resource, the returned value is a pointer to the actual resource value and should not be freed. In addition, this resource is not copied on creation or by **XtSetValues**. The application must ensure that the string remains valid until the shell is realized.

XmNoverrideRedirect
If True, specifies that the widget instance is a temporary window that 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**. The second argument to **XtPopup** must be **XtGrabNone**.

XmNsaveUnder
If True, specifies that it 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**	**Type**	**Default**	**Access**
XmNchildren	XmCReadOnly	WidgetList	NULL	G
XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG

XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for Shell.

Related Information

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

TopLevelShell

Purpose

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	Type	Default	Access
XmNiconic	XmCIconic	Boolean	False	CSG

XmNiconName	XmCIconName	String	NULL	CSG
XmNiconNameEncoding	XmCIconNameEncoding	Atom	dynamic	CSG

XmNiconic If True when the widget is *created*, specifies that the widget should start as an icon when it is realized. A value of False indicates that the widget is not to be realized as an icon. This resource will only override the **XmNinitialState** resource when specified in the call that creates the widget.

XmNiconName
Specifies the short form of the application name to be displayed by the window manager when the application is iconified. When **XtGetValues** is called on this resource, the returned value is a pointer to the actual resource value and should not be freed.

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**	**Type**	**Default**	**Access**
XmNaudibleWarning	XmCAudibleWarning	unsigned char	XmBELL	CSG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNdeleteResponse	XmCDeleteResponse	unsigned char	XmDESTROY	CSG
XmNinputMethod	XmCInputMethod	String	NULL	CSG

XmNinputPolicy	XmCInputPolicy	XmInputPolicy	XmPER_SHELL	CSG
XmNkeyboardFocusPolicy	XmCKeyboardFocusPolicy	unsigned char	XmEXPLICIT	CSG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTabel	XmRenderTable	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	XmLEFT_TO_-RIGHT	CG
XmNmwmDecorations	XmCMwmDecorations	int	-1	CG
XmNmwmFunctions	XmCMwmFunctions	int	-1	CG
XmNmwmInputMode	XmCMwmInputMode	int	-1	CG
XmNmwmMenu	XmCMwmMenu	String	NULL	CG
XmNpreeditType	XmCPreeditType	String	dynamic	CSG
XmNshellUnitType	XmCShellUnitType	unsigned char	XmPIXELS	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNunitType	XmCUnitType	unsigned char	XmPIXELS	CSG
XmNuseAsyncGeometry	XmCUseAsyncGeometry	Boolean	False	CSG

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

XmNmaxWidth	XmCMaxWidth	int	XtUnspecifiedShellInt	CSG
XmNminAspectX	XmCMinAspectX	int	XtUnspecifiedShellInt	CSG
XmNminAspectY	XmCMinAspectY	int	XtUnspecifiedShellInt	CSG
XmNminHeight	XmCMinHeight	int	XtUnspecifiedShellInt	CSG
XmNminWidth	XmCMinWidth	int	XtUnspecifiedShellInt	CSG
XmNtitle	XmCTitle	String	dynamic	CSG
XmNtitleEncoding	XmCTitleEncoding	Atom	dynamic	CSG
XmNtransient	XmCTransient	Boolean	False	CSG
XmNwaitForWm	XmCWaitForWm	Boolean	True	CSG
XmNwidthInc	XmCWidthInc	int	XtUnspecifiedShellInt	CSG
XmNwindowGroup	XmCWindowGroup	Window	dynamic	CSG
XmNwinGravity	XmCWinGravity	int	dynamic	CSG
XmNwmTimeout	XmCWmTimeout	int	5000 ms	CSG

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

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for TopLevelShell.

Related Information

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

TransientShell

Purpose

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**	**Type**	**Default**	**Access**
XmNtransientFor	XmCTransientFor	Widget	NULL	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**	**Type**	**Default**	**Access**
XmNaudibleWarning	XmCAudibleWarning	unsigned char	XmBELL	CSG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNdeleteResponse	XmCDeleteResponse	unsigned char	XmDESTROY	CSG
XmNinputMethod	XmCInputMethod	String	NULL	CSG
XmNinputPolicy	XmCInputPolicy	XmInputPolicy	XmPER_SHELL	CSG
XmNkeyboardFocusPolicy	XmCKeyboardFocusPolicy	unsigned char	XmEXPLICIT	CSG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTabel	XmRenderTable	dynamic	CSG

XmNlayoutDirection	XmCLayoutDirection	XmDirection	XmLEFT_TO_RIGHT	CG
XmNmwmDecorations	XmCMwmDecorations	int	-1	CG
XmNmwmFunctions	XmCMwmFunctions	int	-1	CG
XmNmwmInputMode	XmCMwmInputMode	int	-1	CG
XmNmwmMenu	XmCMwmMenu	String	NULL	CG
XmNpreeditType	XmCPreeditType	String	dynamic	CSG
XmNshellUnitType	XmCShellUnitType	unsigned char	XmPIXELS	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNunitType	XmCUnitType	unsigned char	XmPIXELS	CSG
XmNuseAsyncGeometry	XmCUseAsyncGeometry	Boolean	False	CSG

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

XmNminWidth	XmCMinWidth	int	XtUnspecifiedShellInt	CSG
XmNtitle	XmCTitle	String	dynamic	CSG
XmNtitleEncoding	XmCTitleEncoding	Atom	dynamic	CSG
XmNtransient	XmCTransient	Boolean	True	CSG
XmNwaitForWm	XmCWaitForWm	Boolean	True	CSG
XmNwidthInc	XmCWidthInc	int	XtUnspecifiedShellInt	CSG
XmNwindowGroup	XmCWindowGroup	Window	dynamic	CSG
XmNwinGravity	XmCWinGravity	int	dynamic	CSG
XmNwmTimeout	XmCWmTimeout	int	5000 ms	CSG

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

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG

XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for TransientShell.

Related Information

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

VendorShell

Purpose

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* (ICCCM) allows a window manager to change or control the border width of a reparented top-level window.

VendorShell holds the *XmQTspecifyRenderTable* trait.

Classes

VendorShell inherits behavior, resources, and traits 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	Type	Default	Access
XmNaudibleWarning	XmCAudibleWarning	unsigned char	XmBELL	CSG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNdeleteResponse	XmCDeleteResponse	unsigned char	XmDESTROY	CSG
XmNinputMethod	XmCInputMethod	string	NULL	CSG
XmNinputPolicy	XmCInputPolicy	XmInputPolicy	XmPER_SHELL	CSG
XmNkeyboardFocusPolicy	XmCKeyboardFocusPolicy	unsigned char	XmEXPLICIT	CSG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTabel	XmRenderTable	dynamic	CSG

XmNlayoutDirection	XmCLayoutDirection	XmDirection	XmLEFT_TO_-RIGHT	CG
XmNmwmDecorations	XmCMwmDecorations	int	-1	CG
XmNmwmFunctions	XmCMwmFunctions	int	-1	CG
XmNmwmInputMode	XmCMwmInputMode	int	-1	CG
XmNmwmMenu	XmCMwmMenu	String	NULL	CG
XmNpreeditType	XmCPreeditType	String	dynamic	CSG
XmNverifyPreedit	XmCVerifyPreedit	Boolean	False	CSG
XmNshellUnitType	XmCShellUnitType	unsigned char	XmPIXELS	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNunitType	XmCUnitType	unsigned char	XmPIXELS	CSG
XmNuseAsyncGeometry	XmCUseAsyncGeometry	Boolean	False	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 button descendants. See the **XmNbuttonRenderTable** resource.

XmNbuttonRenderTable
Specifies the render table used for VendorShell's button descendants. If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNbuttonRenderTable** 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 holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, **XmNbuttonRenderTable** is initialized to the **XmBUTTON_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

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. When **XtGetValues** is called on this resource, the returned value is a pointer to the actual resource value and should not be freed.

XmNinputPolicy
Specifies the policy to follow for creating an Input Context (IC) for this shell. This resource can have the following values:

XmPER_SHELL
Specifies that only one XIC is created per shell.

XmPER_WIDGET
Specifies that one XIC is created for each widget.

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 label descendants. See the resource, **XmNlabelRenderTable**.

XmNlabelRenderTable
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 holds the *XmQTspecifyRenderTable* trait.

If such an ancestor is found, **XmNlabelRenderTable** is initialized to the **XmLABEL_RENDER_TABLE** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

XmNlayoutDirection

Specifies the direction in which the subwidgets, children of a widget, or other visual components are to be laid out. This policy will apply as the default layout policy for all descendants of this VendorShell.

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 property, _MOTIF_WM_HINTS. 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.

When **XtGetValues** is called on this resource, the returned value is a pointer to the actual resource value and should not be freed.

XmNpreeditType
Specifies the input method style or styles available to the input manager. The resource can be a comma-separated list of the following values:

Preedit Values

Preedit Value	**XIM Style**
OffTheSpot	XIMPreeditArea

Root	XIMPreeditNothing
None	XIMPreeditNone
OverTheSpot	XIMPreeditPosition
OnTheSpot	XIMPreeditCallbacks

When **XtGetValues** is called on this resource, the returned value is a pointer to the actual resource value and should not be freed.

XmNshellUnitType
This resource is obsolete, and is included only for compatibility with earlier releases of Motif. Use the **XmNunitType** resource instead.

XmNtextFontList
Specifies the font list used for text descendants. See the **XmNtextRenderTable** resource.

XmNtextRenderTable
Specifies the render table used for VendorShell’s Text and List descendants. If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNtextRenderTable** 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 holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, **XmNtextRenderTable** is initialized to the **XmTEXT_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

XmNunitType
Provides the basic support for resolution independence. It defines the type of units a widget uses with sizing and positioning resources. The resource has a default unit type of **XmPIXELS**.

The unit type can also be specified in resource files, with the following format:

```
<floating value><unit>
```

where:

unit	is <" ", *pixels*, *inches*, *centimeters*, *millimeters*, *points*, *font units*>
pixels	is <*pix*, *pixel*, *pixels*>
inches	is <*in*, *inch*, *inches*>
centimeter	is <*cm*, *centimeter*, *centimeters*>
millimeters	is <*mm*, *millimeter*, *millimeters*>
points	is <*pt*, *point*, *points*>
font units	is <*fu*, *font_unit*, *font_units*>
float	is {"+"|"-"}{{<"0"-"9">*}.}<"0"-"9">*

Note that the type Dimension must always be positive.

For example,

```
xmfonts*XmMainWindow.height: 10.4cm
*PostIn.width: 3inches
```

XmNunitType can have the following values:

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

XmMILLIMETERS
All values provided to the widget are treated as normal millimeter values.

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

XmCENTIMETERS
All values provided to the widget are treated as normal centimeter values.

XmINCHES
All values provided to the widget are treated as normal inch values.

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

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

XmFONT_UNITS
All values provided to the widget are treated as normal font units. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

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**.

For more information about units, refer to the **XmConvertUnits** reference page.

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**	**Type**	**Default**	**Access**
XmNbaseHeight	XmCBaseHeight	int	XtUnspecifiedShellInt	CSG
XmNbaseWidth	XmCBaseWidth	int	XtUnspecifiedShellInt	CSG
XmNheightInc	XmCHeightInc	int	XtUnspecifiedShellInt	CSG
XmNiconMask	XmCIconMask	Pixmap	NULL	CSG
XmNiconPixmap	XmCIconPixmap	Pixmap	NULL	CSG
XmNiconWindow	XmCIconWindow	Window	NULL	CSG
XmNiconX	XmCIconX	int	XtUnspecifiedShellInt	CSG
XmNiconY	XmCIconY	int	XtUnspecifiedShellInt	CSG
XmNinitialState	XmCInitialState	int	NormalState	CSG
XmNinput	XmCInput	Boolean	True	CSG
XmNmaxAspectX	XmCMaxAspectX	int	XtUnspecifiedShellInt	CSG
XmNmaxAspectY	XmCMaxAspectY	int	XtUnspecifiedShellInt	CSG
XmNmaxHeight	XmCMaxHeight	int	XtUnspecifiedShellInt	CSG
XmNmaxWidth	XmCMaxWidth	int	XtUnspecifiedShellInt	CSG
XmNminAspectX	XmCMinAspectX	int	XtUnspecifiedShellInt	CSG
XmNminAspectY	XmCMinAspectY	int	XtUnspecifiedShellInt	CSG
XmNminHeight	XmCMinHeight	int	XtUnspecifiedShellInt	CSG
XmNminWidth	XmCMinWidth	int	XtUnspecifiedShellInt	CSG
XmNtitle	XmCTitle	String	dynamic	CSG
XmNtitleEncoding	XmCTitleEncoding	Atom	dynamic	CSG
XmNtransient	XmCTransient	Boolean	False	CSG
XmNwaitForWm	XmCWaitForWm	Boolean	True	CSG
XmNwidthInc	XmCWidthInc	int	XtUnspecifiedShellInt	CSG
XmNwindowGroup	XmCWindowGroup	Window	dynamic	CSG
mNwinGravity	XmCWinGravity	int	dynamic	CSG
XmNwmTimeout	XmCWmTimeout	int	5000 ms	CSG

Shell Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNallowShell-Resize	XmCAllowShell-Resize	Boolean	False	CG
XmNcreatePopup-ChildProc	XmCCreatePopup-ChildProc	XtCreatePopup-ChildProc	NULL	CSG
XmNgeometry	XmCGeometry	String	NULL	CSG
XmNoverride-Redirect	XmCOverride-Redirect	Boolean	False	CSG
XmNpopdown-Callback	XmCCallback	XtCallbackList	NULL	C
XmNpopup-Callback	XmCCallback	XtCallbackList	NULL	C
XmNsaveUnder	XmCSaveUnder	Boolean	False	CSG
XmNvisual	XmCVisual	Visual *	CopyFrom-Parent	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestor-Sensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG

XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for VendorShell.

Related Information

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

WMShell

Purpose

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**	**Type**	**Default**	**Access**
XmNbaseHeight	XmCBaseHeight	int	XtUnspecifiedShellInt	CSG

XmNbaseWidth	XmCBaseWidth	int	XtUnspecifiedShellInt	CSG
XmNheightInc	XmCHeightInc	int	XtUnspecifiedShellInt	CSG
XmNiconMask	XmCIconMask	Pixmap	NULL	CSG
XmNiconPixmap	XmCIconPixmap	Pixmap	NULL	CSG
XmNiconWindow	XmCIconWindow	Window	NULL	CSG
XmNiconX	XmCIconX	int	XtUnspecifiedShellInt	CSG
XmNiconY	XmCIconY	int	XtUnspecifiedShellInt	CSG
XmNinitialState	XmCInitialState	int	NormalState	CSG
XmNinput	XmCInput	Boolean	False	CSG
XmNmaxAspectX	XmCMaxAspectX	int	XtUnspecifiedShellInt	CSG
XmNmaxAspectY	XmCMaxAspectY	int	XtUnspecifiedShellInt	CSG
XmNmaxHeight	XmCMaxHeight	int	XtUnspecifiedShellInt	CSG
XmNmaxWidth	XmCMaxWidth	int	XtUnspecifiedShellInt	CSG
XmNminAspectX	XmCMinAspectX	int	XtUnspecifiedShellInt	CSG
XmNminAspectY	XmCMinAspectY	int	XtUnspecifiedShellInt	CSG
XmNminHeight	XmCMinHeight	int	XtUnspecifiedShellInt	CSG
XmNminWidth	XmCMinWidth	int	XtUnspecifiedShellInt	CSG
XmNtitle	XmCTitle	String	dynamic	CSG
XmNtitleEncoding	XmCTitleEncoding	Atom	dynamic	CSG
XmNtransient	XmCTransient	Boolean	False	CSG
XmNwaitForWm	XmCWaitForWm	Boolean	True	CSG
XmNwidthInc	XmCWidthInc	int	XtUnspecifiedShellInt	CSG
XmNwindowGroup	XmCWindowGroup	Window	dynamic	CSG
XmNwinGravity	XmCWinGravity	int	dynamic	CSG
XmNwmTimeout	XmCWmTimeout	int	5000 ms	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.

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.

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* (ICCCM).

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.

XmNmaxWidth
Specifies the maximum width that the application wants the widget instance to have.

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.

XmNminWidth
Specifies the minimum width that the application wants the widget instance to have.

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. When **XtGetValues** is called on this resource, the returned value is a pointer to the actual resource value and should not be freed.

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 it is *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**	**Type**	**Default**	**Access**
XmNallowShell-Resize	XmCAllowShell-Resize	Boolean	False	CG
XmNcreatePopup-ChildProc	XmCCreatePopup-ChildProc	XtCreatePopup-ChildProc	NULL	CSG
XmNgeometry	XmCGeometry	String	NULL	CSG

XmNoverride Redirect	XmCOverride-Redirect	Boolean	False	CSG
XmNpopdown-Callback	XmCCallback	XtCallbackList	NULL	C
XmNpopup-Callback	XmCCallback	XtCallbackList	NULL	C
XmNsaveUnder	XmCSaveUnder	Boolean	False	CSG
XmNvisual	XmCVisual	Visual *	CopyFrom-Parent	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestor-Sensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallback-List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG

XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for WMShell.

Related Information

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

Chapter 3

Xm Widget Classes

XmArrowButton

Purpose

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.

ArrowButton holds the *XmQTactivatable* trait.

Classes

ArrowButton inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNarrowDirection	XmCArrowDirection	unsigned char	XmARROW_UP	CSG
XmNdetailShadowThickness	XmCDetailShadowThickness	Dimension	2	CSG
XmNdisarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	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**. This callback uses the *XmQTactivatable* trait.

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**

XmNdetailShadowThickness

Specifies the thickness of the inside arrow shadows. The default thickness is 2 pixels.

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**	**Type**	**Default**	**Access**
XmNbottomShadowColor	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadowPixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOn-Enter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmNONE	CSG
XmNpopupHandlerCallback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadow-Thickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG

XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallback-List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResourcesPersistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhenManaged	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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 key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<EnterWindow>:
　　Enter()

<LeaveWindow>:
　　Leave()

c<Btn1Down>:
　　ButtonTakeFocus()

≈c<Btn1Down>:
　　Arm()

≈c<Btn1Down> ,≈c< Btn1Up>:
　　Activate() Disarm()

≈c<Btn1Down>(2+):
　　MultiArm()

≈c<Btn1Up>(2+):
MultiActivate()

≈c<Btn1Up>:
Activate() Disarm()

:<Key>osfActivate:
PrimitiveParentActivate()

:<Key>osfCancel:
PrimitiveParentCancel()

:<Key>osfSelect:
ArmAndActivate()

:<Key>osfHelp:
Help()

≈s ≈m ≈a <Key>Return:
PrimitiveParentActivate()

≈s ≈m ≈a <Key>space:
ArmAndActivate()

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.

ButtonTakeFocus():
Causes the ArrowButton to take keyboard focus when **Ctrl<Btn1Down>** is pressed, without activating the widget.

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 Btn1 is pressed.

LeaveWindow:
Draws the ArrowButton shadow in its unselected state if the pointer leaves the window while Btn1 is pressed.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Related Information

Core(3), **XmCreateArrowButton**(3), and **XmPrimitive**(3).

XmArrowButtonGadget

Purpose

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.

ArrowButtonGadget holds the *XmQTactivatable* trait.

Classes

ArrowButtonGadget inherits behavior, resources, and traits 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).

XmArrowButtonGadget Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNarrowDirection	XmCArrowDirection	unsigned char	XmARROW_UP	CSG
XmNdetailShadowThickness	XmCDetailShadowThickness	Dimension	2	CSG
XmNdisarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	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**. This callback uses the *XmQTactivatable* trait.

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**

XmNdetailShadowThickness
Specifies the thickness of the inside arrow shadows. The default thickness is 2 pixels.

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

XmArrowButtonGadget 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**	**Type**	**Default**	**Access**
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOn-Enter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlight- Pixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	2	CSG
XmNlayoutDirection	XmNCLayout-Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmNONE	CSG
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG

XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

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

Btn1Down: Draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**.

Btn1Down **or** Btn1Up:
Draws the shadow in the unselected state. If the pointer is within the ArrowButtonGadget, calls the callbacks for **XmNactivateCallback**. Calls the callbacks for **XmNdisarmCallback**.

Btn1Down**(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**.

Btn1Up**(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**.

KeyosfSelect:
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.

KeyosfHelp:
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 <Btn1> is pressed.

Leave:	Draws the ArrowButtonGadget shadow in its unselected state if the pointer leaves the gadget while <Btn1> is pressed.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Related Information

Object(3), **RectObj**(3), **XmCreateArrowButtonGadget**(3), and **XmGadget**(3).

XmBulletinBoard

Purpose

The BulletinBoard widget class

Synopsis

#include <Xm/BulletinB.h>

Description

BulletinBoard is a composite widget that provides simple geometry management for child 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**.

BulletinBoard uses the *XmQTtakesDefault* trait, and holds the *XmQTdialogShellSavvy* and *XmQTspecifyRenderTable* traits.

Classes

BulletinBoard inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNallowOverlap	XmCAllowOverlap	Boolean	True	CSG
XmNautoUnmanage	XmCAutoUnmanage	Boolean	True	CG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNcancelButton	XmCWidget	Widget	NULL	SG
XmNdefaultButton	XmCWidget	Widget	NULL	SG
XmNdefaultPosition	XmCDefaultPosition	Boolean	True	CSG
XmNdialogStyle	XmCDialogStyle	unsigned char	dynamic	CSG
XmNdialogTitle	XmCDialogTitle	XmString	NULL	CSG
XmNfocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNmapCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	10	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNnoResize	XmCNoResize	Boolean	False	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_ANY	CSG
XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_-OUT	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNtextTranslations	XmCTranslations	XtTranslations	NULL	C
XmNunmapCallback	XmCCallback	XtCallbackList	NULL	C

XmNallowOverlap
Controls the policy for overlapping child widgets. If this resource is 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 button descendants. See the **XmNbuttonRenderTable** resource.

XmNbuttonRenderTable
Specifies the render table used for BulletinBoard's button descendants. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the render table is initialized to the **XmBUTTON_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

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. Controls the positioning of a DialogShell managing a BulletinBoard. If the BulletinBoard is not being managed by a

DialogShell, then this resource has no effect. If **XmNdefaultPosition** is True, the DialogShell will center itself at the center of its own parent. For example, if the parent of the DialogShell is an ApplicationShell, then the center of the DialogShell will be at the same coordinates as the center of the ApplicationShell. If the DialogShell becomes unmapped (but stays managed) and then remapped, this resource has no influence on the DialogShell's position. If this resource is False, the DialogShell does not automatically center itself.

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.

Posting a modal dialog in response to a button down or key down event (via translation actions or callbacks) can cause the corresponding button up or key up event to be lost. For example, posting a modal dialog from an **XmNincrementCallback** of **XmScrollBar** will cause the loss of the button up event, causing the **XmScrollBar** to auto-increment indefinitely.

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*. The direction of the title is based on the **XmNlayoutDirection** resource of the widget.

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 label descendants. See the **XmNlabelRenderTable** resource.

XmNlabelRenderTable
Specifies the render table used for BulletinBoard's label descendants. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the render table is initialized to the **XmLABEL_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

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 text descendants. See the **XmNtextRenderTable** resource.

XmNtextRenderTable
Specifies the render table used for BulletinBoard's text descendants. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the render table is initialized to the **XmTEXT_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

XmNtextTranslations
It 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**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallback- List	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallback- List	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow-Pixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	XmCReadOnly	WidgetList	NULL	G

XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	N/A
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallback- List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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:

KeyosfCancel:
: 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.

KeyosfActivate:
: 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 KeyosfActivate is called before the BulletinBoard actions associated with KeyosfActivate.

  In a multiline Text widget, any KeyosfActivate event except KeyEnter calls the Text action associated with KeyosfActivate, then the BulletinBoard actions associated with KeyosfActivate. 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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateBulletinBoard**(3), **XmCreateBulletinBoardDialog**(3), **XmDialogShell**(3), and **XmManager**(3).

XmCascadeButton

Purpose

The CascadeButton widget class

Synopsis

#include <Xm/CascadeB.h>

Description

CascadeButton links two menu panes or a MenuBar to a menu pane.

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 menu pane 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 menu pane; or it can include only a label or a pixmap when it is in a MenuBar. The positioning of the PulldownMenu with respect to the CascadeButton depends on the **XmNlayoutDirection** resource of the MenuShell.

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. **BMenu** also performs the **BSelect** actions in all types of menu systems.

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 menu pane 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 the Cascadebutton is in a Pulldown or Popup menu pane 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.

CascadeButton uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits.

Classes

CascadeButton inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C

XmNcascadePixmap	XmCPixmap	Pixmap	dynamic	CSG
XmNcascadingCallback	XmCCallback	XtCallbackList	NULL	C
XmNmappingDelay	XmCMappingDelay	int	180 ms	CSG
XmNsubMenuId	XmCMenuWidget	Widget	NULL	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 menu pane 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. The positioning of the cascade pixmap to either the left of right of the widget, and the direction of the arrow, depend on the **XmNlayoutDirection** resource of the MenuShell.

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 menu pane. The value must not be negative.

XmNsubMenuId
Specifies the widget ID for the Pulldown menu pane to be associated with this CascadeButton. The specified menu pane is displayed when the CascadeButton becomes armed. The menu pane must have been created with the appropriate parentage depending on the type of menu

used. See **XmCreateMenuBar**(3), **XmCreatePulldownMenu**(3), and **XmCreatePopupMenu**(3) 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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	N/A
XmNaccelerator- Text	XmCAccelerator- Text	XmString	NULL	N/A
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	dynamic	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	0	CSG
XmNmarginRight	XmCMarginRight	Dimension	dynamic	CSG
XmNmarginTop	XmCMarginTop	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	dynamic	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonic-CharSet	XmCMnemonic-CharSet	String	XmFONTLIST_-DEFAULT_TAG	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString-Direction	dynamic	CSG

XmPrimitive Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOn- Enter	XmCHighlight-OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlight-Pixmap	Pixmap	dynamic	CSG
XmNhighlight-Thickness	XmCHighlight-Thickness	Dimension	0	CSG
XmNlayoutDirection	XmCLayout-Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigation-Type	XmNavigation- Type	XmNONE	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadow- Thickness	XmCShadow-Thickness	Dimension	2	CSG
XmNtopShadow- Color	XmCTopShadow-Color	Pixel	dynamic	CSG
XmNtopShadow-Pixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	dynamic	G
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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**.

Note that altering translations in **#override** or **#augment** mode is undefined.

The following list describes the translations for a CascadeButton in a MenuBar. The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<EnterWindow>Normal:
MenuBarEnter()

<LeaveWindow>Normal:
MenuBarLeave()

<Btn2Down>:
ProcessDrag()

c<BtnDown>:
MenuButtonTakeFocusUp()

c<BtnUp>:
MenuButtonTakeFocusUp()

≈c<BtnDown>:
MenuBarSelect()

≈c<BtnUp>:
DoSelect()

:<Key>osfSelect:
KeySelect()

:<Key>osfActivate:
KeySelect()

:<Key>osfHelp:
Help()

:<Key>osfCancel:
  CleanupMenuBar()

≈s<Key>Return:
  KeySelect()

≈s<Key>space:
  KeySelect()

The following list describes the translations for a CascadeButton in a PullDown or Popup MenuPane. In a Popup menu system, **Btn3** also performs the **Btn1** systemitem class="Constant"s.

<EnterWindow>:
  DelayedArm()

<LeaveWindow>:
  CheckDisarm()

<Btn2Down>:
  ProcessDrag()

c<BtnDown>:
  MenuButtonTakeFocus()

c<BtnUp>:
  MenuButtonTakeFocusUp()

≈c<BtnDown>:
  StartDrag()

≈c<BtnUp>:
  DoSelect()

:<Key>osfSelect:
  KeySelect()

:<Key>osfActivate:
  KeySelect()

:<Key>osfHelp:
  Help()

:<Key>osfCancel:
  CleanupMenuBar()

≈s<Key>Return:
KeySelect()

≈s<Key>space:
KeySelect()

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 **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.

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 **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.

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**(3).

Related Information

Core(3), **XmCascadeButtonHighlight**(3), **XmCreateCascadeButton**(3), **XmCreateMenuBar**(3), **XmCreatePulldownMenu**(3), **XmCreatePopupMenu**(3), **XmLabel**(3), **XmPrimitive**(3), and **XmRowColumn**(3).

XmCascadeButtonGadget

Purpose

The CascadeButtonGadget widget class

Synopsis

#include <Xm/CascadeBG.h>

Description

CascadeButtonGadget links two menu panes, a MenuBar to a menu pane, or an OptionMenu to a menu pane.

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 menu pane 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 menu pane; or it can include only a label or a pixmap when it is in an OptionMenu. The positioning of the PulldownMenu with respect to the CascadeButton depends on the **XmNlayoutDirection** resource of the MenuShell.

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. **BMenu** also performs the **BSelect** actions in all types of menu systems.

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 menu pane 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 menu pane 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.

CascadeButtonGadget uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits.

Classes

CascadeButtonGadget inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNcascadePixmap	XmCPixmap	Pixmap	dynamic	CSG
XmNcascadingCallback	XmCCallback	XtCallbackList	NULL	C
XmNmappingDelay	XmCMappingDelay	int	180 ms	CSG
XmNsubMenuId	XmCMenuWidget	Widget	NULL	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 menu pane 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**.

The positioning of the cascade pixmap to either the left of right of the widget, and the direction of the arrow, depend on the **XmNlayoutDirection** resource of the MenuShell.

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 menu pane. The value must not be negative.

XmNsubMenuId

Specifies the widget ID for the Pulldown menu pane to be associated with this CascadeButtonGadget. The specified menu pane is displayed when the CascadeButtonGadget becomes armed. The menu pane must have been created with the appropriate parentage depending on the type of menu used. See **XmCreatePulldownMenu**(3), **XmCreatePopupMenu**(3), and **XmCreateOptionMenu**(3) for more information on the menu systems.

Inherited Resources

CascadeButtonGadget 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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	N/A
XmNacceleratorText	XmCAcceleratorText	XmString	NULL	N/A
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	dynamic	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	0	CSG
XmNmarginRight	XmCMarginRight	Dimension	dynamic	CSG
XmNmarginTop	XmCMarginTop	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	dynamic	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG

XmNmnemonic-CharSet	XmCMnemonicCharSet	String	dynamic	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRender- Table	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CSG

XmGadget Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNbottomShadow-Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight- OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlight-Thickness	XmCHighlight-Thickness	Dimension	0	CSG
XmNlayoutDirection	XmNCLayout- Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmNONE	CSG
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow-Pixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Object Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	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, Btn3 also performs the Btn1 actions).

Btn1Down: 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.

Btn1Up: 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.

KeyosfActivate:
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.

KeyosfSelect:
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.

KeyosfHelp: 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.

KeyosfCancel:
In a MenuBar, disarms the CascadeButtonGadget 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. 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 **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 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**(3).

Related Information

Object(3), **RectObj**(3), **XmCascadeButtonHighlight**(3), **XmCreateCascadeButtonGadget**(3), **XmCreatePulldownMenu**(3), **XmCreatePopupMenu**(3), **XmCreateOptionMenu**(3), **XmGadget**(3), **XmLabelGadget**(3), and **XmRowColumn**(3).

XmComboBox

Purpose

The ComboBox widget class

Synopsis

#include <Xm/ComboBox.h>

Description

XmComboBox combines the capabilities of a single-line TextField widget and a List widget. It allows users to perform opoerations like typing and pasting information, and it also provides a list of possible choices that the user can select from to complete the TextField entry field. The list can either be displayed at all times or can be dropped down by the user. When the list portion of the ComboBox is hidden, users are given a visual cue (a downward-pointing arrow) next to the TextField field. The position of the arrow relative to the TextField field depends on the **XmNlayoutDirection** resource of the widget. This version of the ComboBox is called the "drop-down" ComboBox. Drop-down ComboBoxes are useful when screen space is limited, or when users will complete the text entry field more often by typing text than by selecting the entry field text from the list. The user can access the drop-down ComboBox in either one of two ways:

- By clicking and releasing Btn1 on the downward-pointing arrow, which pops the list up, and the list stays up. A later selection of an item in the list will cause the item to appear in the text entry field, and the list will unpost itself.
- By pressing Btn1 on the downward-pointing arrow, dragging it to a list item, and then releasing it there, which selects that item. The list disappears, and the selected item appears in the text entry field.

The application provides an array of strings that fill the list. At creation time, string items can be passed to the ComboBox via an arglist. Each string becomes an item in the list, with the first string becoming the item in position 1, the second string becoming the item in position 2, and so on. The size of the list is set by specifying the number of items that are visible in the list (**XmNvisibleItemCount**). If the number of items in the list exceeds the number of items that are visible, a vertical scroll bar will automatically appear that allows the user to scroll through a large number of items.

ComboBox creates two child widgets: a TextField widget for entering text and a ScrolledWindow containing a List for the list of items. The name of the items list itself is **List**, and the name of the TextField is **Text**. The application or user can specify resource values for these widgets in a resource file, and the application can use **XtNameToWidget** (specifying **"*List"** for the items list or **"*Text"** for the TextField widget) to obtain the widget IDs of the descendant widgets. At creation time, ComboBox passes appropriate resource values in the creation arglist, including **XmNitems**, to the items list. Note that the result of providing the **XmNdestroyCallback** resource in the creation *arglist* is unspecified. The application should use the **XtAddCallback** function to add callbacks to the appropriate widget (TextField or List) after creating it.

ComboBox forces the following resource values on its List child:

- If **XmNcomboBoxType** is **XmCOMBO_BOX**, **XmNtraversalOn** is forced to False.
- **XmNhighlightThickness** is forced to 2 in a drop-down ComboBox and to 0 in other types of ComboBoxes.
- **XmNborderWidth** is forced to 0.
- **XmNnavigationType** is forced to **XmNONE**.
- **XmNselectionPolicy** is forced to **XmBROWSE_SELECT**.
- **XmNlistSizePolicy** is forced to **XmVARIABLE**.
- **XmNspacing** is forced to 0.
- **XmNvisualPolicy** is forced to **XmVARIABLE**.
- **XmNselectedPositions** is forced to NULL.
- *XmNselectedPositionsCount* is forced to 0.

When **XmNcomboBoxType** is **XmDROP_DOWN_LIST**, ComboBox forces the following resource values on its TextField child:

- **XmNeditable** is forced to False.
- **XmNcursorPositionVisible** is forced to False.
- **XmNshadowThickness** is forced to 0.

By contrast, when **XmNcomboBoxType** is **XmCOMBO_BOX** or **XmDROP_DOWN_COMBO_BOX**, ComboBox forces the following resource values on its TextField child:

- **XmNeditable** is forced to True.
- **XmNcursorPositionVisible** is forced to True.
- **XmNeditMode** is forced to **XmSINGLE_LINE_EDIT**.

ComboBox always forces the values of the following resources on the TextField:

- **XmNnavigationType** is forced to **XmNONE**.
- **XmNhighlightThickness** is forced to 0.
- **XmNborderWidth** is forced to 0.

ComboBox allows a single item to be selected in two ways: by selecting the desired item from the List or by entering text into the TextField. ComboBox does not automatically select a list item if the user types that string into the TextField. It selects the item when the user presses **KActivate** or moves the focus. ComboBox supports the Browse Select selection model of List (see the **XmList** reference page for a description of this model), so selections are mutually exclusive. Selecting an item from the list causes that item to be displayed in the TextField portion of the ComboBox. If an application sets the **XmNvalue** resource of TextField, that string is shown in the TextField. If the application has not provided any list items, or if there is no current selection, the TextField is empty.

The TextField in the ComboBox widget can be either editable or noneditable, depending on the value of the **XmNcomboBoxType** resource.

If the TextField is editable, the user can type into it. When the user presses the Return key, the ComboBox will compare the typed entry to the items in the List. If there is an exact match, then the matched List item is selected. If the application wishes to validate the entered text (for example, to ensure that the typed selection is a valid one), it can do so by setting **XmNmodifyVerifyCallback** on the TextField widget.

If the TextField is noneditable, typing text may invoke a matching algorithm that will attempt to match the entered text with items in the list. The specific matching algorithm applied, which may be none, is determined by the value of the **XmNmatchBehavior** resource in ComboBox, which can be either **XmNONE** or **XmQUICK_NAVIGATE**. A value of **XmNONE** indicates that no matching algorithm will occur. A value of **XmQUICK_NAVIGATE** indicates that when the List widget has focus, one-character navigation is supported. In this algorithm, if the typed character is the initial character of some item in the List, this algorithm causes that item to be navigated to and selected, and the item is displayed in the TextField. Subsequently typing the same character will cycle among the items with the same first character.

Regardless of the selection mechanism used (either selected directly from the List or typed into the TextField), when an item in the List is selected, that item is highlighted in the List. In addition, the selected item is displayed in the TextField of the ComboBox. If the user performs an action that would move focus away from ComboBox, or selects a List item, the **XmNselectionCallback** callbacks are invoked to notify the application of the current contents of the TextField (or choice). The application then takes whatever action is required based on those contents (or choice).

ComboBox uses the *XmQTspecifyRenderTable* trait and holds the *XmQTaccessTextual* trait.

Classes

XmComboBox inherits behavior, resources, and traits from **Core**, **Composite**, and **XmManager** classes.

The class pointer is *xmComboBoxWidgetClass*.

The class name is **XmComboBox**.

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 at creation time (C), set by using **XtSetValues** (S), retrieved by using **XtGetValues** (G), or is not applicable (N/A).

XmComboBox Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNarrowSize	XmCArrowSize	Dimension	dynamic	CSG
XmNarrow- Spacing	XmCArrowSpacing	Dimension	dynamic	CSG
XmNcolumns	XmCColumn	short	dynamic	CSG

XmNcomboBox- Type	XmCComboBox- Type	unsigned char	XmCOMBO_- BOX	CG
XmNfontList	XmCFontList	XmFontList	NULL	CSG
XmNhighlight- Thickness	XmCHighlight- Thickness	Dimension	2	CSG
XmNitemCount	XmCItemCount	int	dynamic	CSG
XmNitems	XmCItems	XmString- Table	dynamic	CSG
XmNlist	XmCList	Widget	dynamic	G
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmatchBehavior	XmCMatch- Behavior	unsigned char	dynamic	CSG
XmNpositionMode	XmCPositionMode	XtEnum	XmZERO_- BASED	CG
XmNrenderTable	XmCRenderTable	XmRender- Table	dynamic	CSG
XmNselectedItem	XmCSelectedItem	XmString	NULL	CSG
XmNselected- Position	XmCSelected- Position	int	dynamic	CSG
XmNselection- Callback	XmCCallback	XmCallback- List	NULL	C
XmtextField	XmCTextField	Widget	dynamic	G
XmNvisible- ItemCount	XmCVisibleItem- Count	int	10	CSG

XmNarrowSize

Specifies the width of the arrow. The default size depends on the size of the text, as well as the size of the ComboBox.

XmNarrowSpacing

Specifies the space between the text and arrow visual in pixels. The default value is obtained from the **XmNmarginWidth** resource.

XmNcolumns

Specifies the number of columns in the text field. If unset, the text field’s value is used. Refer to the *XmTextField* man page for more detailed information.

XmNcomboBoxType
Specifies the type of ComboBox to be created. This can be one of the following:

XmCOMBO_BOX
Generates a ComboBox where the list is always displayed, and the text entry field is editable.

XmDROP_DOWN_COMBO_BOX
Generates a ComboBox where the list is hidden unless specifically requested, and the text entry field is editable.

XmDROP_DOWN_LIST
Generates a ComboBox where the list is hidden unless specifically requested, and the text entry field is noneditable.

XmNfontList
Specifies the fontlist associated with **XmComboBox**. The fontlist is an obsolete construct, and has been superseded by the render table. It is included for compatibility with earlier versions of Motif, and for applications that do not easily support render tables. The default fontlist is derived from the default render table, and if both a fontlist and a render table are specified, the render table takes precedence.

XmNhighlightThickness
Specifies the thickness of the highlighting rectangle.

XmNitemCount
Specifies the number of items in the list. If unset, the lists's value is used. Refer to the *XmList* man page for more detailed information.

XmNitems Specifies the items in the list. If unset, the lists's value is used. Refer to the *XmList* man page for more detailed information.

XmNlist The list widget.

XmNmarginWidth
Specifies the horizontal spacing between the child widgets and the boundary of the ComboBox.

XmNmarginHeight
Specifies the vertical spacing between the child widgets and the boundary of the ComboBox.

XmNmatchBehavior

Defines the matching algorithm applied to match the text typed by the user in the TextField field with items in the list. The current values are **XmNONE** and **XmQUICK_NAVIGATE**, as follows:

XmNONE Indicates that there is no assigned matching algorithm.

XmQUICK_NAVIGATE

Is only valid for noneditable ComboBoxes (a value of **XmDROP_DOWN_LIST** of the resource, **XmNcomboBoxType**). This algorithm supports 1-character navigation when the List widget has focus. If the typed character is the initial character of some item in the List, this algorithm causes that item to be navigated to and selected. Subsequently typing the same character will cycle among the items with the same first character.

XmNpositionMode

Specifies how the value of the **XmNselectedPosition** resource and the **item_position** field of the callback structure are to be interpreted. Note that the convenience functions *XmComboBoxDeletePos* and *XmComboBoxAddItem* are not affected by this resource, and (like *XmList*) always use 1-based positions. Valid values for this resource are:

XmZERO_BASED

(The DtComboBox compatibility mode is the default.)**XmNselectedPosition** is in **[0,itemcount-1]**. The **item_position** in the *XmComboBoxCallbackStruct* is 0 if the first element in the list was selected. Note that 0 is also returned if no element in the list was selected (that is, a new item was entered in the text field).

XmONE_BASED

(Motif mode) Both the resource value and the callback fields are 1-based. This is consistent with other Motif widgets.

XmNrenderTable

Specifies the render table associated with ComboBox. This render table is used in both the TextField field and the List in the ComboBox. This is used in conjunction with the **XmNvisibleItemCount** resource of the List to determine the height of the ComboBox widget.

If this value is NULL at initialization, and if the widget parent is **XmBulletinBoard** or its subclasses, **VendorShell** or its subclasses, or **XmMenuShell**, then the widget parent provides the default render table associated with the widget. If both a render table and a fontlist are specified, the render table will take precedence.

XmNselectedItem
Specifies a compound string that represents the current selection of the ComboBox. The selected item is the content of the ComboBox text entry field.

XmNselectedPosition
If the selection in the ComboBox is an item in the list, this is the index of the selected item in the list. If no item in the list is selected, this is 0.

XmNselectionCallback
Specifies the list of callbacks called when an item is selected. The reason field in the **XmComboBoxCallbackStruct** is **XmCR_SELECT**.

XmNtextField
The text field widget.

XmNvisibleItemCount
Specifies the number of visible items in the list. This will override any value specified for the list. Refer to the *XmList* man page for more detailed information.

Inherited Resources

ComboBox inherits behavior and resources from 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**	**Type**	**Default**	**Access**
XmNbottom-ShadowColor	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottom-ShadowPixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelp- Callback	XmCCallback	XtCallbackList	NULL	C

XmNhighlight-Color	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlight-Pixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitial- Focus	XmCInitialFocus	Widget	NULL	CSG
XmNlayout-Direction	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigation-Type	XmCNavigationType	XmNavigation-Type	XmSTICKY_TAG_-GROUP	CSG
XmNpopup-HandlerCallback	XmCCallback	XtCallback- List	NULL	C
XmNshadow-Thickness	XmCShadowThickness	Dimension	2	CSG
XmNstring-Direction	XmCStringDirection	XmString-Direction	dynamic	CG
XmNtop-ShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtop-ShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestor-Sensitive	XmCSensitive	Boolean	dynamic	G

XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroy- Callback	XmCCallback	XtCallback- List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources- Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen- Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback Information

A pointer to the following structure is passed to each callback. The callback structure is defined as follows:

```
typedef struct
{
        int reason;
        XEvent *event;
        XmString item_or_text;
        int item_position;
} XmComboBoxCallbackStruct;
```

reason Indicates why the callback was invoked.

event	Points to the *XEvent* that triggered the callback. It can be NULL.
item_or_text	The contents of the text field at the time the event caused the callback. The *item_or_text* parameter points to a temporary storage space that is reused after the callback is finished. If an application needs to save the item, it should copy *item_or_text* into its own data space.
item_position	The position of item in the list's **XmNitems** 1-based array. If this is 0, it means that the *item_or_text* was not selected from the List.

Translations

The ComboBox translations are listed below.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

Note: The KPageUp and KPageDown translations do not take effect unless the **List** child widget is posted.

<Btn1Down>:
CBArmAndDropDownList()

<Btn1Up>: **CBDisarm()**

The following lists the List translations in the drop-down list. When ComboBox **XmNcomboBoxType** is **XmDROP_DOWN_LIST**, **osfActivate**, **osfCancel**, and **Return** are overriden by ComboBox systemitem class="Constant"s.

:c <Key>osfDown:
CBDropDownList()

:c <Key>osfUp:
CBDropDownList()

:<Key>osfCancel:
CBCancel()

:<Key>osfActivate:
CBActivate()

≈s ≈m ≈a<Key>Return:
CBActivate()

Accelerators

The following accelerators are added to ComboBox and its children. The accelerators may not directly correspond to a translation table. If the translation is not listed below, it may depend on the context of the event.

:c <Key>osfUp:
CBDropDownList()

:<Key>osfUp:
CBListAction(*Up***)**

:c <Key>osfDown:
CBDropDownList()

:<Key>osfDown:
CBListAction(*Down***)**

:c <Key>osfBeginLine:
CBListAction(*ListBeginData***)**

:c <Key>osfEndLine:
CBListAction(*ListEndData***)**

:<Key>**osfPageUp**:
CBListAction(*ListPrevPage***)**

:<Key>**osfPageDown**:
CBListAction(*ListNextPage***)**

A drop-down ComboBox also adds accelerators to its List child. Aside from the accelerators that are already listed in this section, those accelerators are the default TextField key translations.

Action Routines

The **XmComboBox** action routines are as follows:

CBActivate():
Calls the **XmNselectionCallback** callbacks. If the **XmNcomboBoxType** is **XmDROP_DOWN_COMBO_BOX** or **XmDROP_DOWN_LIST**, it unposts the list. If the parent is a manager, passes the event to the parent.

CBArmAndDropDownList():
If the pointer is within the down arrow, draws the shadow of the arrow in the selected state, and then posts the list.

CBCancel():

If the **XmNcomboBoxType** is **XmDROP_DOWN_COMBO_BOX** or **XmDROP_DOWN_LIST**, pops down the list. If the parent is a manager, passes the event to the parent.

CBDisarm():

Redraws the arrow in an unselected state.

CBDropDownList():

If **XmNcomboBoxType** is **XmDROP_DOWN** or **XmDROP_DOWN_LIST**, and list is not displayed, posts the list. If list is displayed, it unposts the list.

CBListAction(*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 **XmNbrowseSelectionCallback** selection callback.

CBListAction(*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 **XmNbrowseSelectionCallback** selection callback.

CBListAction(*ListPrevPage*):

Scrolls the list to the previous page, moving the location cursor to a new item. This action also selects the new item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks. If the ComboBox is not a drop-down type, then this action does nothing.

CBListAction(*ListNextPage*):

Scrolls the list to the next page, moving the location cursor to a new item. This action also selects the new item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks. If the ComboBox is not a drop-down type, then this action does nothing.

CBListAction(*Up*):

Moves the location cursor to the previous item in the list. This action also selects the previous item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks. Note that, unlike the List **ListPrevItem** action, this action wraps around.

CBListAction(*Down*):

Moves the location cursor to the next item in the list. This action also selects the next item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks. Note that, unlike the List **ListNextItem** action, this action wraps around.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Errors/Warnings

The toolkit will display a warning if the application tries to set the value of **XmNlist** or the **XmNtextField** resource, which are read-only (marked G in the resource table).

Related Information

Composite(3), **Core**(3), **XmCreateComboBox**(3), **XmList**(3), **XmManager**(3), and **XmTextField**(3).

XmCommand

Purpose

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.

Descendants

Command automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
ItemsList	**XmList**	command-history list region
ItemsListSW	**XmScrolledWindow**	the ScrolledWindow parent of **ItemsList**
Selection	**XmLabelGadget**	command-line prompt
Text	**XmTextField**	command-line text-input field

Classes

Command inherits behavior, resources, and traits from **Core**, **Composite**, **Constraint**, **XmManager**, **XmBulletinBoard**, and **XmSelectionBox**.

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	Type	Default	Access
XmNcommand	XmCTextString	XmString	""	CSG
XmNcommand- ChangedCallback	XmCCallback	XtCallbackList	NULL	C
XmNcommandEntered- Callback	XmCCallback	XtCallbackList	NULL	C
XmNhistoryItems	XmCItems	XmStringTable	NULL	CSG
XmNhistoryItemCount	XmCItemCount	int	0	CSG
XmNhistoryMaxItems	XmCMaxItems	int	100	CSG
XmNhistoryVisibleItemCount	XmCVisibleItemCount	int	dynamic	CSG
XmNpromptString	XmCPromptString	XmString	dynamic	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 after each time 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

XmNlayoutDirection resource and the locale. In the C locale the default is > (right angle bracket).

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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**	**Type**	**Default**	**Access**
XmNapplyCallback	XmCCallback	XtCallbackList	NULL	N/A
XmNapply-LabelString	XmCApplyLabel- String	XmString	dynamic	N/A
XmNcancelCallback	XmCCallback	XtCallbackList	NULL	N/A
XmNcancelLabel-String	XmCCancel- LabelString	XmString	dynamic	N/A
XmNchildPlacement	XmCChild- Placement	unsigned char	XmPLACE_ABOVE_-SELECTION	CSG
XmNdialogType	XmCDialogType	unsigned char	XmDIALOG_-COMMAND	G
XmNhelp-LabelString	XmCHelpLabel- String	XmString	dynamic	N/A
XmNlistItemCount	XmCItemCount	int	0	CSG
XmNlistItems	XmCItems	XmStringTable	NULL	CSG
XmNlistLabelString	XmCListLabel- String	XmString	NULL	N/A
XmNlistVisibleItem-Count	XmCVisibleItem- Count	int	dynamic	CSG
XmNminimize-Buttons	XmCMinimizeButtons	Boolean	False	N/A
XmNmustMatch	XmCMustMatch	Boolean	False	N/A

XmNnoMatch-Callback	XmCCallback	XtCallbackList	NULL	N/A
XmNokCallback	XmCCallback	XtCallbackList	NULL	N/A
XmNokLabelString	XmCOkLabelString	XmString	dynamic	N/A
XmNselectionLabel-String	XmCSelection-LabelString	XmString	dynamic	CSG
XmNtextAccelerators	XmCText- Accelerators	XtAccelerators	default	C
XmNtextColumns	XmCColumns	short	dynamic	CSG
XmNtextString	XmCTextString	XmString	""	CSG

XmBulletinBoard Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNallowOverlap	XmCAllowOverlap	Boolean	True	CSG
XmNautoUnmanage	XmCAutoUnmanage	Boolean	False	N/A
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	N/A
XmNbuttonRender-Table	XmCButtonRender-Table	XmRenderTable	dynamic	CSG
XmNcancelButton	XmCWidget	Widget	NULL	N/A
XmNdefaultButton	XmCWidget	Widget	NULL	N/A
XmNdefaultPosition	XmCDefaultPosition	Boolean	False	CSG
XmNdialogStyle	XmCDialogStyle	unsigned char	dynamic	CSG
XmNdialogTitle	XmCDialogTitle	XmString	NULL	CSG
XmNfocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabel-RenderTable	XmCLabelRender- Table	XmRenderTable	dynamic	CSG
XmNmapCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	10	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNnoResize	XmCNoResize	Boolean	False	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_- NONE	CSG
XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_- OUT	CSG

XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRender- Table	XmRenderTable	dynamic	CSG
XmNtextTranslations	XmCTranslations	XtTranslations	NULL	C
XmNunmapCallback	XmCCallback	XtCallbackList	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadow-Thickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CG
XmNtopShadow-Color	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow-Pixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	N/A
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallback- List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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 in bytes of the **XmString** value. This member is obsolete and exists for compatibility with earlier releases.

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:

SelectionBoxUpOrDown(*Previous*|*Next*|*First*|*Last*):

When called with an argument of **Previous**, or 0 (zero) for compatibility, selects the previous item in the history list and replaces the text with that item.

When called with an argument of **Next**, or 1 for compatibility, selects the next item in the history list and replaces the text with that item.

When called with an argument of **First**, or 2 for compatibility, selects the first item in the history list and replaces the text with that item.

When called with an argument of **Last**, or 3 for compatibility, 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:

KeyosfCancel:
: If the parent of the Command is a manager, the event is passed to the parent.

KeyosfActivate 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**.

BtnDown**(2+)** or KeyosfActivate 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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmBulletinBoard**(3), **XmCommandAppendValue**(3), **XmCommandError**(3), **XmCommandGetChild**(3), **XmCommandSetValue**(3), **XmCreateCommand**(3), **XmManager**(3), and **XmSelectionBox**(3).

XmContainer

Purpose

The Container widget class

Synopsis

#include <Xm/Container.h>

Description

Container manages child widgets that have the **ContainerItem** trait. These child widgets can be viewed in several different layout formats, selected using different selection types and techniques, and directly manipulated by the user.

Three different formats or views are supported by the Container. They are specified via the **XmNentryViewType** resource:

- **XmLARGE_ICON**
- **XmSMALL_ICON**
- **XmANY_ICON**

Three different layout types are supported by the Container. They are specified by the **XmNlayoutType** resource:

- **XmSPATIAL**
- **XmOUTLINE**
- **XmDETAIL**

In the **XmSPATIAL** layout type, several other resources (**XmNspatialStyle**, **XmNspatialIncludeModel**, **XmNspatialSnapModel**) control the positioning of the items within the Container; an application programmer can specify these resources so that the Container strictly enforces the position of each item or so that the Container positions items according to constraint resources specified for each item. The user, however, can alter the position of an item in the **XmSPATIAL** layout type within the Container by direct manipulation; for example, by pressing **BTransfer** over the item and then dragging and releasing **BTransfer** over some location within the Container.

In the **XmOUTLINE** layout type, the Container controls the positioning of the items. Items within the Container can have parent-child relationships between them. Each item's **XmNentryParent** resource can specify another item in the same Container as its parent; items with a non-NULL value for **XmNentryParent** can only be displayed in the **XmOUTLINE** layout type. In the **XmOUTLINE** layout type, items are positioned in a tree configuration with connecting lines drawn to illustrate the relationships. Items are positioned top to bottom in the order specified by **XmNpositionIndex** within their parent. Container positions a PushButton next to each item that has a parent relationship to other items. The PushButton contains a pixmap to illustrate whether the child items are shown or not; the user can activate the PushButton to toggle between showing or hiding the child items. Direct manipulation to alter the position of the item is not supported in the **XmOUTLINE** layout type. Note that the **XmNtraversalOn** resource of the PushButtons created by Container are set to False.

The **XmDETAIL** layout type is the same as **XmOUTLINE**, except that each item can also display additional information as rows in columns with column headers specified in the **XmNdetailColumnHeading** resources. In each item row, the item's detail information (see the reference page on **XmIconGadget** for a description of the *XmNentryDetail* resource) is displayed. Items are positioned top to bottom in the order specified by **XmNpositionIndex** within the parent.

Selection

When a child widget of the container is selected, the container specifies that the item should display the appropriate visual information to the user via the **ContainerItem** trait. The application program is notified of selection changes through **XmNselectionCallback**.

The container uses four selection policies:

- Single
- Browse
- Multiple
- Extended

In Single Select and Browse Select modes, only one item can be selected at a time. Pressing **BSelect** on an item selects it and deselects any other selected item. Pressing **BSelect** over an empty space in the Container deselects all items. In Browse Select, dragging **BSelect** moves the selection as the pointer is moved.

In Multiple Select and Extended Select modes, any number of items can be selected at the same time. In Multiple Select, pressing and dragging **BSelect** or **BExtend** to specify an item, range of items, or group of discontiguous items causes the selection states of those items to be toggled. In Extended Select, pressing and dragging **BSelect** to indicate an item, range of items, or group of discontiguous items selects those items and deselects all others. Pressing and dragging **BExtend** in Extended Select to indicate an item, range of items, or discontiguous group of items causes the selection states of those items to be toggled.

Several techniques are available to indicate an item, range of items, or group of discontiguous items in the Multiple Select and Extended Select modes.

In the **XmSPATIAL** and **XmOUTLINE** layout types, the **XmNselectionTechnique** resource specifies the techniques to be used to indicate items. The default specification of **XmTOUCH_OVER** allows both the Random-Swipe and Marquee techniques to be used when *XmNlayoutStyle* is **XmSPATIAL**. The default specification of **XmTOUCH_OVER** allows the Range-Swipe, Range-Click, and Marquee techniques to be used when *XmNlayoutStyle* is **XmOUTLINE**.

Discontiguous groups of items can be selected using the Random-Swipe technique. In the Random-Swipe technique, pressing **BSelect** (or **BExtend**) over an item and dragging **BSelect** over other items selects all of those items. Only those items that pointer passed over are selected.

In the Range-Swipe technique, the user presses **BSelect** (or **BExtend**) over the first item and releases **BSelect** over the last item; all items within the range between the first and last item are selected whether the pointer actually passed over them or not. In the Range-Click technique, the user presses and releases **BSelect** (or **BExtend**) over the first item and then presses and releases **BExtend** over the last item.

In the Marquee technique, pressing **BSelect** (or **BExtend**) over a blank space within the Container indicates the starting point of a Marquee rectangle. Dragging **BSelect** draws a Marquee rectangle (rubberband line) between the starting point and current pointer. All items completely within the Marquee rectangle are selected.

Specifying **XmTOUCH_ONLY** for **XmNselectionTechnique** enforces the Random-Swipe technique even when **BSelect** (or **BExtend**) is pressed over a blank space. Similarly, specifying **XmMARQUEE** enforces the Marquee technique even when **BSelect** (or **BExtend**) is pressed over an item; since the item over which the press occurs is only partially included in the Marquee rectangle, it is not selected. **XmMARQUEE_EXTEND_START** and **XmMARQUEE_EXTEND_BOTH** enforce the Marquee technique and also cause the rectangle to extend automatically around the first item indicated and, for **XmMARQUEE_EXTEND_BOTH**, the last item.

In the **XmDETAIL** layout type, the Range-Swipe and Range-Click techniques are available to indicate a range of items for selection.

Container uses the *XmQTcontainerItem*, *XmQTscrollFrame*, and *XmQTspecifyRenderTable* traits and holds the **XmQTcontainer** and *XmQTtransfer* traits.

Data Transfer Behavior

Container supports dragging of selected items from the widget. Depending on the value of **XmNprimaryOwnership**, Container can also support primary selection.

As a source of data, Container supports the following targets and associated conversions of data to these targets:

locale If the *locale* target matches the widget's locale, the widget transfers the selected items in the encoding of the locale. The value for each item transferred, except the last, includes a trailing separator. Each item value is the **XmNlabelString** of the item.

COMPOUND_TEXT
The widget transfers the selected items as type *COMPOUND_TEXT*. The value for each item transferred, except the last, includes a trailing separator. Each item value is the **XmNlabelString** of the item.

DELETE The widget deletes the selected items.

PIXMAP The widget transfers a list of the pixmap IDs of the selected items as type *DRAWABLE*.

STRING The widget transfers the selected items as type *STRING*. The value for each item transferred, except the last, includes a trailing separator. Each item value is the **XmNlabelString** of the item.

TEXT If the selected items are fully convertible to the encoding of the locale, the widget transfers the selected items in the encoding of the locale. Otherwise, the widget transfers the selected items as type *COMPOUND_TEXT*. The value for each item transferred, except the last, includes a trailing separator. Each item value is the **XmNlabelString** of the item.

_MOTIF_CLIPBOARD_TARGETS
The widget transfers, as type *ATOM*, a list of the targets it supports for immediate transfer for the *CLIPBOARD* selection. These include _MOTIF_COMPOUND_STRING and *PIXMAP*. If the selected items

are fully convertible to *STRING*, these also include *STRING*; otherwise, they also include *COMPOUND_TEXT*.

_MOTIF_COMPOUND_STRING
: The widget transfers the selected items as a compound string in Byte Stream format. The value for each item transferred, except the last, includes a trailing separator. Each item value is the **XmNlabelString** of the item.

_MOTIF_DEFERRED_CLIPBOARD_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets it supports for delayed transfer for the *CLIPBOARD* selection. This widget currently supplies no targets for _MOTIF_DEFERRED_CLIPBOARD_TARGETS.

_MOTIF_DRAG_OFFSET
: The widget transfers a list of two 16-bit numbers, of type *INTEGER*, representing an x and y offset for an item being dragged. This offset is calculated so that, if the offset were added to the x and y coordinates at the drop site, and the dragged pixmap placed at that position, it would correspond to the position the user would expect the pixmap to placed at, based on the drag icon used at the drop site.

_MOTIF_EXPORT_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets to be used as the value of the DragContext's **XmNexportTargets** in a drag-and-drop transfer. These include _MOTIF_COMPOUND_STRING, *PIXMAP*, *COMPOUND_TEXT*, the encoding of the locale, *STRING*, *TEXT*, *BACKGROUND*, and *FOREGROUND*.

As a source of data, Container also supports the following standard Motif targets:

BACKGROUND
: The widget transfers **XmNbackground** as type *PIXEL*.

CLASS
: The widget finds the first shell in the widget hierarchy that has a WM_CLASS property and transfers the contents as text in the current locale.

CLIENT_WINDOW
: The widget finds the first shell in the widget hierarchy and transfers its window as type *WINDOW*.

COLORMAP
: The widget transfers **XmNcolormap** as type *COLORMAP*.

FOREGROUND
: The widget transfers **XmNforeground** as type *PIXEL*.

NAME
: The widget finds the first shell in the widget hierarchy that has a WM_NAME property and transfers the contents as text in the current locale.

TARGETS
: The widget transfers, as type *ATOM*, a list of the targets it supports. These include the standard targets in this list. These also include _MOTIF_COMPOUND_STRING, *PIXMAP*, *COMPOUND_TEXT*, the encoding of the locale, *STRING*, and *TEXT*.

TIMESTAMP
: The widget transfers the timestamp used to acquire the selection as type *INTEGER*.

_MOTIF_RENDER_TABLE
: The widget transfers **XmNrenderTable** if it exists, or else the default text render table, as type *STRING*.

_MOTIF_ENCODING_REGISTRY
: The widget transfers its encoding registry as type *STRING*. The value is a list of NULL separated items in the form of tag encoding pairs. This target symbolizes the transfer target for the Motif Segment Encoding Registry. Widgets and applications can use this Registry to register text encoding formats for specified render table tags. Applications access this Registry by calling **XmRegisterSegmentEncoding** and **XmMapSegmentEncoding**.

As a destination for data, Container supports only the dropping of items being dragged from the same widget. Subclasses and the **XmNdestinationCallback** procedures are responsible for any other data transfers to the widget.

Classes

Container inherits behavior, resources, and traits from the **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is *xmContainerWidgetClass*.

The class name is **XmContainer**.

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 whether 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).

XmContainer Resource Set

Name	Class	Type	Default	Access
XmNautomatic-Selection	XmCAutomatic-Selection	unsigned char	XmAUTO_- SELECT	CSG
XmNcollapsedState-Pixmap	XmCCollapsedState-Pixmap	Pixmap	dynamic	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNdefaultAction-Callback	XmCCallback	XtCallbackList	NULL	C
XmNdestination-Callback	XmCCallback	XtCallbackList	NULL	C
XmNdetailColumn-Heading	XmCDetailColumn-Heading	XmStringTable	NULL	CSG
XmNdetailColumn-HeadingCount	XmCDetailColumn-HeadingCount	Cardinal	0	CSG
XmNdetailOrder	XmCDetailOrder	Cardinal *	NULL	CSG
XmNdetailOrder- Count	XmCDetailOrder-Count	Cardinal	dynamic	CSG
XmNdetailTabList	XmCDetailTabList	XmTabList	NULL	CSG
XmNentryViewType	XmCEntryViewType	unsigned char	XmANY_- ICON	CSG
XmNexpandedState-Pixmap	XmCExpandedState-Pixmap	Pixmap	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	NULL	CSG
XmNlargeCellHeight	XmCCellHeight	Dimension	dynamic	CSG

XmNlargeCellWidth	XmCCellWidth	Dimension	dynamic	CSG
XmNlayoutType	XmCLayoutType	unsigned char	XmSPATIAL	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	0	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	0	CSG
XmNoutline-ButtonPolicy	XmCOutlineButton-Policy	unsigned char	XmOUTLINE_BUTTON_-PRESENT	CSG
XmNoutlineChanged-Callback	XmCCallback	XtCallbackList	NULL	C
XmNoutlineColumn-Width	XmCOutline-ColumnWidth	Dimension	dynamic	CSG
XmNoutline-Indentation	XmCOutline-Indentation	Dimension	40	CSG
XmNoutlineLine- Style	XmCLineStyle	unsigned char	XmSINGLE	CSG
XmNprimary-Ownership	XmCprimary-Ownership	unsigned char	XmOWN_POSSIBLE_-MULTIPLE	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNselectColor	XmCSelectColor	Pixel	dynamic	CSG
XmNselectedObjects	XmCSelectedObjects	WidgetList	NULL	SG
XmNselectedObject-Count	XmCSelected-ObjectCount	unsigned int	0	SG
XmNselection-Callback	XmCCallback	XtCallbackList	NULL	C
XmNselectionPolicy	XmCSelectionPolicy	unsigned char	XmEXTENDED_-SELECT	CSG
XmNselection-Technique	XmCSelection-Technique	unsigned char	XmTOUCH_OVER	CSG
XmNsmall- CellHeight	XmCCellHeight	Dimension	dynamic	CSG
XmNsmallCellWidth	XmCCellWidth	Dimension	dynamic	CSG
XmNspatialInclude-Model	XmCSpatialInclude-Model	unsigned char	XmAPPEND	CSG
XmNspatialResize-Model	XmCSpatialResize-Model	unsigned char	XmGROW_- MINOR	CSG

XmNspatialSnapModel	XmCSpatial-SnapModel	unsigned char	XmNONE	CSG
XmNspatialStyle	XmCSpatialStyle	unsigned char	XmGRID	CSG

XmNautomaticSelection

Indicates whether the Container invokes selection callbacks when each item is selected (or toggled) or whether selection callbacks are not invoked until the user has completed selection actions (for example, the user has released the mouse button). It can have one of the following values:

XmAUTO_SELECT

Makes selection callbacks automatically when each item is selected or toggled. This may also be the value **TRUE**.

XmNO_AUTO_SELECT

Delays selection callbacks until the user has finished selection actions. This may also be the value **FALSE**.

XmNcollapsedStatePixmap

Specifies the pixmap to display on a PushButton next to a Container item with child items, when **XmNoutlineButtonPolicy** is **XmOUTLINE_BUTTON_PRESENT**. **XmNcollapsedStatePixmap** indicates that the child items are not displayed. If set to **XmUNSPECIFIED_PIXMAP**, a default pixmap showing an arrow pointing up is used.

XmNconvertCallback

Specifies a list of callbacks called when the Container is asked to convert a selection. The type of the structure whose address is passed to these callbacks is **XmConvertCallbackStruct**. The reason is **XmCR_OK**.

XmNdefaultActionCallback

Specifies a list of callbacks to call when the user double-clicks an item or presses **Enter** or **Return** over an item. The callback structure is **XmContainerSelectCallbackStruct**. The reason is **XmCR_DEFAULT_ACTION**.

XmNdestinationCallback

Specifies a list of callbacks called when the Container is the destination of a transfer operation. The type of the structure whose address is passed to these callbacks is **XmDestinationCallbackStruct**. The reason is **XmCR_OK**.

XmNdetailColumnHeading
Specifies a table of **XmString**s to display as the headings to columns. If NULL, or if **XmNlayoutType** is not **XmDETAIL**, no heading is displayed.

XmNdetailColumnHeadingCount
Specifies a count of **XmString**s in the table specified for **XmNdetailColumnHeading**.

XmNdetailOrder
Specifies an array of **Cardinal**s that indicate which column detail information, and in which order, each Container child will display its detail information. This resource is ignored if **XmNlayoutType** is not **XmDETAIL**. If NULL, the the default behavior is determined by **XmNdetailOrderCount**.

XmNdetailOrderCount
Specifies a count of **Cardinal**s in the array specified for **XmNdetailOrder**. If **XmNdetailOrder** is NULL and **XmNdetailOrderCount** is not 0, then each Container child displays its detail information in order from column 1 to the **XmNdetailOrderCount** column number. If **XmNdetailOrderCount** is 0, then a default is calculated from the detail order count information of each item accessed via the **ContainerItem** trait.

XmNdetailTabList
Indicates an **XmTabList** specifying the start of each column in the **XmDETAIL** layout. If this resource is set to NULL, then Container calculates an **XmTabList**. This resource is ignored if **XmNlayoutType** is not **XmDETAIL**.

XmNentryViewType
Specifies the view type for all Container children. The view type is specified for each item via the **ContainerItem** trait. It can have one of the following values:

XmANY_ICON
No specification is made for Container children. Children use their own default specifications.

XmLARGE_ICON
The view type for all children is **XmLARGE_ICON**.

XmSMALL_ICON
The view type for all children is **XmSMALL_ICON**.

XmNexpandedStatePixmap
Specifies the pixmap to display on a PushButton next to a Container item with child items, when **XmNoutlineButtonPolicy** is **XmOUTLINE_BUTTON_PRESENT**. **XmNexpandedStatePixmap** indicates that the child items are displayed. If set to **XmUNSPECIFIED_PIXMAP**, a default pixmap showing an arrow pointing down is used.

XmNfontList
Specifies the fontlist associated with **XmContainer**. The fontlist is an obsolete construct and has been superseded by the render table. It is included for compatibility with earlier versions of Motif, and for applications that do not easily support render tables. The default fontlist is derived from the default render table; and if both a fontlist and a render table are specified, the render table takes precedence.

XmNlargeCellHeight
Specifies the height of a cell for **XmGRID** or **XmCELLS** spatial style when **XmNentryViewType** is **XmLARGE_ICON** or **XmANY_ICON**.

XmNlargeCellWidth
Specifies the width of a cell for **XmGRID** or **XmCELLS** spatial style when **XmNentryViewType** is **XmLARGE_ICON** or **XmANY_ICON**.

XmNlayoutType
Specifies the policy for laying out child widgets within the Container. It can have one of the following values:

XmDETAIL
Displays items in the same manner as when the resource is **XmOUTLINE**, except that each item displays detail information next to it.

XmOUTLINE
Displays items in a tree configuration, in **XmNpositionIndex** within **XmNentryParent** order, with connecting lines drawn to show their parent-child relationships.

XmSPATIAL
Displays items according to **XmNspatialStyle**. Items with **XmNentryParent** values are not displayed.

XmNmarginHeight
Specifies the margin spacing at the top and bottom of the Container.

XmNmarginWidth
Specifies the margin spacing at the left and right sides of the Container.

XmNoutlineButtonPolicy
Specifies whether or not to display buttons for users to expand and collapse the display of items. It can have one of the following values:

XmOUTLINE_BUTTON_ABSENT
Do not display the outline buttons.

XmOUTLINE_BUTTON_PRESENT
Display the outline buttons.

XmNoutlineChangedCallback
Specifies a list of callbacks to call when an item's **XmNoutlineState** is changed. The callback structure is **XmContainerOutlineCallbackStruct**. The reason is **XmCR_COLLAPSED** or **XmCR_EXPANDED**, depending on the new value of **XmNoutlineState**.

XmNoutlineColumnWidth
Specifies the width of the first column displayed when **XmNlayoutType** is **XmDETAIL**. Specifies the preferred width of the Container (without the margins) when **XmNlayoutType** is **XmOUTLINE**. If not specified, Container will determine a default value equal to the widest space necessary to display an item's pixmap and **XmNoutlineIndentation**.

XmNoutlineIndentation
Specifies the distance to indent for the display of child items when **XmNlayoutType** is **XmOUTLINE** or **XmDETAIL**.

XmNoutlineLineStyle
Specifies whether to draw lines between items with parent-child relationships when **XmNlayoutType** is **XmOUTLINE** or **XmDETAIL**. It can have one of the following values:

XmNO_LINE
Draws no line.

XmSINGLE
Draws a line one pixel wide.

XmNprimaryOwnership
Specifies whether Container takes ownership of the primary selection when a selection is made inside it. This resource can take the following values:

XmOWN_NEVER
Never takes ownership.

XmOWN_ALWAYS
Always takes ownership.

XmOWN_MULTIPLE
Only takes ownership if more than one element has been selected.

XmOWN_POSSIBLE_MULTIPLE
Only takes ownership if more than one element can be selected at a time.

XmNrenderTable
Specifies the **XmRenderTable** that is inherited by all children of the Container. The default is implementation dependent. If both a render table and a fontlist are specified, the render table will take precedence.

XmNselectColor
Specifies a Pixel that can be accessed by children of the Container and used to indicate that the child is in a selected state. In addition to a Pixel value, the following symbolic values can be specified:

XmDEFAULT_SELECT_COLOR
Specifies a color between the background and the bottom shadow color.

XmREVERSED_GROUND_COLORS
Forces the select color to the foreground color and causes the default color of any text rendered over the select color to be the background color.

HIGHLIGHT_COLOR
Forces the fill color to use the highlight color.

XmNselectedObjectCount
Specifies the number of widgets in the selected items list. The value must be the number of items in **XmNselectedObjects**.

XmNselectedObjects
An array of widgets that represents the Container items that are currently selected, either by the user or by the application.

If the application sets **XmNselectedObjects** to an array of widgets, those array elements that are valid Container items are selected.

XmNselectionCallback
Specifies a list of callbacks to call when an item is selected. The callback structure is **XmContainerSelectCallbackStruct**.
The reason is **XmCR_SINGLE_SELECT**, **XmCR_BROWSE_SELECT**, **XmCR_MULTIPLE_SELECT**, or **XmCR_EXTENDED_MULTIPLE**, depending on **XmNselectionPolicy**.

XmNselectionPolicy
Defines the interpretation of the selection action. This can be one of the following values:

XmSINGLE_SELECT
Allows only single selections.

XmBROWSE_SELECT
Allows "drag and browse" selections.

XmMULTIPLE_SELECT
Allows multiple selections.

XmEXTENDED_SELECT
Allows extended selections.

XmNselectionTechnique
Specifies the selection technique to use when the Container displays items in a 2-dimensional layout (**XmNentryViewType** is **XmLARGE_ICON** or **XmSMALL_ICON**). In the **XmDETAIL** layout, the **XmNselectionTechnique** resource is treated as **XmTOUCH_ONLY**. In either case, it can have one of the following values:

XmMARQUEE
Uses the Marquee technique only.

XmMARQUEE_EXTEND_START
Uses the Marquee technique only and extends the Marquee rectangle around any item under the Marquee start point.

XmMARQUEE_EXTEND_BOTH
Uses the Marquee technique only and extends the Marquee rectangle around any items under the Marquee start and end points.

XmTOUCH_ONLY
Uses the Random-Swipe technique only if **XmNlayoutType** is **XmSPATIAL**. Otherwise, uses the Range-Swipe and Range-Click techniques.

XmTOUCH_OVER
If the selection action begins over an item and **XmNlayoutType** is **XmSPATIAL**, uses the Random-Swipe technique. If the selection action begins over an item and **XmNlayoutType** is **XmOUTLINE** or **XmDETAIL**, uses the Range-Swipe and Range-Click techniques. Uses the Marquee technique if the select action begins over an unoccupied area in the Container.

XmNsmallCellHeight
Specifies the height of a cell for **XmGRID** or **XmCELLS** spatial style when **XmNentryViewType** is **XmSMALL_ICON**.

XmNsmallCellWidth
Specifies the width of a cell for **XmGRID** or **XmCELLS** spatial style when **XmNentryViewType** is **XmSMALL_ICON**.

XmNspatialIncludeModel
Specifies the layout of an item when the item is managed in the Container when **XmNlayoutType** is **XmSPATIAL** and **XmNspatialStyle** is **XmGRID** or **XmCELLS**. It can have one of the following values:

XmAPPEND
Places the item after the last occupied cell according to **XmNlayoutDirection**.

XmCLOSEST
Places the item in the free cell closest to the position specified by **XmNx** and **XmNy**.

XmFIRST_FIT
Places the item in the first free cell according to **XmNlayoutDirection**.

XmNspatialResizeModel
Specifies how Container will attempt to grow its dimensions when **XmNlayoutType** is **XmSPATIAL** and **XmNspatialStyle** is **XmGRID** or **XmCELLS** and there are not enough cells to contain a new Container item. It can have one of the following values:

XmGROW_BALANCED
Container will request both width and height growth from its parent.

XmGROW_MAJOR
Container will request growth in its major dimension from its parent. Container's major dimension is width when the precedence of **XmNlayoutDirection** is horizontal, and height when vertical.

XmGROW_MINOR
Container will request growth in its minor dimension from its parent. Container's minor dimension is height when the precedence of **XmNlayoutDirection** is horizontal, and width when vertical.

XmNspatialSnapModel
Specifies how Container will position an item within the cell layout when **XmNlayoutType** is **XmSPATIAL** and **XmNspatialStyle** is **XmGRID** or **XmCELLS**. It can have one of the following values:

XmCENTER
Center the items as follows, depending on the value of **XmNentryViewType**:

XmLARGE_ICON
The child is centered in the cell horizontally and baseline-aligned vertically.

XmSMALL_ICON
The child is centered in the cell vertically on its baseline and aligned with the left or right of the cell horizontally, depending on the value of **XmNlayoutDirection**.

XmSNAP_TO_GRID
Position the item at the upper-left or upper-right corner of the cell(s), depending on the value of **XmNlayoutDirection**.

XmNONE Position the item according to the position specified by **XmNx** and **XmNy**. If the position is not within the coordinates of the cell(s), then position the item at the upper-left or upper-right corner of the cell(s), depending on the value of **XmNlayoutDirection**.

XmNspatialStyle
Specifies the layout of Container items when **XmNlayoutType** is **XmSPATIAL**. It can have one of the following values:

XmCELLS Lays out items within a grid of same-size cells. Each item occupies as many cells as required to contain the item dimensions.

XmGRID Lays out items within a grid of same-size cells. Each item occupies only one cell. Items that are larger than the cell size may overlap other items.

XmNONE Lays out items according to **XmNx** and **XmNy**.

XmContainer Constraint Resource Set

Name	**Class**	**Type**	**Default**	**Access**
XmNentryParent	XmCWidget	Widget	NULL	CSG
XmNoutlineState	XmCOutlineState	unsigned char	XmCOLLAPSED	CSG
XmNpositionIndex	XmCPositionIndex	int	dynamic	CSG

XmNentryParent
Specifies the widget that is this item's logical parent. A value of NULL indicates that this is a root-level item. Parent-child information is displayed only when the *XmNlayoutPolicy* is **XmOUTLINE** or **XmDETAIL**.

XmNoutlineState
Specifies whether to display child items when *XmNlayoutPolicy* is **XmOUTLINE** or **XmDETAIL**. It can have one of the following values:

XmCOLLAPSED
Does not display child items.

XmEXPANDED
Displays child items.

XmNpositionIndex
Specifies the order of items in the Container for display. When **XmNlayoutType** is **XmOUTLINE** or **XmDETAIL**, items are displayed in **XmNpositionIndex** order within **XmNentryParent**. Items that have an **XmNentryParent** resource are ignored when **XmNlayoutType** is **XmSPATIAL**. If **XmNpositionIndex** is not specified, it defaults to the **XmNpositionIndex** value plus 1 of the item with the highest **XmNpositionIndex** that has the same **XmNentryParent** if such an item exists; otherwise, it defaults to 0.

Inherited Resources

Container 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	Type	Default	Access
XmNbottomShadow- Color	XmCBottomShadowColor	Pixel	dynamic	CSG
XmNbottomShadow- Pixmap	XmCBottomShadow- Pixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler- Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG

XmNstringDirection	XmCStringDirection	XmString-Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C

XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback Information

A pointer to the following structure is passed to callbacks for **XmNoutlineChangedCallback**.

typedef struct
{
    int *reason*;
    XEvent * *event*;
    Widget *item*;
    unsigned char *new_outline_state*;
} XmContainerOutlineCallbackStruct;

reason Specifies the reason for the callback.

event Points to the *XEvent* that triggered the callback. It can be NULL.

item Specifies the container item affected by the event.

new_outline_state
Specifies the next **XmNoutlineState** for *item*. The user can change this value in the callback.

A pointer to the following structure is passed to callbacks for **XmNdefaultActionCallback** and **XmNselectionCallback**.

typedef struct
{
    int *reason*;
    XEvent * *event*;
    WidgetList *selected_items*;
    int *selected_item_count*;

```
        unsigned char auto_selection_type;
} XmContainerSelectCallbackStruct;
```

reason Specifies the reason for the callback. It corresponds to the **XmNselectionPolicy** at the time the selection was made, or indicates that the default action should be taken.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selected_items
Specifies a list of items selected at the time of the *event* that caused the callback. The *selected_items* field 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
Specifies the number of items in the *selected_items* list. This number must be positive or 0 (zero).

auto_selection_type
Indicates the cause of the selection when **XmNautomaticSelection** is **XmAUTO_SELECT**. Valid values are the following:

XmAUTO_UNSET
Returned when **XmNautomaticSelection** is **XmNO_AUTO_SELECT**.

XmAUTO_BEGIN
Indicates the beginning of automatic selection.

XmAUTO_MOTION
Indicates that there is a button drag selection.

XmAUTO_CANCEL
Indicates that the new selection is canceled.

XmAUTO_NO_CHANGE
Indicates that the currently selected item matches the initial item.

XmAUTO_CHANGE
Indicates that the currently selected item does not match the initial item.

A pointer to the following structure is passed to the **XmNconvertCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent  *event;
        Atom selection;
        Atom target;
        XtPointer source_data;
        XtPointer location_data;
        int flags;
        XtPointer parm;
        int parm_format;
        unsigned long parm_length;
        int status;
        XtPointer value;
        Atom type;
        int format;
        unsigned long length;
} XmConvertCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which conversion is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

target Indicates the conversion target.

source_data Contains information about the selection source. When the selection is _MOTIF_DROP, *source_data* is the DragContext. Otherwise, it is NULL.

location_data
Contains information about the location of data to be converted. If the value is NULL, the data to be transferred consists of the widget's current selection. Otherwise, it is the widget ID of the item being transferred, or the widget ID of the Container if all items are being transferred.

flags Indicates the status of the conversion. Following are the possible values:

XmCONVERTING_NONE
This flag is currently unused.

XmCONVERTING_PARTIAL
The target widget was able to be converted, but some data was lost.

XmCONVERTING_SAME
The conversion target is the source of the data to be transferred.

XmCONVERTING_TRANSACT
This flag is currently unused.

parm Contains parameter data for this target. If no parameter data exists, the value is NULL.

When *selection* is *CLIPBOARD* and *target* is _MOTIF_CLIPBOARD_TARGETS or _MOTIF_DEFERRED_CLIPBOARD_TARGETS, the value is the requested operation (**XmCOPY**, **XmMOVE**, or **XmLINK**).

parm_format
Specifies whether the data in *parm* should be viewed as a list of *char*, *short*, or *long* quantities. Possible values are 0 (when *parm* is NULL), 8 (when the data in *parm* should be viewed as a list of *char*s), 16 (when the data in *parm* should be viewed as a list of *short*s), or 32 (when the data in *parm* should be viewed as a list of *long*s). Note that *parm_format* symbolizes a data type, not the number of bits in each list element. For example, on some machines, a *parm_format* of 32 means that the data in *parm* should be viewed as a list of 64-bit quantities, not 32-bit quantities.

parm_length Specifies the number of elements of data in *parm*, where each element has the size specified by *parm_format*. When *parm* is NULL, the value is 0.

status An IN/OUT member that specifies the status of the conversion. The initial value is **XmCONVERT_DEFAULT**. The callback procedure can set this member to one of the following values:

XmCONVERT_DEFAULT
The widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it overwrites the data provided by the callback procedures in the *value* member.

XmCONVERT_MERGE
The widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it appends its data to the data provided by the callback procedures in the *value* member. This value is intended for use with targets that result in lists of data, such as *TARGETS*.

XmCONVERT_DONE
The callback procedure has successfully finished the conversion. The widget class conversion procedure, if any, is not called after the callback procedures return.

XmCONVERT_REFUSE
The callback procedure has terminated the conversion process without completing the requested conversion. The widget class conversion procedure, if any, is not called after the callback procedures return.

value An IN/OUT parameter that contains any data that the callback procedure produces as a result of the conversion. The initial value is NULL. If the callback procedure sets this member, it must ensure that the *type*, *format*, and *length* members correspond to the data in *value*. The callback procedure is responsible for allocating, but not for freeing, memory when it sets this member.

type An IN/OUT parameter that indicates the type of the data in the *value* member. The initial value is *INTEGER*.

format An IN/OUT parameter that specifies whether the data in *value* should be viewed as a list of *char*, *short*, or *long* quantities. The initial value is 8. The callback procedure can set this member to 8 (for a list of *char*), 16 (for a list of *short*), or 32 (for a list of *long*).

length An IN/OUT member that specifies the number of elements of data in *value*, where each element has the size symbolized by *format*. The initial value is 0 (zero).

A pointer to the following callback structure is passed to the **XmNdestinationCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent  *event;
        Atom selection;
        XtEnum operation;
        int flags;
        XtPointer transfer_id;
        XtPointer destination_data;
        XtPointer location_data;
        Time time;
} XmDestinationCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which data transfer is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

operation Indicates the type of transfer operation requested.

- When the selection is *PRIMARY*, possible values are **XmMOVE**, **XmCOPY**, and **XmLINK**.
- When the selection is *SECONDARY* or *CLIPBOARD*, possible values are **XmCOPY** and **XmLINK**.
- When the selection is _MOTIF_DROP, possible values are **XmMOVE**, **XmCOPY**, **XmLINK**, and **XmOTHER**. A value of **XmOTHER** means that the callback procedure must get further information from the **XmDropProcCallbackStruct** in the *destination_data* member.

flags Indicates whether or not the destination widget is also the source of the data to be transferred. Following are the possible values:

XmCONVERTING_NONE
The destination widget is not the source of the data to be transferred.

XmCONVERTING_SAME
The destination widget is the source of the data to be transferred.

transfer_id
Serves as a unique ID to identify the transfer transaction.

destination_data
Contains information about the destination. When the selection is _MOTIF_DROP, the callback procedures are called by the drop site's **XmNdropProc**, and *destination_data* is a pointer to the **XmDropProcCallbackStruct** passed to the **XmNdropProc** procedure. When the selection is *SECONDARY*, *destination_data* is an Atom representing a target recommmended by the selection owner for use in converting the selection. Otherwise, *destination_data* is NULL.

location_data
Contains information about the location where data is to be transferred. The value is always NULL when the selection is *SECONDARY* or *CLIPBOARD*. If the value is NULL, the data is to be inserted at the widget's cursor position. Otherwise, the value is a pointer to an *XPoint* structure containing the x and y coordinates at the location where the data is to be transferred. Once *XmTransferDone* procedures start to be called, **location_data** will no longer be stable.

time Indicates the time when the transfer operation began.

Translations

The **XmContainer** translations are listed below.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

≈c ≈s ≈m ≈a <Btn1Down>:
ContainerBeginSelect()

<Btn1Motion>:
ContainerButtonMotion()

≈c ≈s ≈m ≈a <Btn1Up>:
ContainerEndSelect()

c ≈s ≈m ≈a <Btn1Down>:
ContainerBeginToggle()

c ≈s ≈m ≈a <Btn1Up>:
: **ContainerEndToggle()**

≈c s ≈m ≈a <Btn1Down>:
: **ContainerBeginExtend()**

≈c s ≈m ≈a <Btn1Up>:
: **ContainerEndExtend()**

c s ≈m ≈a <Btn1Down>:
: **ContainerBeginExtend()**

c s ≈m ≈a <Btn1Up>:
: **ContainerEndExtend()**

≈c ≈s ≈m ≈a <Btn2Down>:
: **ContainerStartTransfer(***Copy***)**

c s ≈m ≈a <Btn2Down>:
: **ContainerStartTransfer(***Link***)**

≈c s ≈m ≈a <Btn2Down>:
: **ContainerStartTransfer(***Move***)**

≈m ≈a <Btn2Up>:
: **ContainerEndTransfer()**

:c s a <Key>osfInsert:
: **ContainerPrimaryLink()**

:c s m <Key>osfInsert:
: **ContainerPrimaryLink()**

:a <Key>osfInsert:
: **ContainerPrimaryCopy()**

:m <Key>osfInsert:
: **ContainerPrimaryCopy()**

:s a <Key>osfDelete:
: **ContainerPrimaryMove()**

:s m <Key>osfDelete:
: **ContainerPrimaryMove()**

:<Key>osfCancel:
: **ContainerCancel()**

:s <Key>osfSelect:
ContainerExtend()

:<Key>osfSelect:
ContainerSelect()

:<Key>osfSelectAll:
ContainerSelectAll()

:<Key>osfDeselectAll:
ContainerDeselectAll()

:<Key>osfAddMode:
ContainerToggleMode()

:<Key>osfActivate:
ContainerActivate()

s ≈c ≈m ≈a <Key>space:
ContainerExtend()

≈s ≈c ≈m ≈a <Key>space:
ContainerSelect()

≈s ≈c ≈m ≈a <Key>Return:
ContainerActivate()

≈s c ≈m ≈a <Key>slash:
ContainerSelectAll()

≈s c ≈m ≈a <Key>backslash:
ContainerDeselectAll()

:c s <Key>osfBeginLine:
ContainerExtendCursor(*First***)**

:c s <Key>osfEndLine:
ContainerExtendCursor(*Last***)**

:c <Key>osfBeginLine:
ContainerMoveCursor(*First***)**

:c <Key>osfEndLine:
ContainerMoveCursor(*Last***)**

:c <Key>osfLeft:
ContainerExpandOrCollapse(*Left***)**

:c <Key>osfRight:
  ContainerExpandOrCollapse(*Right***)**

:s <Key>osfUp:
  ContainerExtendCursor(*Up***)**

:s <Key>osfDown:
  ContainerExtendCursor(*Down***)**

:s <Key>osfLeft:
  ContainerExtendCursor(*Left***)**

:s <Key>osfRight:
  ContainerExtendCursor(*Right***)**

:<Key>osfUp:
  ContainerMoveCursor(*Up***)**

:<Key>osfDown:
  ContainerMoveCursor(*Down***)**

:<Key>osfLeft:
  ContainerMoveCursor(*Left***)**

:<Key>osfRight:
  ContainerMoveCursor(*Right***)**

s ≈m ≈a <Key>Tab:
  ManagerGadgetPrevTabGroup()

≈s ≈m ≈a <Key>Tab:
  ManagerGadgetNextTabGroup()

The Container button event translations are modified when Display's **XmNenableBtn1Transfer** resource does not have a value of **XmOFF** (in other words, it is either *XmBUTTON2_TRANSFER* or *XmBUTTON2_ADJUST*). This option allows the actions for selection and transfer to be integrated on Btn1, and the actions for extending the selection can be bound to Btn2. The actions for Btn1 that are defined in the preceding list still apply when the Btn1 event occurs over text that is not selected. The following actions apply when the Btn1 event occurs over text that is selected:

~c ~s ~m ~a <Btn1Down:
  ContainerHandleBtn1Down(*ContainerBeginSelect,Copy***)**

c ~s ~m ~a <Btn1Down>:
  ContainerHandleBtn1Down(*ContainerBeginToggle,Copy***)**

c s ~m ~a <Btn1Down>:
  ContainerHandleBtn1Down(*ContainerNoop,Link***)**

~c s ~m ~a <Btn1Down>:
  ContainerHandleBtn1Down(*ContainerBeginExtend,Move***)**

<Btn1Motion>:
  ContainerHandleBtn1Motion(*ContainerButtonMotion***)**

~c ~s ~m ~a <Btn1Up>:
  ContainerHandleBtn1Up(*ContainerEndSelect***)**

c ~s ~m ~a <Btn1Up>:
  ContainerHandleBtn1Up(*ContainerEndToggle***)**

~c s ~m ~a <Btn1Up>:
  ContainerHandleBtn1Up(*ContainerEndExtend***)**

c s ~m ~a <Btn1Down>:
  ContainerHandleBtn1Down(*ContainerBeginExtend***)**

c s ~m ~a <Btn1Up>:
  ContainerHandleBtn1Up(*ContainerEndExtend***)**

When Display's **XmNenableBtn1Transfer** resource has a value of *XmBUTTON2_ADJUST*, the following systemitem class="Constant"s apply:

~c ~s ~m ~a <Btn2Down>:
  ContainerHandleBtn2Down(*ContainerStartTransfer,Copy***)**

c s ~m ~a <Btn2Down>:
  ContainerHandleBtn2Down(*ContainerStartTransfer,Link***)**

~c s ~m ~a <Btn2Down>:
  ContainerHandleBtn2Down(*ContainerStartTransfer,Move***)**

<Btn2Motion>:
  ContainerHandleBtn2Motion(*ContainerButtonMotion***)**

~m ~a <Btn2Up>:
  ContainerHandleBtn2Up(*ContainerEndTransfer***)**

Action Routines

The Container action routines are described below. The current selections are always shown with the background color specified by the **XmNselectColor** resource.

ContainerActivate():
: This action calls **XmNdefaultActionCallback** with reason **XmCR_DEFAULT_ACTION**.

ContainerBeginExtend():
: Simply returns if **XmNselectionPolicy** is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**. Simply returns if **XmNlayoutType** is **XmSPATIAL**.

: Otherwise, this action sets the selection state of all items between the anchor item and the item under the pointer to the selection state of the anchor item. The location cursor is moved to the item under the pointer. If **XmNautomaticSelection** is **XmAUTO_SELECT**, the **XmNselectionCallback**(s) is called with either **XmCR_MULTIPLE_SELECT** or **XmCR_EXTENDED_SELECT** as the reason depending on **XmNselectionPolicy**, and with **auto_selection_type XmAUTO_CHANGE**.

ContainerBeginSelect():
: If this is a second ContainerBeginSelect() action that has occurred within the time specified by the display's multiclick time, this action calls **XmNdefaultActionCallback** with reason **XmCR_DEFAULT_ACTION** and returns.

: Otherwise, processing depends on the value of **XmNselectionPolicy** as follows:

: **XmSINGLE_SELECT**
: This action deselects all items and toggles the item (if any) under the pointer.

: **XmBROWSE_SELECT**
: This action deselects all items and toggles the item (if any) under the pointer. This item is now the anchor item for further selection. If **XmNautomaticSelection** is **XmAUTO_SELECT** and a change in any item's selection state is made, the **XmNselectionCallback**(s) is called with reason **XmCR_BROWSE_SELECT** and **auto_selection_type XmAUTO_BEGIN**.

XmMULTIPLE_SELECT

If the pointer is over an item and **XmNselectionTechnique** is not **XmMARQUEE**, this action toggles the selection state of that item. The item becomes the anchor item for further selection. If **XmNselectionTechnique** is **XmMARQUEE**, **XmMARQUEE_EXTEND_START**, or **XmMARQUEE_EXTEND_BOTH**, this action sets the start point for the Marquee rectangle. If **XmNselectionTechnique** is **XmMARQUEE_EXTEND_START** or **XmMARQUEE_EXTEND_BOTH** and the pointer is over an item, this action draws the Marquee rectangle around the item. If **XmNautomaticSelection** is **XmAUTO_SELECT**, the **XmNselectionCallback**(s) is called with reason **XmCR_MULTIPLE_SELECT** and **auto_selection_type XmAUTO_BEGIN**.

XmEXTENDED_SELECT

All items are first deselected. Processing is then identical to the case where **XmNselectionPolicy** is **XmMULTIPLE_SELECT**, except that **XmCR_EXTENDED_SELECT** is the callback reason given if **XmNselectionCallback** is called.

ContainerBeginToggle():

Simply returns if **XmNselectionPolicy** is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**.

Otherwise, if the pointer is over an item and **XmNselectionTechnique** is not **XmMARQUEE**, this action toggles the selection state of that item. The item becomes the anchor item for further selection. If **XmNselectionTechnique** is **XmMARQUEE**, **XmMARQUEE_EXTEND_START**, or **XmMARQUEE_EXTEND_BOTH** this action sets the start point for the Marquee rectangle. If **XmNselectionTechnique** is **XmMARQUEE_EXTEND_START** or **XmMARQUEE_EXTEND_BOTH** and the pointer is over an item, this action draws the Marquee rectangle around the item. If **XmNautomaticSelection** is **XmAUTO_SELECT**, the **XmNselectionCallback**(s) is called with either

XmCR_MULTIPLE_SELECT or **XmCR_EXTENDED_SELECT** as the reason, depending on **XmNselectionPolicy**, and with **auto_selection_type XmAUTO_BEGIN**.

ContainerButtonMotion():

Processing depends on the value of **XmNselectionPolicy**, as follows:

XmSINGLE_SELECT

This action simply returns to the caller.

XmBROWSE_SELECT

Simply returns if this action follows a ContainerBeginExtend() action or ContainerBeginToggle() action.

If the pointer is no longer over the current anchor item, this action toggles the current anchor item and then toggles the item under the pointer (if any) and makes it the new anchor item for further processing. If **XmNautomaticSelection** is **XmAUTO_SELECT** and a change in any item's selection state is made, the **XmNselectionCallback**(s) is called with reason **XmCR_BROWSE_SELECT** and **auto_selection_type XmAUTO_MOTION**.

XmMULTIPLE_SELECT

If a previous action has set a Marquee rectangle start point, this action draws the Marquee rectangle between the current pointer position and the Marquee start point. If the **XmNselectionTechnique** is **XmMARQUEE_EXTEND_BOTH** and the pointer is over an item, the end point of the Marquee rectangle is extended to include the item. The selection states of all items within the Marquee rectangle are toggled to match the state of the anchor item.

If no Marquee rectangle start point is set and the pointer is over an item, processing depends on the **XmNlayoutType** resource. The anchor item from the previous action is used. If **XmNlayoutType** is **XmSPATIAL**, the selection state of the item under the pointer is toggled to match the selection state of the anchor item. If **XmNlayoutType** is **XmOUTLINE** or **XmDETAIL**, the selection state of all items between the anchor item and the item under the pointer are toggled to match the selection state of the anchor item.

If **XmNautomaticSelection** is **XmAUTO_SELECT** and a change in any item's selection state is made, the **XmNselectionCallback**(s) is called with reason **XmCR_MULTIPLE_SELECT** and **auto_selection_type XmAUTO_MOTION**.

XmEXTENDED_SELECT
Processing is identical to the case where **XmNselectionPolicy** is **XmMULTIPLE_SELECT**, except that **XmCR_EXTENDED_SELECT** is the callback reason given if **XmNselectionCallback** is called.

ContainerCancel():
If a selection is in progress, this action restores selection states of all items to their state before the selection began. If **XmNautomaticSelection** is True and a change in any item's selection state is made, the **XmNselectionCallback** is called with reason **XmCR_BROWSE_SELECT**, **XmMULTIPLE_SELECT**, or **XmCR_EXTENDED_SELECT** depending on the **XmNselectionPolicy** resource and **auto_selection_type XmAUTO_CANCEL**.

ContainerDeselectAll():
This action deselects all items and calls **XmNselectionCallback** with reason depending on **XmNselectionPolicy**.

ContainerEndExtend():
Simply returns if **XmNselectionPolicy** is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**. Simply returns if **XmNlayoutType** is **XmSPATIAL**.

Otherwise, if **XmNautomaticSelection** is **XmNO_AUTO_SELECT**, **XmNselectionCallback**(s) is called with either **XmCR_MULTIPLE_SELECT** or **XmCR_EXTENDED_SELECT** as the reason depending on **XmNselectionPolicy**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and no change is made in any item's selection state by this action, **XmNselectionCallback**(s) is called with either **XmCR_MULTIPLE_SELECT** or **XmCR_EXTENDED_SELECT** as the reason depending on **XmNselectionPolicy** and **auto_selection_type** **XmAUTO_CHANGE**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and this action makes no change in any item’s selection state, **XmNselectionCallback**(s) is called with either **XmCR_MULTIPLE_SELECT** or **XmCR_EXTENDED_SELECT** as the reason depending on **XmNselectionPolicy** and **auto_selection_type** **XmAUTO_NO_CHANGE**.

ContainerEndSelect():
Processing depends on the value of **XmNselectionPolicy**, as follows:

XmSINGLE_SELECT
This action calls **XmNselectionCallback** with reason **XmCR_SINGLE_SELECT**.

XmBROWSE_SELECT
If the pointer is no longer over the current anchor item, this action toggles the current anchor item and then toggles the item under the pointer (if any). If **XmNautomaticSelection** is **XmNO_AUTO_SELECT**, the **XmNselectionCallback**(s) is called with reason **XmCR_BROWSE_SELECT**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and a change in any item’s selection state is made, **XmNselectionCallback**(s) is called with reason **XmCR_BROWSE_SELECT** and **auto_selection_type** **XmAUTO_CHANGE**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and no change is made in any item’s selection state by this action, **XmNselectionCallback**(s) is called with reason **XmCR_BROWSE_SELECT** and **auto_selection_type** **XmAUTO_NO_CHANGE**.

XmMULTIPLE_SELECT
This action first performs the same processing as the ContainerButtonMotion() action, except that **XmNselectionCallback** is not called. If **XmNautomaticSelection** is **XmNO_AUTO_SELECT**, the **XmNselectionCallback**(s) is called with reason **XmCR_MULTIPLE_SELECT**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and a change in any item’s selection state is made, the **XmNselectionCallback**(s) is called

with reason **XmCR_MULTIPLE_SELECT** and **auto_selection_type XmAUTO_CHANGE**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and this action makes no change in any item's selection state, **XmNselectionCallback**(s) is called with reason **XmCR_MULTIPLE_SELECT** and **auto_selection_type XmAUTO_NO_CHANGE**.

XmEXTENDED_SELECT

This action first performs the same processing as the ContainerButtonMotion() action, except that **XmNselectionCallback** is not called. If **XmNautomaticSelection** is **XmNO_AUTO_SELECT**, the **XmNselectionCallback**(s) is called with reason **XmCR_EXTENDED_SELECT**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and a change in any item's selection state is made, **XmNselectionCallback**(s) is called with reason **XmCR_EXTENDED_SELECT** and **auto_selection_type XmAUTO_CHANGE**. If **XmNautomaticSelection** is **XmAUTO_SELECT** and this action makes no change in any item's selection state, **XmNselectionCallback**(s) is called with reason **XmCR_EXTENDED_SELECT** and **auto_selection_type XmAUTO_NO_CHANGE**.

ContainerEndToggle():

Simply returns if **XmNselectionPolicy** is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**. If **XmNselectionPolicy** is **XmMULTIPLE_SELECT** or **XmEXTENDED_SELECT**, this action performs the same processing as the ContainerEndSelect() action.

ContainerEndTransfer()::

If the elapsed time since a ContainerStartTransfer() action has occurred exceeds the time span specified by the display's multiclick time, this action returns.

Otherwise, the ContainerPrimaryCopy(), ContainerPrimaryLink(), or ContainerPrimaryMove() action is invoked, depending on the value of the operation parameter saved by ContainerStartTransfer().

ContainerExpandOrCollapse(*Left*|*Right*|*Collapse*|*Expand*):

This action changes the value of the **XmNoutlineState** of the current focus widget. If the argument value is **Collapse** or **Left**, the **XmNoutlineState** resource value is set to **XmCOLLAPSED**. If the argument value is **Expand** or **Right**, the **XmNoutlineState** resource value is set to **XmEXPANDED**.

If the argument is **Left** or **Right** and the layout is right to left, then the setting of the **XmNoutlineState** value is reversed from that described in the preceding paragraph.

Simply returns if **XmNlayoutType** is **XmSPATIAL**.

ContainerExtend():

Processing depends on the value of **XmNselectionPolicy**, as follows:

If the selection policy is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**, this action returns. If **XmNlayoutType** is **XmSPATIAL**, this action returns.

If the selection policy is **XmMULTIPLE_SELECT**, this action sets the selection state of all items between the anchor item and the location cursor to the selection state of the anchor item.

If the selection policy is **XmEXTENDED_SELECT** and the Container is in Normal Mode, this action deselects all items and selects all items between the anchor item and the location cursor. If the selection policy is **XmEXTENDED_SELECT** and the Container is in Add Mode, this action sets the selection state of all items between the anchor item and the location cursor to the selection state of the anchor item.

XmNselectionCallback is called with reason **XmCR_MULTIPLE_SELECT** or **XmCR_EXTENDED_SELECT** depending on **XmNselectionPolicy**.

ContainerExtendCursor(*Left*|*Right*|*Up*|*Down*):

Processing depends on the value of **XmNselectionPolicy**, as follows:

If the selection policy is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**, this action returns. If **XmNlayoutType** is **XmSPATIAL**, this action returns.

This action moves the location cursor one item in the indicated direction, if possible. If the value of the argument string is **First** or **Last**, this action moves the location cursor to the indicated item. For other values of the argument string, the location cursor is not affected.

If the selection policy is **XmMULTIPLE_SELECT**, this action sets the selection state of all items between the anchor item and the location cursor to the selection state of the anchor item.

If the selection policy is **XmEXTENDED_SELECT** and the Container is in Normal Mode, this action deselects all items and selects all items between the anchor item and the location cursor. If the selection policy is **XmEXTENDED_SELECT** and the Container is in Add Mode, this action sets the selection state of all items between the anchor item and the location cursor to the selection state of the anchor item.

XmNselectionCallback is called with reason **XmCR_MULTIPLE_SELECT** or **XmCR_EXTENDED_SELECT** depending on **XmNselectionPolicy**.

ContainerHandleBtn1Down(*string*)
When Display's **XmNenableBtn1Transfer** resource is not **XmOFF**, the actions for selection and transfer are integrated on Btn1. If the pointer is over an unselected item or background, the item is first selected before the transfer is started. Otherwise, if the item is already selected, the transfer is started. The value of *string* can be one of the following actions:

- **ContainerBeginSelect,Copy**
- **ContainerBeginToggle,Copy**
- **ContainerNoop,Link**
- **ContainerBeginExtend,Move**

ContainerHandleBtn1Motion(*string*)
When Display's **XmNenableBtn1Transfer** resource is not **XmOFF**, the actions for selection and transfer are integrated on Btn1. When this action is invoked, and a selection is in progress, a drag is performed. Otherwise, the default action as specified in *string* is performed. The value of *string* can be **ContainerButtonMotion**.

ContainerHandleBtn1Up(*string*)
If a Button 1 transfer was in progress, then when this action is invoked, that transfer is cancelled. Otherwise, the default action as specified in

string is performed. The value of *string* can be one of the following actions:

- **ContainerEndSelect**
- **ContainerEndToggle**
- **ContainerEndExtend**

ContainerHandleBtn2Down(*string*)
When Display's **XmNenableBtn1Transfer** resource has a value of *XmBUTTON2_ADJUST*, the actions for extending selection are bound on Btn2. Otherwise, the action that is performed depends on the value of *string*, which can be one of the following actions:

- **ContainerStartTransfer,Copy**
- **ContainerStartTransfer,Link**
- **ContainerStartTransfer,Move**

ContainerHandleBtn2Motion(*string*)
When Display's **XmNenableBtn1Transfer** resource is not *XmBUTTON2_ADJUST*, and a selection is in progress, a drag is performed. Otherwise, the default action that is performed depends on the value of *string*, which can be **ContainerButtonMotion**.

ContainerHandleBtn2Up(*string*)
When Display's **XmNenableBtn1Transfer** resource has a value of *XmBUTTON2_ADJUST*, this action ends an extend. Otherwise, the action that is performed depends on the value of *string*, which can be **ContainerEndTransfer**.

ContainerMoveCursor(*Left*|*Right*|*Up*|*Down*|*First*|*Last*):
If the argument is **Left**, **Right**, **Up**, or **Down**, this action moves the location cursor one item in the indicated direction, if possible. If the value of the argument string is **First** or **Last**, this action moves the location cursor to the indicated item. Any other arguments are ignored.

If **XmNselectionPolicy** is **XmBROWSE_SELECT**, or if **XmNselectionPolicy** is **XmEXTENDED_SELECT** and the Container is in Normal Mode, this action deselects all items, selects the item at the location cursor, and calls **XmNselectionCallback** with the reason depending on **XmNselectionPolicy**.

ContainerPrimaryCopy():

This action requests that primary selection data be copied to the Container. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmCOPY** operation. By default, the Container does not do any transfer, and copying the selection is the responsibility of the **XmNdestinationCallback** procedures.

ContainerPrimaryLink():

This action requests that primary selection data be linked to the Container. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmLINK** operation. By default, the Container does not do any transfer, and linking the selection is the responsibility of the **XmNdestinationCallback** procedures.

ContainerPrimaryMove():

This action requests that primary selection data be copied to the Container and deleted from the primary source. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmMOVE** operation. By default, the Container does not do any transfer, and moving the selection is the responsibility of the **XmNdestinationCallback** procedures. If the transfer is successful, this action then calls the selection owner's **XmNconvertCallback** procedures for the *PRIMARY* selection and the *DELETE* target.

ContainerSelect():

Processing depends on the value of **XmNselectionPolicy**, as follows:

If the selection policy is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**, this action deselects all items and selects the item at the location cursor.

If the selection policy is **XmMULTIPLE_SELECT**, this action toggles the selection state of the item at the location cursor. This item becomes the anchor item for further selections.

If the selection policy is **XmEXTENDED_SELECT** and the Container is in Normal Mode, this action deselects all items and selects the item at the location cursor. If the selection policy is **XmEXTENDED_SELECT** and the Container is in Add Mode, this action toggles the selection state of the item at the location cursor. The selected/toggled item becomes the anchor item for further selections.

XmNselectionCallback is called with the reason depending on **XmNselectionPolicy**.

ContainerSelectAll():

If **XmNselectionPolicy** is **XmSINGLE_SELECT** or **XmBROWSE_SELECT**, this action deselects all items and selects the item at the location cursor position.

If **XmNselectionPolicy** is **XmMULTIPLE_SELECT** or **XmEXTENDED_SELECT**, this action selects all items.

XmNselectionCallback is called with the reason depending on the value of **XmNselectionCallback**.

ContainerStartTransfer(*Copy*|*Move*|*Link*):

This action saves the event and the operation specified in the argument string for use by subsequent actions. If no ContainerEndTransfer() actions occur within the time span specified by the display's multiclick time and if **XmNlayoutType** is **XmSPATIAL**, this action creates a DragContext and starts a drag transfer by using *string* to specify the transfer operation. If no argument string is specified, **Copy** is the default value.

Unless default drag and drop behavior has been overridden by a **XmNconvertCallback** procedure, if the drop operation occurs within the Container, then the item(s) being dragged are relocated at the position of the drop operation. If the item targeted by the Drag operation is not in the selected state, then only that item is moved. If the item is in the selected state, however, all items in the selected state are moved.

ContainerToggleMode():

If **XmNselectionPolicy** is **XmEXTENDED_SELECT**, this action toggles the Container between Normal Mode and Add Mode.

Additional Behavior

The Container widget has the following additional behavior:

Btn1Down**(2+)**

If a button click is followed by another button click within the time span specified by the display's multiclick time, the Container interprets that as a double-click and calls the **XmNdefaultActionCallback** callbacks.

FocusIn: If the focus policy is explicit, sets the focus and draws the location cursor.

FocusOut: If the focus policy is explicit, removes the focus and erases the location cursor.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmContainerCopy**, **XmContainerCopyLink**, **XmContainerCut**, **XmContainerGetItemChildren**, **XmContainerPaste**, **XmContainerPasteLink**, **XmContainerRelayout**(3), **XmContainerReorder**(3), **XmCreateContainer**(3), **XmCreateIconGadget**(3), **XmIconGadget**(3), and **XmManager**(3).

XmDialogShell

Purpose

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.

DialogShell uses the *XmQTdialogShellSavvy* trait.

Classes

DialogShell inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNtransientFor	XmCTransientFor	Widget	NULL	CSG

VendorShell Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaudibleWarning	XmCAudibleWarning	unsigned char	XmBELL	CSG

XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNdeleteResponse	XmCDeleteResponse	unsigned char	XmUNMAP	CSG
XmNinputMethod	XmCInputMethod	String	NULL	CSG
XmNinputPolicy	XmCInputPolicy	XmInputPolicy	XmPER_- SHELL	CSG
XmNkeyboardFocusPolicy	XmCKeyboardFocusPolicy	unsigned char	XmEXPLICIT	CSG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTabel	XmRenderTable	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	XmLEFT_TO_- RIGHT	CG
XmNmwmDecorations	XmCMwmDecorations	int	-1	CG
XmNmwmFunctions	XmCMwmFunctions	int	-1	CG
XmNmwmInputMode	XmCMwmInputMode	int	-1	CG
XmNmwmMenu	XmCMwmMenu	String	NULL	CG
XmNpreeditType	XmCPreeditType	String	dynamic	CSG
XmNshellUnitType	XmCShellUnitType	unsigned char	XmPIXELS	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNunitType	XmCUnitType	unsigned char	XmPIXELS	CSG
XmNuseAsyncGeometry	XmCUseAsyncGeometry	Boolean	False	CSG

WMShell Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbaseHeight	XmCBaseHeight	int	XtUnspecifiedShellInt	CSG
XmNbaseWidth	XmCBaseWidth	int	XtUnspecifiedShellInt	CSG
XmNheightInc	XmCHeightInc	int	XtUnspecifiedShellInt	CSG
XmNiconMask	XmCIconMask	Pixmap	NULL	CSG
XmNiconPixmap	XmCIconPixmap	Pixmap	NULL	CSG
XmNiconWindow	XmCIconWindow	Window	NULL	CSG
XmNiconX	XmCIconX	int	XtUnspecifiedShellInt	CSG
XmNiconY	XmCIconY	int	XtUnspecifiedShellInt	CSG

XmNinitialState	XmCInitialState	int	NormalState	CSG
XmNinput	XmCInput	Boolean	True	CSG
XmNmaxAspectX	XmCMaxAspectX	int	XtUnspecifiedShellInt	CSG
XmNmaxAspectY	XmCMaxAspectY	int	XtUnspecifiedShellInt	CSG
XmNmaxHeight	XmCMaxHeight	int	XtUnspecifiedShellInt	CSG
XmNmaxWidth	XmCMaxWidth	int	XtUnspecifiedShellInt	CSG
XmNminAspectX	XmCMinAspectX	int	XtUnspecifiedShellInt	CSG
XmNminAspectY	XmCMinAspectY	int	XtUnspecifiedShellInt	CSG
XmNminHeight	XmCMinHeight	int	XtUnspecifiedShellInt	CSG
XmNminWidth	XmCMinWidth	int	XtUnspecifiedShellInt	CSG
XmNtitle	XmCTitle	String	dynamic	CSG
XmNtitleEncoding	XmCTitleEncoding	Atom	dynamic	CSG
XmNtransient	XmCTransient	Boolean	True	CSG
XmNwaitForWm	XmCWaitForWm	Boolean	True	CSG
XmNwidthInc	XmCWidthInc	int	XtUnspecifiedShellInt	CSG
XmNwindowGroup	XmCWindowGroup	Window	dynamic	CSG
XmNwinGravity	XmCWinGravity	int	dynamic	CSG
XmNwmTimeout	XmCWmTimeout	int	5000 ms	CSG

Note: If values for **XmNminWidth** and **XmNminHeight** are present, and values for **XmNbaseWidth** and **XmNbaseHeight** are absent, **XmNminWidth** and **XmNminHeight** will be used as default values for **XmNbaseWidth** and **XmNbaseHeight**, and these values will be added to the shell size specified by the user. To work around this, add arguments during widget creation to explicitly set **XmNbaseWidth** and **XmNbaseHeight** to zero.

Shell Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNallowShellResize	XmCAllowShellResize	Boolean	False	CG
XmNcreatePopup-ChildProc	XmCCreatePopup-ChildProc	XtCreatePopup-ChildProc	NULL	CSG
XmNgeometry	XmCGeometry	String	NULL	CSG
XmNoverrideRedirect	XmCOverrideRedirect	Boolean	False	CSG

XmNpopdownCallback	XmCCallback	XtCallbackList	NULL	C
XmNpopupCallback	XmCCallback	XtCallbackList	NULL	C
XmNsaveUnder	XmCSaveUnder	Boolean	True	CSG
XmNvisual	XmCVisual	Visual *	CopyFrom- Parent	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallback-List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen- Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG

XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for XmDialogShell.

Related Information

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

XmDisplay

Purpose

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**	**Type**	**Default**	**Access**
XmNdefaultButton- Emphasis	XmCDefaultButton- Emphasis	XtEnum	XmEXTERNAL_- HIGHLIGHT	C
XmNdefaultVirtual- Bindings	XmCDefaultVirtual- Bindings	String	dynamic	C
XmNdragInitiatorProtocol- Style	XmCDragInitiator- ProtocolStyle	unsigned char	XmDRAG_PREFER_- RECEIVER	CG
XmNdragReceiverProtocol- Style	XmCDragReceiver- ProtocolStyle	unsigned char	XmDRAG_PREFER_- DYNAMIC	CG
XmNdragStartCallback	XmCCallback	XtCallback- List	NULL	C
XmNenableBtn1Transfer	XmCEnableBtn1- Transfer	XtEnum	XmOFF	C
XmNenableButtonTab	XmCEnableButtonTab	Boolean	False	C
XmNenableDragIcon	XmCEnableDragIcon	Boolean	False	C
XmNenableEtchedInMenu	XmCEnable- EtchedInMenu	Boolean	False	C

XmNenableToggleColor	XmCEnable- ToggleColor	Boolean	False	C
XmNenableToggleVisual	XmCEnable- ToggleVisual	Boolean	False	C
XmNenableUnselectable-Drag	XmCEnableUnselectable-Drag	Boolean	True	C
XmNenableWarp	XmCEnableWarp	XtEnum	True	CSG
XmNmotifVersion	XmCMotifVersion	int	XmVERSION	CSG
XmNnoFontCallback	XmCCallback	XtCallback- List	NULL	C
XmNnoRenditionCallback	XmCCallback	XtCallback- List	NULL	C
XmNuserData	XmCUserData	XtPointer	NULL	CSG

XmNdefaultButtonEmphasis

Specifies whether to change the look of the PushButton widget and gadget that have the **XmNshowAsDefault** resource set. When the PushButton is the default, it has an etched out button which is enclosed with another etched in border. The **XmNdefaultButtonEmphasis** has the follow possible values, which affect the location cursor:

XmINTERNAL_HIGHLIGHT

Causes the location cursor to appear in between the two etched borders to minimize the space required.

XmEXTERNAL_HIGHLIGHT

Causes the PushButton to draw the location cursor outside the second border.

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      \n\
        ...
        osfLeft:       <Key>left, Ctrl<Key>H
```

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 default value of this resource is dependent on the capabilities of the display. If the display supports the shape extension, allowing the dynamic protocol to use arbitrarily sized drag cursors, the default of this resource is **XmDRAG_PREFER_DYNAMIC**, otherwise the default is **XmDRAG_PREFER_PREREGISTER**.

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**.

XmNdragStartCallback
Specifies the list of callbacks that are invoked when the **XmDragStart** function is called. The type of structure whose address is passed to this callback is **XmDragStartCallbackStruct**. The callback reason is **XmCR_DRAG_START**.

XmNenableBtn1Transfer
Specifies if selection and transfer actions are integrated on Btn1 and extend actions are activated on Btn2. This resource can take the following values:

XmOFF Disables integration and selection activation on Btn1.

XmBUTTON2_TRANSFER
Enables integration and selection activation on Btn1 and transfer on Btn2.

XmBUTTON2_ADJUST
Enables integration and selection activation on Btn1 and adjust on Btn2.

This resource affects the actions of Text, TextField, List, and Container.

XmNenableButtonTab
Specifies if the action for the **Tab** key (**KNextField** and **KPrevField** actions) is to be modified. A value of True modifies the key to move as an arrow key until the boundary of a tab group is reached. Then, at the boundary of the tab group, **KNextField** and **KPrevField** will move to the next or previous tab group, respectively. A value of False does not cause modification.

XmNenableDragIcon
Specifies which set of icons are to be used for system default cursors during drag and drop operations. A value of False specifies that earlier versions of Motif release icons are used, a value of True specifies that alternate icons are used. This resource affects both the 16x16 and the 32x32 icons that the system defaults for each of the Screen objects associated with this display.

XmNenableEtchedInMenu
Specifies the shadowing of the button widgets and gadgets in menus when the control is activated. A value of True causes the selected menu to be drawn with the shadow etched in; this shadow style is consistent with the selected appearance of other button widgets outside of menus. A value of False causes the selected menu to be draw with the shadow etched out. This resource affects the actions of PushButton, ToggleButton, and CascadeButton widgets and gadgets when they are children of Menu.

When this resource is set, the background of a button in a menu uses the **XmNselectColor** (derived from the **XmNselectPixel**) when armed as a default. A **PushButton** uses the **XmNarmColor** if it is defined. A **ToggleButton** uses the **XmNselectColor** if **XmNindicatorOn** is **False** and **XmNfillOnSelect** is **True**.

XmNenableToggleColor
Specifies how to determine the default value of the **XmNselectColor** resource of ToggleButton and ToggleButtonGadget. A value of True causes the default value of **XmNselectColor** to be set to the value of **XmNhighlightColor**. A value of False

causes the default value of **XmNselectColor** to be set to the value of **XmNbackground**. This resource only affects the appearance of ToggleButton widgets and gadgets that are in **XmONE_OF_MANY** or **XmONE_OF_MANY_ROUND** mode. In addition, **XmNenableToggleColor** only influences the default value of **XmNselectColor**. That is, if the user or application sets a value for **XmNselectColor**, then **XmNenableToggleColor** is ignored.

XmNenableToggleVisual
Specifies the visual appearance of the ToggleButton widget and/or gadget. This resource affects the default value of the ToggleButton[Gadget] **XmNindicatorType** and **XmNindicatorOn** resources. When the ToggleButton is in a RadioBox, a value of True causes the **XmONE_OF_MANY_ROUND** (a shadowed circle) to be the default. Otherwise, when this resource is True, the ToggleButton **XmNindicatorOn** resource causes a default of **XmN_OF_MANY**, which will be a shadowed square with a check mark (check box).

A value of False causes the following:

XmONE_OF_MANY
Is a shadowed diamond.

XmN_OF_MANY
Is a shadowed square.

XmNenableUnselectableDrag
Specifies whether or not it is possible to drag from Label and Scale. A value of True enables the drag; a value of False disables it.

XmNenableWarp
Specifies if an application is allowed to warp the pointer from the user. A value of True enables warping, a value of False does not.

XmNmotifVersion
Specifies the current version of Motif that the current implementation is supposed to behave like. By default, this resource gets its value from release values in **Xm.h**.

XmNnoFontCallback
This callback is called whenever a rendition attempts to load a font or fontset and fails. This can happen on creation if the font is specified as **XmLOAD_IMMEDIATE** or when an attempt is made to render an **XmString** using a font specified as **XmLOAD_DEFERRED**. An

application can have this callback attempt to remedy this problem by calling **XmRenditionUpdate** on the input rendition to provide a font for the widget to use. This may be done by either providing an an alternative font name to be loaded using the **XmNfontName** and **XmNfontType** resources or with an already loaded font using the **XmNfont** resource. The callback reason is **XmCR_NO_FONT**. This callback uses the **XmDisplayCallbackStruct** structure.

XmNnoRenditionCallback
This callback is called whenever an attempt is made to render a segment with a *RENDITION* tag which does not match any renditions in a given render table. The callback reason is **XmCR_NO_RENDITION**. This callback uses the **XmDisplayCallbackStruct** structure.

An application can have this callback attempt to remedy this problem by creating a new rendition with the given tag and adding it to **render_table**.

The **XmNnoRenditionCallback** should deallocate the render table passed in in the **render_table** field of the callback structure. Note that the table will automatically be deallocated if the **XmRenderTableAddRenditions** function is called on it. The callback should NOT deallocate the modified render table that is passed back to Motif in the **render_table** field. If the application wishes to manipulate this render table further, it should make a copy with the **XmRenderTableCopy** function before returning from the callback.

XmNuserData
Specifies a client data pointer for applications. An internally unused resource.

Inherited Resources

All of the superclass resources inherited by XmDisplay are designated N/A (not applicable).

Callback Information

A pointer to the following structure is passed to the **XmNdragStartCallback** callback:

```
typedef struct
{
        int reason;
        XEvent  *event;
        Widget timeStamp;
        Boolean doit;
}XmDragStartCallbackStruct;
```

reason Indicates why the callback was invoked

event Points to the *XEvent* that triggered the callback

widget Indicates the ID of the widget from which the drag was initiated.

doit Is an IN/OUT member that allows the callback to determine whether to continue with the drag or cancel. Setting *doit* to False will cancel the drag. The default value is NULL.

A pointer to the following structure is passed to the **XmNnoFontCallback** and **XmNnoRenditionCallback** callbacks:

```
typedef struct
{
        int reason;
        XEvent *event;
        XmRendition rendition;
        char *font_name;
        XmRenderTable render_table;
        XmStringTag tag;
}XmDisplayCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

rendition Specifies the rendition with the missing font.

font_name Specifies the name of the font or font set which could not be loaded.

render_table
Specifies the render table with the missing rendition.

tag Specifies the tag of the missing rendition.

The following table describes the reasons for which the individual callback structure fields are valid.

Reason	**Valid Fields**
XmCR_NO_FONT	*rendition*, *font_name*
XmCR_NO_RENDITION	*render_table*, *tag*

Related Information

ApplicationShell(3), **Composite**(3), **Core**(3), **TopLevelShell**(3), **VendorShell**(3), **WMShell**(3), **XmGetXmDisplay**(3), and **XmScreen**(3).

XmDragContext

Purpose

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	Type	Default	Access
XmNblendModel	XmCBlendModel	unsigned char	XmBLEND_ALL	CG
XmNclientData	XmCClientData	XtPointer	NULL	CSG
XmNconvertProc	XmCConvertProc	XtConvert-SelectionIncrProc	NULL	CSG
XmNcursorBackground	XmCCursor-Background	Pixel	dynamic	CSG
XmNcursorForeground	XmCCursorForeground	Pixel	dynamic	CSG
XmNdragDropFinish-Callback	XmCCallback	XtCallbackList	NULL	CSG
XmNdragMotion-Callback	XmCCallback	XtCallbackList	NULL	C

XmNdragOperations	XmCDragOperations	unsigned char	XmDROP_COPY | XmDROP_MOVE	C
XmNdropFinish-Callback	XmCCallback	XtCallbackList	NULL	C
XmNdropSiteEnter-Callback	XmCCallback	XtCallbackList	NULL	C
XmNdropSiteLeave-Callback	XmCCallback	XtCallbackList	NULL	C
XmNdropStartCallback	XmCCallback	XtCallbackList	NULL	C
XmNexportTargets	XmCExportTargets	Atom *	NULL	CSG
XmNincremental	XmCIncremental	Boolean	False	CSG
XmNinvalidCursor-Foreground	XmCCursor-Foreground	Pixel	dynamic	CSG
XmNnoneCursor-Foreground	XmCCursor-Foreground	Pixel	dynamic	CSG
XmNnumExportTargets	XmCNum-ExportTargets	Cardinal	0	CSG
XmNoperationChanged-Callback	XmCCallback	XtCallbackList	NULL	C
XmNoperationCursorIcon	XmCOperation-CursorIcon	Widget	dynamic	CSG
XmNsourceCursorIcon	XmCSource-CursorIcon	Widget	dynamic	CSG
XmNsourcePixmapIcon	XmCSource-PixmapIcon	Widget	dynamic	CSG
XmNstateCursorIcon	XmCStateCursorIcon	Widget	dynamic	CSG
XmNtopLevelEnter-Callback	XmCCallback	XtCallbackList	NULL	C
XmNtopLevelLeave-Callback	XmCCallback	XtCallbackList	NULL	C
XmNvalidCursor-Foreground	XmCCursor-Foreground	Pixel	dynamic	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

If **XmNincremental** is True, 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 is an *XtConvertSelectionProc*, and 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**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorS- ensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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**.

This field is invalid if the **dropAction** field is set to **XmDROP_CANCEL**.

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. The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

Button1<Enter>:
DragMotion()

Button1<Leave>:
DragMotion()

Button1<Motion>:
DragMotion()

Button2<Enter>:
DragMotion()

Button2<Leave>:
DragMotion()

Button2<Motion>:
DragMotion()

<Btn2Up>: **FinishDrag()**

<Btn1Up>: **FinishDrag()**

<Key>Return:
FinishDrag()

<Key>osfActivate:
FinishDrag()

<BtnDown>:
  IgnoreButtons()

<BtnUp>: **IgnoreButtons()**

:<Key>osfCancel:
  CancelDrag()

:<Key>osfHelp:
  HelpDrag()

:<Key>osfUp:
  DragKey(*Up***)**

:<Key>osfDown:
  DragKey(*Down***)**

:<Key>osfLeft:
  DragKey(*Left***)**

:<Key>osfRight:
  DragKey(*Right***)**

:<KeyUp>:
  DragKey(*Update***)**

:<KeyDown>:
  DragKey(*Update***)**

Action Routines

The XmDragContext action routines are

CancelDrag():
  Cancels the drag operation and frees the associated DragContext.

DragKey(*String*)
  If the value of *String* is **Left**, **Right**, **Up**, or **Down**, this action moves the dragged object in the corresponding location. Any other values of *String* are ignored.

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**(3).

Related Information

Core(3), **XmDisplay**(3), **XmDragCancel**(3), **XmDragIcon**(3), **XmDragStart**(3), **XmDropSite**(3), **XmDropTransfer**(3), and **XmScreen**(3).

XmDragIcon

Purpose

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**	**Type**	**Default**	**Access**
XmNattachment	XmCAttachment	unsigned char	XmATTACH_-NORTH_WEST	CSG
XmNdepth	XmCDepth	int	1	CSG
XmNheight	XmCHeight	Dimension	0	CSG
XmNhotX	XmCHot	Position	0	CSG
XmNhotY	XmCHot	Position	0	CSG
XmNmask	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNoffsetX	XmCOffset	Position	0	CSG
XmNoffsetY	XmCOffset	Position	0	CSG
XmNpixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNwidth	XmCWidth	Dimension	0	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**	**Type**	**Default**	**Access**
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C

Related Information

Object(3), **XmCreateDragIcon**(3), **XmDisplay**(3), **XmDragContext**(3), **XmDropSite**(3), **XmDropTransfer**(3), and **XmScreen**(3).

XmDrawingArea

Purpose

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.

DrawingArea uses the **XmNinitialFocus** resource of **XmManager** to define whether or not DrawingArea will receive focus when it is traversed to, even if it has traversable children. If **XmNinitialFocus** is NULL, DrawingArea receives focus only if it does not have any traversable children. If **XmNinitialFocus** is not NULL, then DrawingArea receives focus when traversed to. In the latter case, the application first needs to be able to realize that the DrawingArea will receive focus, then, as appropriate, needs to either call the **XmProcessTraversal** function for the desired child, or to navigate across the private DrawingArea graphics objects.

The following resources are not currently used by the DrawingArea widget: **XmNshadowThickness**, **XmNtopShadowPixmap**, **XmNbottomShadowPixmap**, **XmNtopShadowColor**, and **XmNbottomShadowColor**

Data Transfer Behavior

DrawingArea has no widget class conversion or destination procedure. Subclasses and the **XmNconvertCallback** procedures are responsible for any conversion of selections. Subclasses and the **XmNdestinationCallback** procedures are responsible for any data transfers to the widget.

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**	**Type**	**Default**	**Access**
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNdestinationCallback	XmCCallback	XtCallbackList	NULL	C
XmNexposeCallback	XmCCallback	XtCallbackList	NULL	C
XmNinputCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	10	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNresizeCallback	XmCCallback	XtCallbackList	NULL	C
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_ANY	CSG

XmNconvertCallback
Specifies a list of callbacks called when the DrawingArea is asked to convert a selection. The type of the structure whose address is passed to these callbacks is **XmConvertCallbackStruct**. The reason is **XmCR_OK**.

XmNdestinationCallback
Specifies a list of callbacks called when the DrawingArea is the destination of a transfer operation. The type of the structure whose

address is passed to these callbacks is **XmDestinationCallbackStruct**. The reason is **XmCR_OK**.

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**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG

XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow-Pixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG

XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback Information

A pointer to the following structure is passed to the **XmNexposeCallback**, **XmNinputCallback**, and **XmNresizeCallback** procedures:

```
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.

A pointer to the following structure is passed to the **XmNconvertCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent *event;
        Atom selection;
        Atom target;
        XtPointer source_data;
        XtPointer location_data;
        int flags;
        XtPointer parm;
        int parm_format;
        unsigned long parm_length;
        int status;
        XtPointer value;
        Atom type;
        int format;
        unsigned long length;
} XmConvertCallbackStruct;
```

reason	Indicates why the callback was invoked.
event	Points to the *XEvent* that triggered the callback. It can be NULL.
selection	Indicates the selection for which conversion is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.
target	Indicates the conversion target.
source_data	Contains information about the selection source. When the selection is _MOTIF_DROP, *source_data* is the DragContext. Otherwise, it is NULL.
location_data	Contains information about the location of data to be converted. If the value is NULL, the data to be transferred consists of the widget's current selection.
flags	Indicates the status of the conversion. Following are the possible values:

XmCONVERTING_NONE
This flag is currently unused.

XmCONVERTING_PARTIAL
The target widget was able to be converted, but some data was lost.

XmCONVERTING_SAME
The conversion target is the source of the data to be transferred.

XmCONVERTING_TRANSACT
This flag is currently unused.

parm Contains parameter data for this target. If no parameter data exists, the value is NULL.

When *selection* is *CLIPBOARD* and *target* is _MOTIF_CLIPBOARD_TARGETS or _MOTIF_DEFERRED_CLIPBOARD_TARGETS, the value is the requested operation (**XmCOPY**, **XmMOVE**, or **XmLINK**).

parm_format
Specifies whether the data in *parm* should be viewed as a list of *char*, *short*, or *long* quantities. Possible values are 0 (when *parm* is NULL), 8 (when the data in *parm* should be viewed as a list of *char*s), 16 (when the data in *parm* should be viewed as a list of *short*s), or 32 (when the data in *parm* should be viewed as a list of *long*s). Note that *parm_format* symbolizes a data type, not the number of bits in each list element. For example, on some machines, a *parm_format* of 32 means that the data in *parm* should be viewed as a list of 64-bit quantities, not 32-bit quantities.

parm_length
Specifies the number of elements of data in *parm*, where each element has the size specified by *parm_format*. When *parm* is NULL, the value is 0.

status An IN/OUT member that specifies the status of the conversion. The initial value is **XmCONVERT_DEFAULT**. The callback procedure can set this member to one of the following values:

XmCONVERT_DEFAULT
: This value means that the widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it overwrites the data provided by the callback procedures in the *value* member.

XmCONVERT_MERGE
: This value means that the widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it appends its data to the data provided by the callback procedures in the *value* member. This value is intended for use with targets that result in lists of data, such as *TARGETS*.

XmCONVERT_DONE
: This value means that the callback procedure has successfully finished the conversion. The widget class conversion procedure, if any, is not called after the callback procedures return.

XmCONVERT_REFUSE
: This value means that the callback procedure has terminated the conversion process without completing the requested conversion. The widget class conversion procedure, if any, is not called after the callback procedures return.

value An IN/OUT parameter that contains any data that the callback procedure produces as a result of the conversion. The initial value is NULL. If the callback procedure sets this member, it must ensure that the *type*, *format*, and *length* members correspond to the data in *value*. The callback procedure is responsible for allocating, but not for freeing, memory when it sets this member.

type An IN/OUT parameter that indicates the type of the data in the *value* member. The initial value is *INTEGER*.

format An IN/OUT parameter that specifies whether the data in *value* should be viewed as a list of *char*, *short*, or *long* quantities. The initial value is 8. The callback procedure can set this member to 8 (for a list of *char*), 16 (for a list of *short*), or 32 (for a list of *long*).

length An IN/OUT member that specifies the number of elements of data in *value*, where each element has the size symbolized by *format*. The initial value is 0.

A pointer to the following callback structure is passed to the **XmNdestinationCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent *event;
        Atom selection;
        XtEnum operation;
        int flags;
        XtPointer transfer_id;
        XtPointer destination_data;
        XtPointer location_data;
        Time time;
} XmDestinationCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which data transfer is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

operation Indicates the type of transfer operation requested.

- When the selection is *PRIMARY*, possible values are **XmMOVE**, **XmCOPY**, and **XmLINK**.
- When the selection is *SECONDARY* or *CLIPBOARD*, possible values are **XmCOPY** and **XmLINK**.
- When the selection is _MOTIF_DROP, possible values are **XmMOVE**, **XmCOPY**, **XmLINK**, and **XmOTHER**. A value of **XmOTHER** means that the callback procedure must get further information from the **XmDropProcCallbackStruct** in the *destination_data* member.

flags Indicates whether or not the destination widget is also the source of the data to be transferred. Following are the possible values:

XmCONVERTING_NONE
The destination widget is not the source of the data to be transferred.

XmCONVERTING_SAME
The destination widget is the source of the data to be transferred.

transfer_id Serves as a unique ID to identify the transfer transaction.

destination_data
Contains information about the destination. When the selection is _MOTIF_DROP, the callback procedures are called by the drop site's **XmNdropProc**, and *destination_data* is a pointer to the **XmDropProcCallbackStruct** passed to the **XmNdropProc** procedure. When the selection is *SECONDARY*, *destination_data* is an Atom representing a target recommmended by the selection owner for use in converting the selection. Otherwise, *destination_data* is NULL.

location_data
Contains information about the location where data is to be transferred. The value is always NULL when the selection is *SECONDARY* or *CLIPBOARD*. If the value is NULL, the data is to be inserted at the widget's cursor position. **location_data** is only valid for the duration of a transfer. Once *XmTransferDone* procedures start to be called, **location_data** will no longer be stable.

time Indicates the time when the transfer operation began.

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. The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<BtnDown>:
DrawingAreaInput()

<BtnUp>: **DrawingAreaInput()**

<KeyDown>:
DrawingAreaInput()ManagerGadgetKeyInput()

<KeyUp>: **DrawingAreaInput()**

Action Routines

The **XmDrawingArea** action routines are

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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateDrawingArea**(3), and **XmManager**(3).

XmDrawnButton

Purpose
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. DrawnButton uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits, and holds the *XmQTactivatable* trait.

Classes

BulletinBoard inherits behavior, resources, and traits from the **Core**, **Composite**, **Constraint**, and **XmManager** 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**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNdisarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNexposeCallback	XmCCallback	XtCallbackList	NULL	C
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	CSG
XmNpushButton-Enabled	XmCPushButtonEnabled	Boolean	False	CSG
XmNresizeCallback	XmCCallback	XtCallbackList	NULL	C
XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_ETCHED_IN	CSG

XmNactivateCallback
Specifies the list of callbacks that is called when the widget becomes selected. The reason sent by the callback is **XmCR_ACTIVATE**. This callback uses the *XmQTactivatable* trait.

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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	N/A
XmNacceleratorText	XmCAcceleratorText	XmString	NULL	N/A

XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelString	XmCXmString	XmString	"\0"	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	0	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	0	CSG
XmNmarginRight	XmCMarginRight	Dimension	0	CSG
XmNmarginTop	XmCMarginTop	Dimension	0	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	N/A
XmNmnemonic-CharSet	XmCMnemonicCharSet	String	XmFONTLIST_-DEFAULT_TAG	N/A
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CSG

XmPrimitive Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvert-Callback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG

XmNhighlight-OnEnter	XmCHighlightOn- Enter	Boolean	False	CSG
XmNhighlight-Pixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlight-Thickness	XmCHighlight-Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigation- Type	XmCNavigationType	XmNavigation-Type	XmNONE	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadow-Thickness	XmCShadowThickness	Dimension	2	CSG
XmNtopShadow-Color	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow-Pixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG

XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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. The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

c<Btn1Down>:
ButtonTakeFocus()

≈c<Btn1Down>:
Arm()

≈c<Btn1Down>,≈cBtn1Up:
Activate() Disarm()

≈c<Btn1Down>(2+):
MultiArm()

≈c<Btn1Up>(2+):
MultiActivate()

≈cBtn1Up:
Activate() Disarm()

:<Key>osfActivate:
PrimitiveParentActivate()

:<Key>osfCancel:
PrimitiveParentCancel()

:<Key>osfSelect:
ArmAndActivate()

:<Key>osfHelp:
Help()

≈s ≈m ≈a <Key>Return:
PrimitiveParentActivate()

≈s ≈m ≈a <Key>space:
ArmAndActivate()

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.

ButtonTakeFocus():
Causes the PushButton to take keyboard focus when **Ctrl<Btn1Down>** is pressed, without activating the widget.

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**(3).

Related Information

Core(3), **XmCreateDrawnButton**, **XmLabel**(3), **XmPrimitive**(3), **XmPushButton**, and **XmSeparator**(3).

XmDropTransfer

Purpose

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**	**Type**	**Default**	**Access**
XmNdropTransfers	XmCDropTransfers	XmDropTransfer-EntryRec *	NULL	CG
XmNincremental	XmCIncremental	Boolean	False	CSG
XmNnumDrop-Transfers	XmCNumDrop-Transfers	Cardinal	0	CSG
XmNtransferProc	XmCTransferProc	XtSelection-CallbackProc	NULL	CSG
XmNtransferStatus	XmCTransferStatus	unsigned char	XmTRANSFER_-SUCCESS	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**	**Type**	**Default**	**Access**
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C

Related Information

Object(3), **XmDisplay**(3), **XmDragContext**(3), **XmDragIcon**(3), **XmDropSite**(3), **XmDropTransferAdd**(3), and **XmDropTransferStart**(3).

XmFileSelectionBox

Purpose

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
- An optional text input field for displaying and editing a filter 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**. The layout direction of the buttons depends on the **XmNlayoutDirection** resource.

####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:

FilterLabel	**FilterText**
Filter Text	**TextField**
Directory List	**DirList**
Directory List Label	**Dir**
DirL	**Label**
DirText	**TextField**

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.

Optionally, the search pattern mask and the base directory can be displayed in two separate text fields. This option is controlled by the **XmNpathMode** resource. Using this alternate display does not change the meaning of resources that control the content of these fields: **XmNdirectory**, **XmNdirMask**, **XmNpattern**.

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 or, when the FileSelectionBox has the optional **XmNpathMode XmPATH_MODE_RELATIVE** display, the filter text field. 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**)

FileSelectionBox uses the *XmQTactivatable* trait.

Data Transfer Behavior

Child widgets of a FileSelectionBox support the data transfer operations and targets associated with their widget classes.

In addition, if the source of a data transfer is the directory list and if **XmNdirSearchProc** has its default value, the directory list supports the *FILE* and *FILE_NAME* targets.

If the source of a data transfer is the file list and if **XmNfileSearchProc** has its default value, the file list supports the *FILE* and *FILE_NAME* targets.

In either case, FileSelectionBox adds an **XmNconvertCallback** procedure to the appropriate list. This procedure adds *FILE* and *FILE_NAME* to the *TARGETS* returned by the list. It treats requests for conversion of a selection to *FILE* and *FILE_NAME* exactly like requests for conversion to *TEXT*.

If an application changes **XmNdirSearchProc** or **XmNfileSearchProc** and wants to support the *FILE* and *FILE_NAME* targets on the corresponding list, it must provide support itself by adding a procedure to the list's **XmNconvertCallback** list.

Descendants

FileSelectionBox automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
=		
Apply	**XmPushButtonGadget**	Apply button
Cancel	**XmPushButtonGadget**	Cancel button
Dir	**XmLabelGadget**	title above list of directories
DirList	**XmList**	list of directories
DirListSW	**XmScrolledWindow**	ScrolledWindow parent of **DirList**
FilterLabel	**XmLabelGadget**	title above filter box
FilterText	**XmText** or **XmTextField**	text within filter box
Help	**XmPushButtonGadget**	Help button
Items	**XmLabelGadget**	title above list of filenames
ItemsList	**XmList**	list of filenames
ItemsListSW	**XmScrolledWindow**	ScrolledWindow parent of **ItemsList**
OK	**XmPushButtonGadget**	OK button
Selection	**XmLabelGadget**	title above selection box
Separator	**XmSeparatorGadget**	optional dividing line
Text	**XmText** or **XmTextField**	text within selection box

Classes

FileSelectionBox inherits behavior, resources, and traits 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	Type	Default	Access
XmNdirectory	XmCDirectory	XmString	dynamic	CSG
XmNdirectoryValid	XmCDirectoryValid	Boolean	dynamic	SG
XmNdirListItems	XmCDirListItems	XmStringTable	dynamic	SG
XmNdirListItemCount	XmCDirListItemCount	int	dynamic	SG
XmNdirListLabel-String	XmCDirListLabelString	XmString	dynamic	CSG
XmNdirMask	XmCDirMask	XmString	dynamic	CSG
XmNdirSearchProc	XmCDirSearchProc	XmSearchProc	default procedure	CSG
XmNdirSpec	XmCDirSpec	XmString	dynamic	CSG
XmNdirTextLabel-String	XmCDirTextLabelString	XmString	NULL	C
XmNfileFilterStyle	XmCFileFilterStyle	XtEnum	XmFILTER_- NONE	C
XmNfileListItems	XmCItems	XmStringTable	dynamic	SG
XmNfileListItem-Count	XmCItemCount	int	dynamic	SG

XmNfileListLabel-String	XmCFileListLabel- String	XmString	dynamic	CSG
XmNfileSearchProc	XmCFileSearchProc	XmSearchProc	default procedure	CSG
XmNfileTypeMask	XmCFileTypeMask	unsigned char	XmFILE_-REGULAR	CSG
XmNfilterLabel- String	XmCFilterLabelString	XmString	dynamic	CSG
XmNlistUpdated	XmCListUpdated	Boolean	dynamic	SG
XmNnoMatchString	XmCNoMatchString	XmString	" [ ] "	CSG
XmNpathMode	XmCPathMode	XtEnum	XmPATH_-MODE_FULL	C
XmNpattern	XmCPattern	XmString	dynamic	CSG
XmNqualifySearch-DataProc	XmCQualifySearch-DataProc	XmQualifyProc	default procedure	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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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.

XmNdirTextLabelString

Uses the specified **XmString** as the label above the TextField directory. The resource takes effect when the **XmNpathMode** resource has a value of **XmPATH_MODE_RELATIVE**. It is ignored when the **XmNpathMode** resource has a value of **XmPATH_MODE_FULL**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

XmNfileFilterStyle

Specifies whether or not the "hidden" files (those whose names begin with . (period) in POSIX systems) will be listed in the file and directory scrolling lists (where the default directory search procedure is used). The possible values are:

XmFILTER_NONE

Does not filter hidden files.

XmFILTER_HIDDEN_FILES

Restricts the list of possible file names, such as those beginning with . (period).

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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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. The possible values are

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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

XmNlistUpdated
Specifies an attribute that is set only by the directory and file search procedures. This resource is 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.

XmNpathMode

Specifies whether or not an additional text field will be used to display and edit the filter. The possible values are

XmPATH_MODE_FULL

Specifies that no additional text field will be used to display the filter. There will just be a single text field to display **XmNdirMask**.

XmPATH_MODE_RELATIVE

Specifies that there will be two text field displays, one to display the **XmNdirectory** and one to display the **XmNpattern**. In this instance, the **XmNfilterLabelString** resource applies to the text field for **XmNpattern** and **XmNdirTextLabelString** applies to the text field for **XmNdirectory**.

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 structure contains the input data. The second callback structure 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 structure.

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**	**Type**	**Default**	**Access**
XmNapplyCallback	XmCCallback	XtCallbackList	NULL	C
XmNapplyLabelString	XmCApplyLabel-String	XmString	dynamic	CSG
XmNcancelCallback	XmCCallback	XtCallbackList	NULL	C
XmNcancelLabelString	XmCCancel-LabelString	XmString	dynamic	CSG
XmNchildPlacement	XmCChildPlacement	unsigned char	XmPLACE_ABOVE_-SELECTION	CSG
XmNdialogType	XmCDialogType	unsigned char	XmDIALOG_FILE_-SELECTION	G
XmNhelpLabelString	XmCHelpLabelString	XmString	dynamic	CSG
XmNlistItemCount	XmCItemCount	int	dynamic	CSG
XmNlistItems	XmCItems	XmStringTable	dynamic	CSG
XmNlistLabelString	XmCListLabelString	XmString	dynamic	CSG
XmNlistVisible-ItemCount	XmCVisible-ItemCount	int	dynamic	CSG
XmNminimizeButtons	XmCMinimize-Buttons	Boolean	False	CSG

XmNmustMatch	XmCMustMatch	Boolean	False	CSG
XmNnoMatchCallback	XmCCallback	XtCallbackList	NULL	C
XmNokCallback	XmCCallback	XtCallbackList	NULL	C
XmNokLabelString	XmCOkLabelString	XmString	dynamic	CSG
XmNselectionLabel-String	XmCSelection-LabelString	XmString	dynamic	CSG
XmNtextAccelerators	XmCText-Accelerators	XtAccelerators	default	C
XmNtextColumns	XmCColumns	short	dynamic	CSG
XmNtextString	XmCTextString	XmString	dynamic	CSG

XmBulletinBoard Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNallow- Overlap	XmCAllow- Overlap	Boolean	True	CSG
XmNauto- Unmanage	XmCAuto- Unmanage	Boolean	False	CG
XmNbutton- FontList	XmCButton- FontList	XmFontList	dynamic	CSG
XmNbutton- RenderTable	XmCButton- RenderTable	XmRender- Table	dynamic	CSG
XmNcancelButton	XmCWidget	Widget	Cancel button	SG
XmNdefault- Button	XmCWidget	Widget	OK button	SG
XmNdefault- Position	XmCDefault- Position	Boolean	True	CSG
XmNdialogStyle	XmCDialogStyle	unsigned char	dynamic	CSG
XmNdialogTitle	XmCDialogTitle	XmString	NULL	CSG
XmNfocus- Callback	XmCCallback	XtCallback- List	NULL	C
XmNlabel- FontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabel- RenderTable	XmCLabel- RenderTable	XmRender- Table	dynamic	CSG
XmNmapCallback	XmCCallback	XtCallback- List	NULL	C
XmNmargin- Height	XmCMarginHeight	Dimension	10	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNnoResize	XmCNoResize	Boolean	False	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_-ANY	CSG

XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_-OUT	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRender- Table	XmCTextRender- Table	XmRender- Table	dynamic	CSG
XmNtext- Translations	XmCTranslations	XtTranslations	NULL	C
XmNunmap- Callback	XmCCallback	XtCallback- List	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_- GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	N/A
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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* This member is obsolete and exists for compatibility with earlier releases.
mask	Specifies the current value of **XmNdirMask**
mask_length	Specifies the number of bytes in *mask* This member is obsolete and exists for compatibility with earlier releases.
dir	Specifies the current base directory
dir_length	Specifies the number of bytes in *dir* This member is obsolete and exists for compatibility with earlier releases.
pattern	Specifies the current search pattern
pattern_length	Specifies the number of bytes in *pattern* This member is obsolete and exists for compatibility with earlier releases.

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(*Previous*|*Next*|*First*|*Last*):

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 **Previous**, or 0 (zero) for compatibility, this action selects the previous item in the list and replaces the text with that item.

When called with an argument of **Next**, or 1 for compatibility, this action selects the next item in the list and replaces the text with that item.

When called with an argument of **First**, or 2 for compatibility, this action selects the first item in the list and replaces the text with that item.

When called with an argument of **Last**, or 3 for compatibility, 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:

KeyosfCancel:
: 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.

KeyosfActivate 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**.

KeyosfActivate 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**.

Btn1Down**(2+)** or KeyosfActivate 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**.

Btn1Down**(2+)** or KeyosfActivate in File List:
: Calls the file list widget's **XmNdefaultActionCallback** callbacks and calls the **XmNokCallback** callbacks with reason **XmCR_OK**.

KeyosfSelect 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.

KeyosfSelect in File List:
: Replaces the selection text with the selected list item.

Btn2Down in File List:
: Drags the content of one or more selected list items using the drag and drop facility. If **<Btn2Down** is pressed on an unselected item, drags only that item, excluding any other selected items.

: This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures of the file list, possibly multiple times, for the _MOTIF_DROP selection.

Btn2Down in Directory List:

Drags the content of one or more selected list items using the drag and drop facility. If **<Btn2Down** is pressed on an unselected item, it drags only that item, excluding any other selected items.

This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures of the directory list, possibly multiple times, for the _MOTIF_DROP selection.

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, 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**.

KeyosfActivate:

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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmBulletinBoard**(3), **XmCreateFileSelectionBox**(3), **XmCreateFileSelectionDialog**(3),

XmFileSelectionBoxGetChild(3), **XmFileSelectionDoSearch**(3), **XmManager**(3), and **XmSelectionBox**(3).

XmForm

Purpose

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, resources, and traits 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**	**Type**	**Default**	**Access**
XmNfractionBase	XmCMaxValue	int	100	CSG
XmNhorizontalSpacing	XmCSpacing	Dimension	0	CSG
XmNrubberPositioning	XmCRubberPositioning	Boolean	False	CSG
XmNverticalSpacing	XmCSpacing	Dimension	0	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 depends on the value of the **XmNlayoutDirection**.)

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**	**Type**	**Default**	**Access**
XmNbottomAttachment	XmCAttachment	unsigned char	XmATTACH_NONE	CSG
XmNbottomOffset	XmCOffset	int	0	CSG
XmNbottomPosition	XmCPosition	int	0	CSG
XmNbottomWidget	XmCWidget	Widget	NULL	CSG
XmNleftAttachment	XmCAttachment	unsigned char	XmATTACH_NONE	CSG
XmNleftOffset	XmCOffset	int	0	CSG
XmNleftPosition	XmCPosition	int	0	CSG
XmNleftWidget	XmCWidget	Widget	NULL	CSG
XmNresizable	XmCBoolean	Boolean	True	CSG
XmNrightAttachment	XmCAttachment	unsigned char	XmATTACH_NONE	CSG
XmNrightOffset	XmCOffset	int	0	CSG
XmNrightPosition	XmCPosition	int	0	CSG
XmNrightWidget	XmCWidget	Widget	NULL	CSG
XmNtopAttachment	XmCAttachment	unsigned char	XmATTACH_NONE	CSG
XmNtopOffset	XmCOffset	int	0	CSG
XmNtopPosition	XmCPosition	int	0	CSG
XmNtopWidget	XmCWidget	Widget	NULL	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.

XmATTACH_FORM

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 depends on the value of the **XmNlayoutDirection** 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 depends on the value of the **XmNlayoutDirection** 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 depends on the value of the **XmNlayoutDirection** 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 depends on the value of the **XmNlayoutDirection** 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 depends on the value of the **XmNlayoutDirection** 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 depends on the value of the **XmNlayoutDirection** 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 depends on the value of the **XmNlayoutDirection** 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 that whether this resource actually applies to the left or right side of the child and its attachment depends on the value of the **XmNlayoutDirection** 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 the **XmNbottomAttachment** resource 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 a value of 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**	**Type**	**Default**	**Access**
XmNallowOverlap	XmCAllowOverlap	Boolean	True	CSG
XmNautoUnmanage	XmCAutoUnmanage	Boolean	True	CG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNcancelButton	XmCWidget	Widget	NULL	SG
XmNdefaultButton	XmCWidget	Widget	NULL	SG
XmNdefaultPosition	XmCDefaultPosition	Boolean	True	CSG
XmNdialogStyle	XmCDialogStyle	unsigned char	dynamic	CSG
XmNdialogTitle	XmCDialogTitle	XmString	NULL	CSG
XmNfocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNmapCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	0	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	0	CSG

XmNnoResize	XmCNoResize	Boolean	False	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_ANY	CSG
XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_OUT	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNtextTranslations	XmCTranslations	XtTranslations	NULL	C
XmNunmapCallback	XmCCallback	XtCallbackList	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallback- List	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	N/A
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallback- List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

XmForm inherits translations from XmBulletinBoard.

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmBulletinBoard**(3), **XmCreateForm**, **XmCreateFormDialog**(3), and **XmManager**(3).

XmFrame

Purpose

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 wants 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**	**Type**	**Default**	**Access**
XmNmarginWidth	XmCMarginWidth	Dimension	0	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	0	CSG
XmNshadowType	XmCShadowType	unsigned char	dynamic	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**	**Type**	**Default**	**Access**
XmNchildType	XmCChildType	unsigned char	XmFRAME_-WORKAREA_CHILD	CSG
XmNchildHorizontal-Alignment	XmCChildHorizontal-Alignment	unsigned char	XmALIGNMENT_-BEGINNING	CSG
XmNchildHorizontal-Spacing	XmCChildHorizontal-Spacing	Dimension	dynamic	CSG
XmNchildVertical-Alignment	XmCChildVertical-Alignment	unsigned char	XmALIGNMENT_- CENTER	CSG
XmNframeChild- Type	XmCFrameChildType	unsigned char	XmFRAME_WORKAREA_-CHILD	CSG

XmNchildType
Refer to the **XmNframeChildType** resource description. The **XmNchildType** resource is obsoleted by **XmNframeChildType**, but is kept here for backward compatibility.

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_CHILD_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_CHILD_BOTTOM

Causes the bottom edge of the title area to align vertically with the top shadow of the Frame.

XmNframeChildType

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**. This resource replaces **XmNchildType**.

Inherited Resources

Frame 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**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallback- List	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhenManaged	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

XmFrame inherits translations from XmManager.

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateFrame**(3), and **XmManager**(3).

XmGadget

Purpose

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**.

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**	**Type**	**Default**	**Access**
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNbottomShadow- Color	XmCBottomShadow- Color	Pixel	dynamic	CSG
XmNbottomShadow- Pixmap	XmCBottomShadow- Pixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOn- Enter	XmCHighlightOnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlight- Thickness	XmCHighlightThickness	Dimension	2	CSG
XmNlayout- Direction	XmNCLayoutDirection	XmDirection	dynamic	CG
XmNnavigation- Type	XmCNavigationType	XmNavigation- Type	XmNONE	CSG
XmNshadow- Thickness	XmCShadowThickness	Dimension	2	CSG
XmNtopShadow- Color	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

XmNbackground
Specifies the background color for the gadget.

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

XmNbottomShadowColor
Contains the color to use to draw the bottom and right sides of the border shadow.

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 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.

XmNhighlightPixmap
Specifies the pixmap used to draw the highlighting rectangle.

XmNhighlightThickness
Specifies the thickness of the highlighting rectangle.

XmNlayoutDirection
Specifies the direction in which components of the manager (including strings) are laid out. The values are of type **XmDirection**. If the widget's parent is a manager or shell, the value is inherited from the widget's parent. Otherwise, it is inherited from the closest ancestor vendor or menu shell.

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.

XmNtopShadowPixmap
Specifies the pixmap 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**.

The unit type can also be specified in resource files, with the following format:

```
<floating value><unit>
```

where:

unit	is <" ", pixels, inches, centimeters, millimeters, points, font units>
pixels	is <*pix*, *pixel*, *pixels*>
inches	is <*in*, *inch*, *inches*>
centimeter	is <*cm*, *centimeter*, *centimeters*>
millimeters	is <*mm*, *millimeter*, *millimeters*>
points	is <*pt*, *point*, *points*>
font units	is <*fu*, *font_unit*, *font_units*
float	is {"+"|"-"}{{<"0"-"9">*}.}<"0"-"9">*

Note that the type Dimension must always be positive.

For example,

```
xmfonts*XmMainWindow.height: 10.4cm
*PostIn.width: 3inches
```

XmNunitType can have the following values:

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

XmMILLIMETERS
All values provided to the widget are treated as normal millimeter values.

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

XmCENTIMETERS
All values provided to the widget are treated as normal centimeter values.

XmINCHES
All values provided to the widget are treated as normal inch values.

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

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

XmFONT_UNITS
All values provided to the widget are treated as normal font units. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

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 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**	**Type**	**Default**	**Access**
XmNancestor-Sensitive	XmCSensitive	Boolean	dynamic	G
XmNborder- Width	XmCBorder- Width	Dimension	0	N/A
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Object Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	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(3), **RectObj**(3), **XmManager**(3), and **XmScreen**(3).

XmIconGadget

Purpose

The IconGadget widget class

Synopsis

#include <Xm/IconG.h>

Description

IconGadget is an instantiable widget used to display both text and a pixmap in various combinations. Other widgets that hold the **XmQTcontainer** trait, such as Container, can use IconGadget to represent objects.

IconGadget text is a compound string. If no text is supplied, then the compound string is generated from the gadget name. IconGadget text is placed relative to the type of associated pixmap.

Depending upon the **XmNviewType** resource, IconGadget can display two views:

XmLARGE_ICON
The IconGadget text string is displayed below the pixmap, and centered.

XmSMALL_ICON
The IconGadget text string is placed on the side of the small icon, in the widget's **XmNlayoutDirection**.

A bitmap mask can be supplied for each pixmap to clip the pixmap into some shape other than a rectangle. The **XmNlargeIconMask** and **XmNsmallIconMask** resources specify the large and small bitmap masks respectively. Visual emphasis for the IconGadget is provided with the **XmNvisualEmphasis** resource. IconGadget's **XmNdetail** and **XmNdetailCount** resources provide a detail view for IconGadgets, enabling the display of Strings alongside the IconGadget. The exact layout ordering of the strings depends on the associated containing widget.

IconGadget uses the **XmQTcontainer** and *XmQTspecifyRenderTable* traits, and holds the *XmQTcareParentVisual* and *XmQTcontainerItem* traits.

Classes

IconGadget inherits behaviour, resources, and traits from **Object, RectObject**, and **XmGadget** classes.

The class pointer is **xmIconGadgetClass**.

The class name is **XmIconGadget**.

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 whether 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).

XmIconGadget Resource Set

Name	Class	Type	Default	Access
Xmalignment	XmCAlignment	unsigned char	XmALIGNMENT_-CENTER	CSG
XmNdetail	XmCDetail	XmStringTable	NULL	CSG
XmNdetailCount	XmCDetailCount	Cardinal	0	CSG
XmNfontList	XmCFontList	XmFontList	NULL	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlargeIcon- Mask	XmCIconMask	Pixmap	dynamic	CSG
XmNlargeIcon-Pixmap	XmCIconPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNmargin- Height	XmCMarginHeight	Dimension	2	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNsmall-IconMask	XmCIconMask	Pixmap	dynamic	CSG
XmNsmallIcon-Pixmap	XmCIconPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG

XmNviewType	XmCViewType	unsigned char	XmLARGE_ICON	CSG
XmNvisual-Emphasis	XmCVisual-Emphasis	unsigned char	XmNOT_SELECTED	CSG
XmNspacing	XmCSpacing	Dimension	4	CSG

XmNalignment
Specifies the horizontal alignment of the pixmap with respect to the label when the icon is in *LARGE_ICON* view. Valid values are **XmALIGNMENT_BEGINNING**, **XmALIGNMENT_CENTER**, and **XmALIGNMENT_END**.

XmNdetail Specifies an array of **XmString**s that are the detail information associated with the gadget.

XmNdetailCount
Specifies the size of the **XmNdetail** array.

XmNfontList
Specifies the font list associated with **XmIconGadget**. The font list is an obsolete construct, and has been superseded by the render table. It is included for compatibility with earlier versions of Motif, and for applications that do not easily support render tables. The default font list is derived from the default render table, and if both a font list and a render table are specified, the render table takes precedence.

XmNlabelString
Specifies the compound string. If this value is NULL, it is initialized by converting the name of the gadget to a compound string. Refer to **XmString**(3) for more information on the creation and structure of compound strings.

XmNlargeIconMask
Specifies the icon mask used when **XmNviewType** is **XmLARGE_ICON**.

XmNlargeIconPixmap
Specifies the pixmap when **XmNviewType** is **XmLARGE_ICON**. If this resource's value is **XmUNSPECIFIED_PIXMAP**, there is no pixmap. If a large icon pixmap is specified, and if during conversion an associated mask can be fetched, then the **XmNlargeIconMask** resource is set to that mask.

XmNmarginHeight
Specifies the amount of vertical space between the highlight and the inside (pixmap and label).

XmNmarginWidth
Specifies the amount of horizontal space between the highlight and the inside (pixmap and label).

XmNrenderTable
Specifies the **XmRenderTable** of the text used in the gadget. If **XmNrenderTable** is NULL when the IconGadget is created, the parent's render table resource value is used if there is a render table. If the parent does not have a render table, the parent hierarchy of the widget is searched for a widget that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the render table is initialized to the **XmLABEL_RENDER_TABLE** value of the ancestor widget. If no such widget is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a **XmRenderTable**. If both a render table and a font list are specified, the render table will take precedence.

XmNsmallIconMask
Specifies the icon mask used when **XmNviewType** is **XmSMALL_ICON**.

XmNsmallIconPixmap
Specifies the pixmap when **XmNviewType** is **XmSMALL_ICON**. If this resource's value is **XmUNSPECIFIED_PIXMAP**, there is no pixmap. If a small icon pixmap is specified, and if during conversion an associated mask can be fetched, then the **XmNsmallIconMask** resource is set to that mask.

XmNspacing
Specifies the amount of space between the pixmap and the label parts of the icon.

XmNviewType
Specifies the view (combination of pixmaps/text) that will be displayed. If the IconGadget is a child of a Container widget, however, then the specification of this resource will be taken from the Container—if Container's **XmNentryViewType** is either **XmLARGE_ICON** or **XmSMALL_ICON**, then IconGadget's **XmNviewType** takes that

value; otherwise, the default is **XmLARGE_ICON**. This resource is set to one of the following:

XmLARGE_ICON
The pixmap specified by **XmNlargeIconPixmap** is displayed with the **XmNlabelString** beneath it.

XmSMALL_ICON
The pixmap specified by **XmNsmallIconPixmap** is displayed with the **XmNlabelString** displayed in the direction of the **XmNlayoutDirection** resource.

XmNvisualEmphasis
Specifies the visual state of the IconGadget. If the IconGadget is in a selected state all visuals are displayed using the Container **XmNselectColor** resource. It is set to one of the following:

XmSELECTED
The IconGadget is in the selected state and displays the appropriate visuals.

XmNOT_SELECTED
The IconGadget is not in the selected state.

Inherited Resources

IconGadget 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**	**Type**	**Default**	**Access**
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNbottomShadow-Color	XmCBottomShadow- Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG

XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlight- Thickness	XmCHighlight- Thickness	Dimension	0	CSG
XmNlayoutDirection	XmNCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmNONE	CSG
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	False	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

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

Additional Behavior

IconGadget has no behavior.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Errors/Warnings

The toolkit will display a warning if an incorrect value is given for an enumeration resource.

Related Information

Core(3), **XmContainer**(3), **XmCreateIconGadget**(3), and **XmGadget**(3).

XmLabel

Purpose

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 *Motif 2.1—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, depending on the layout direction.

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 menu panes, Pulldown menu panes, and OptionMenus. Otherwise, these resources default to False.

Label uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits, and holds the *XmQTaccessTextual*, *XmQTmenuSavvy*, and *XmQTtransfer* traits.

Data Transfer Behavior

Label and it subclasses, except when used in a menu system, support dragging of the label contents from the Label. However, the label contents are draggable only if the **XmNenableUnselectableDrag** resource of **XmDisplay** is set to True.

As a source of data, Label and its subclasses support the following targets and associated conversions of data to these targets:

locale If the *locale* target matches the widget's locale, the widget transfers **XmNlabelString** in the encoding of the locale. This target is supported only when **XmNlabelType** is **XmSTRING**.

COMPOUND_TEXT
The widget transfers **XmNlabelString** as type *COMPOUND_TEXT*. This target is supported only when **XmNlabelType** is **XmSTRING**.

PIXMAP The widget transfers **XmNlabelPixmap** as type *DRAWABLE*. This target is supported only when **XmNlabelType** is **XmPIXMAP**.

STRING The widget transfers **XmNlabelString** as type *STRING*. This target is supported only when **XmNlabelType** is **XmSTRING**.

TEXT If **XmNlabelString** is fully convertible to the encoding of the locale, the widget transfers **XmNlabelString** in the encoding of the locale. Otherwise, the widget transfers **XmNlabelString** as type *COMPOUND_TEXT*. This target is supported only when **XmNlabelType** is **XmSTRING**.

_MOTIF_CLIPBOARD_TARGETS
The widget transfers, as type *ATOM*, a list of the targets it supports for the *CLIPBOARD* selection. When **XmNlabelType** is **XmSTRING**, these include the following targets:

- _MOTIF_COMPOUND_STRING
- *COMPOUND_TEXT*
- The encoding of the locale, if **XmNlabelString** is fully convertible to the encoding of the locale

- *STRING*, if **XmNlabelString** is fully convertible to *STRING*

When **XmNlabelType** is **XmPIXMAP**, the targets include *PIXMAP*.

_MOTIF_COMPOUND_STRING
: The widget transfers **XmNlabelString** as a compound string in Byte Stream format. This target is supported only when **XmNlabelType** is **XmSTRING**.

_MOTIF_EXPORT_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets to be used as the value of the DragContext’s **XmNexportTargets** in a drag-and-drop transfer. When **XmNlabelType** is **XmSTRING**, these include _MOTIF_COMPOUND_STRING, *COMPOUND_TEXT*, the encoding of the locale, *STRING*, *TEXT*, *BACKGROUND*, and *FOREGROUND*. When **XmNlabelType** is **XmPIXMAP**, these include *PIXMAP*, *BACKGROUND*, and *FOREGROUND*.

As a source of data, Label also supports the following standard Motif targets:

BACKGROUND
: The widget transfers **XmNbackground** as type *PIXEL*.

CLASS
: The widget finds the first shell in the widget hierarchy that has a WM_CLASS property and transfers the contents as text in the current locale.

CLIENT_WINDOW
: The widget finds the first shell in the widget hierarchy and transfers its window as type *WINDOW*.

COLORMAP
: The widget transfers **XmNcolormap** as type *COLORMAP*.

FOREGROUND
: The widget transfers **XmNforeground** as type *PIXEL*.

NAME
: The widget finds the first shell in the widget hierarchy that has a WM_NAME property and transfers the contents as text in the current locale.

TARGETS
: The widget transfers, as type *ATOM*, a list of the targets it supports. These include the standard targets in this list. When **XmNlabelType** is **XmSTRING**, these also include _MOTIF_COMPOUND_STRING,

COMPOUND_TEXT, the encoding of the locale, *STRING*, and *TEXT*. When **XmNlabelType** is **XmPIXMAP**, these also include *PIXMAP*.

TIMESTAMP
The widget transfers the timestamp used to acquire the selection as type *INTEGER*.

_MOTIF_RENDER_TABLE
The widget transfers **XmNrenderTable** if it exists, or else the default text render table, as type *STRING*.

_MOTIF_ENCODING_REGISTRY
The widget transfers its encoding registry as type *STRING*. The value is a list of NULL separated items in the form of tag encoding pairs. This target symbolizes the transfer target for the Motif Segment Encoding Registry. Widgets and applications can use this Registry to register text encoding formats for specified render table tags. Applications access this Registry by calling **XmRegisterSegmentEncoding** and **XmMapSegmentEncoding**.

Classes

Label inherits behavior, resources, and traits 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	Type	Default	Access
XmNaccelerator	XmCAccelerator	String	NULL	CSG

XmNacceleratorText	XmCAcceleratorText	XmString	NULL	CSG
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	0	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	0	CSG
XmNmarginRight	XmCMarginRight	Dimension	0	CSG
XmNmarginTop	XmCMarginTop	Dimension	0	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonic-CharSet	XmCMnemonicCharSet	String	XmFONTLIST_-DEFAULT_TAG	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString-Direction	dynamic	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 menu panes.

XmNacceleratorText

Specifies the text displayed for the accelerator. The text is displayed adjacent to the label string or pixmap. The direction of its layout depends on the **XmNlayoutDirection** resource of the widget. 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 **XmNlayoutDirection** is **XmLEFT_TO_RIGHT**. When that resource is **XmRIGHT_TO_LEFT**, 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. **XmNfontList** is obsolete and exists for compatibility with previous releases. You should now use **XmNrenderTable** instead of **XmNfontList**. If both are specified, the render table will take precedence. If **XmNfontList** is NULL at initialization, Label searches its parent hierarchy for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or XmMenuShell widget class. If such an ancestor is found, Label initializes **XmNfontList** to the **XmNlabelFontList** of the ancestor widget. Similarly, button subclasses of Label initialize **XmNfontList** to the **XmNbuttonFontList** of the ancestor widget. (Currently, all subclasses of Label are button subclasses.) If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList**(3) 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**(3) 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 depends on the value of the **XmNlayoutDirection** resource.

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 depends on the value of the **XmNlayoutDirection** resource.

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 menu pane, or a Pulldown menu pane 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.

XmNrenderTable

Specifies the render table associated with the **labelString**. If this value is NULL at initialization, Label searches its parent hierarchy for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, Label initializes **XmNrenderTable** to the **XmLABEL_RENDER_TABLE** value of the ancestor widget. Similarly, button subclasses of Label initialize **XmNrenderTable** to the **XmBUTTON_RENDER_TABLE** value of the ancestor widget. (Note that all current subclasses of Label are button subclasses.) If no such ancestor is found, the default is implementation dependent. If a font list and a render table are both sepcified, the render table will take precedence. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

XmNstringDirection

Is a synthetic resource for setting **XmNlayoutDirection**. The values for this resource are **XmSTRING_DIRECTION_L_TO_R** and **XmSTRING_DIRECTION_R_TO_L**. Refer to the **XmNlayoutDirection** resource description. The **XmNstringDirection** resource is obsoleted by **XmNlayoutDirection**, but is kept here for backward compatibility.

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**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOn- Enter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	0	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmNONE	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	False	CSG

XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

XmLabel includes translations from Primitive. The XmLabel translations are described in the following list.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<Btn2Down>:
  ProcessDrag()

:<Key>osfHelp:
  Help()

The translations used by subclasses of XmLabel for menu traversal are described in the following list.

:KeyosfCancel:
  MenuEscape()

:KeyosfLeft:
  MenuTraverseLeft()

:KeyosfRight:
  MenuTraverseRight()

:KeyosfUp:
  MenuTraverseUp()

:KeyosfDown:
  MenuTraverseDown()

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. The direction of the wrapping depends on the **XmNlayoutDirection** resource.

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. The preceding description applies when the **XmNlayoutDirection** horizontal direction is **XmLEFT_TO_RIGHT**. If the **XmNlayoutDirection** horizontal direction is **XmRIGHT_TO_LEFT**, then the following applies.

If 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 a CascadeButton, then this action posts its associated submenu. If the current menu item is not a CascadeButton and is not at the left edge of a MenuPane, this action disarms the current item and arms the item to its left, wrapping if necessary. If the current menu item is not a CascadeButton and is at the left 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 left. 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 left edge of a row (except the bottom row), this action wraps to the rightmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, leftmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, rightmost menu item of 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. The preceding description applies when the **XmNlayoutDirection** horizontal direction is **XmLEFT_TO_RIGHT**. If the **XmNlayoutDirection** horizontal direction is **XmRIGHT_TO_LEFT**, then the following applies. 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 right edge of a MenuPane, this action disarms the current item and arms the item to its right. If the current menu item is at the right edge of a submenu attached to a MenuBar item, then this action unposts the submenu and traverses to the MenuBar item to the right, 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 right 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 right edge, this action wraps within the MenuPane. If the current menu item is at the right edge of the MenuPane and not in the top row, this action wraps to the leftmost menu item in the row above. If the current menu item is in the upper, rightmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, leftmost menu item in 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. The direction of the wrapping depends on the **XmNlayoutDirection** resource.

ProcessDrag():

Drags the contents of a Label, identified when **BTransfer** is pressed. This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures, possibly multiple times, for the _MOTIF_DROP selection. 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**(3).

Related Information

Core(3), **XmCreateLabel**(3), **XmFontListAppendEntry**(3), **XmStringCreate**(3), **XmStringCreateLtoR**(3), and **XmPrimitive**(3).

XmLabelGadget

Purpose

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 *Motif 2.1—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, depending on the layout direction.

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 menu panes, Pulldown menu panes, and OptionMenus. Otherwise these resources default to False.

LabelGadget uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits, and holds the *XmQTaccessTextual*, *XmQTcareParentVisual*, *XmQTmenuSavvy*, and *XmQTtransfer* traits.

Data Transfer Behavior

LabelGadget and it subclasses, except when used in a menu system, support dragging of the label contents from the LabelGadget. However, the label contents are draggable only if the **XmNenableUnselectableDrag** resource of **XmDisplay** is set to True.

As a source of data, LabelGadget and its subclasses support the following targets and associated conversions of data to these targets:

locale If the *locale* target matches the widget's locale, the widget transfers **XmNlabelString** in the encoding of the locale. This target is supported only when **XmNlabelType** is **XmSTRING**.

COMPOUND_TEXT
The widget transfers **XmNlabelString** as type *COMPOUND_TEXT*. This target is supported only when **XmNlabelType** is **XmSTRING**.

PIXMAP The widget transfers **XmNlabelPixmap** as type *DRAWABLE*. This target is supported only when **XmNlabelType** is **XmPIXMAP**.

STRING The widget transfers **XmNlabelString** as type *STRING*. This target is supported only when **XmNlabelType** is **XmSTRING**.

TEXT If **XmNlabelString** is fully convertible to the encoding of the locale, the widget transfers **XmNlabelString** in the encoding of the locale. Otherwise, the widget transfers **XmNlabelString** as type *COMPOUND_TEXT*. This target is supported only when **XmNlabelType** is **XmSTRING**.

_MOTIF_CLIPBOARD_TARGETS
The widget transfers, as type *ATOM*, a list of the targets it supports for the *CLIPBOARD* selection. When **XmNlabelType** is **XmSTRING**, these include the following targets:

- _MOTIF_COMPOUND_STRING
- *COMPOUND_TEXT*

- The encoding of the locale, if **XmNlabelString** is fully convertible to the encoding of the locale
- *STRING*, if **XmNlabelString** is fully convertible to *STRING*

When **XmNlabelType** is **XmPIXMAP**, the targets include *PIXMAP*.

_MOTIF_COMPOUND_STRING
: The widget transfers **XmNlabelString** as a compound string in Byte Stream format. This target is supported only when **XmNlabelType** is **XmSTRING**.

_MOTIF_EXPORT_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets to be used as the value of the DragContext's **XmNexportTargets** in a drag-and-drop transfer. When **XmNlabelType** is **XmSTRING**, these include _MOTIF_COMPOUND_STRING, *COMPOUND_TEXT*, the encoding of the locale, *STRING*, *TEXT*, *BACKGROUND*, and *FOREGROUND*. When **XmNlabelType** is **XmPIXMAP**, these include *PIXMAP*, *BACKGROUND*, and *FOREGROUND*.

As a source of data, LabelGadget also supports the following standard Motif targets:

BACKGROUND
: The widget transfers the parent's **XmNbackground** as type *PIXEL*.

CLASS
: The widget finds the first shell in the widget hierarchy that has a WM_CLASS property and transfers the contents as text in the current locale.

CLIENT_WINDOW
: The widget finds the first shell in the widget hierarchy and transfers its window as type *WINDOW*.

COLORMAP
: The widget transfers the parent's **XmNcolormap** as type *COLORMAP*.

FOREGROUND
: The widget transfers the parent's **XmNforeground** as type *PIXEL*.

NAME
: The widget finds the first shell in the widget hierarchy that has a WM_NAME property and transfers the contents as text in the current locale.

TARGETS
: The widget transfers, as type *ATOM*, a list of the targets it supports. These include the standard targets in this list. When **XmNlabelType**

is **XmSTRING**, these also include _MOTIF_COMPOUND_STRING, *COMPOUND_TEXT*, the encoding of the locale, *STRING*, and *TEXT*. When **XmNlabelType** is **XmPIXMAP**, these also include *PIXMAP*.

TIMESTAMP
The widget transfers the timestamp used to acquire the selection as type *INTEGER*.

_MOTIF_RENDER_TABLE
The widget transfers **XmNrenderTable** if it exists, or else the default text render table, as type *STRING*.

_MOTIF_ENCODING_REGISTRY
The widget transfers its encoding registry as type *STRING*. The value is a list of NULL separated items in the form of tag encoding pairs. This target symbolizes the transfer target for the Motif Segment Encoding Registry. Widgets and applications can use this Registry to register text encoding formats for specified render table tags. Applications access this Registry by calling **XmRegisterSegmentEncoding** and **XmMapSegmentEncoding**.

Classes

LabelGadget inherits behavior, resources, and traits from **Object**, **RectObj** and **XmGadget**.

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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	CSG
XmNacceleratorText	XmCAccelerator- Text	XmString	NULL	CSG
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive- Pixmap	XmCLabelInsensitive- Pixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	0	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	0	CSG
XmNmarginRight	XmCMarginRight	Dimension	0	CSG
XmNmarginTop	XmCMarginTop	Dimension	0	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonic- CharSet	XmCMnemonic- CharSet	String	dynamic	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	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. The direction of its layout depends on the **XmNlayoutDirection** resource of the widget. 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 **XmNlayoutDirection** is **XmLEFT_TO_RIGHT**. When that resource is **XmRIGHT_TO_LEFT**, 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 gadget is a subclass of **XmLabelGadget**. Otherwise, the default is **XmALIGNMENT_CENTER**.

XmNfontList

Specifies the font of the text used in the gadget. **XmNfontList** is obsolete and exists for compatibility with previous releases. You should now use **XmNrenderTable** instead of **XmNfontList**. If both are specified, the render table will take precedence. If **XmNfontList** 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**(3) 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**(3) 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 depends on the value of **XmNlayoutDirection**.

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 depends on the value of **XmNlayoutDirection**.

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 menu pane, or a Pulldown menu pane 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.

XmNrenderTable

Specifies the render table associated with the **labelString**. If this value is NULL at initialization, Label searches its parent hierarchy for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, Label initializes **XmNrenderTable** to the **XmLABEL_RENDER_TABLE** value of the ancestor widget. Similarly, button subclasses of Label initialize **XmNrenderTable** to the **XmBUTTON_RENDER_TABLE** value of the ancestor widget. (Note that all current subclasses of Label are button subclasses.) If no such ancestor is found, the default is implementation dependent. If a font list and a render table are both specified, the render table will take precedence. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

XmNstringDirection

Is a synthetic resource for setting **XmNlayoutDirection**. The values for this resource are **XmSTRING_DIRECTION_L_TO_R** and **XmSTRING_DIRECTION_R_TO_L**. Refer to the **XmNlayoutDirection** resource description. The **XmNstringDirection** resource is obsoleted by **XmNlayoutDirection**, but is kept here for backward compatibility.

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**	**Type**	**Default**	**Access**
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallback- List	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight- Thickness	Dimension	0	CSG
XmNlayoutDirection	XmNCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmNONE	CSG
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	False	CSG

XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

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

Behavior

XmLabelGadget includes behavior from XmGadget. Additional XmLabelGadget behavior is described in the following list:

Btn2Down: Drags the contents of a LabelGadget, identified when **BTransfer** is pressed. This action is undefined for LabelGadgets used in a menu system.

KeyosfHelp: 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.

keyosfCancel:
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.

KeyosfDown:

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. The direction of the wrapping depends on the **XmNlayoutDirection** resource.

KeyosfLeft: 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. The preceding description applies when the **XmNlayoutDirection** horizontal direction is **XmLEFT_TO_RIGHT**. If the **XmNlayoutDirection** horizontal direction is **XmRIGHT_TO_LEFT**, then the following applies.

If 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 a CascadeButton, then this action posts its associated submenu. If the current menu item is not a CascadeButton and is not at the left edge of a MenuPane, this action disarms the current item and arms the item to its left, wrapping if necessary. If the current menu item is not a CascadeButton and is at the left 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 left. 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 left edge of a row (except the bottom row), this action wraps to the rightmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, leftmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, rightmost menu item of the MenuPane.

KeyosfRight:

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. The preceding description applies when the **XmNlayoutDirection** horizontal direction is **XmLEFT_TO_RIGHT**. If the **XmNlayoutDirection** horizontal direction is **XmRIGHT_TO_LEFT**, then the following applies.

When 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 not at the right edge of a MenuPane, this action disarms the current item and arms the item to its right. If the current menu item is at the right edge of a submenu attached to a MenuBar item, then this action unposts the submenu and traverses to the MenuBar item to the right, 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 right 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 right edge, this action wraps within the MenuPane. If the current menu item is at the right edge of the MenuPane and not in the top row, this action wraps to the leftmost menu item in the row above. If the current menu item is in the upper, rightmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, leftmost menu item in the MenuPane.

KeyosfUp: 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. The direction of the wrapping depends on the **XmNlayoutDirection** resource.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Related Information

Object(3), **RectObj**(3), **XmCreateLabelGadget**(3), **XmFontListCreate**(3), **XmStringCreate**(3), **XmStringCreateLtoR**(3), and **XmGadget**(3).

XmList

Purpose

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 Btn1 mouse button or the key defined as osfSelect. 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.

List uses the *XmQTspecifyRenderTable*, *XmQTscrollFrame*, and *XmQTnavigator* traits, and holds the *XmQTtransfer* trait.

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 Btn1 on an item toggles its selection state and deselects any other selected item. In Browse Select, pressing Btn1 on an item selects it and deselects any other selected item; dragging Btn1 moves the selection as the pointer is moved. Releasing Btn1 on an item moves the location cursor to that item.

In Multiple Select, any number of items can be selected at a time. Pressing Btn1 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 Btn1 on an item selects it and deselects any other selected item. Dragging Btn1 or pressing or dragging ShiftBtn1 following a Btn1 action selects all items between the item under the pointer and the item on which Btn1 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 CtrlBtn1 on an item toggles its selection state but does not deselect any other selected items. Dragging CtrlBtn1 or pressing or dragging ShiftBtn1 following a CtrlBtn1 action sets the selection state of all items between the item under the pointer and the item on which CtrlBtn1 was pressed to the state of the item on which CtrlBtn1 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 osfSelect select the item at the location cursor and deselect any other selected items. In Add Mode, navigation operations have no effect on selection, and osfSelect 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 osfAddMode. In Extended Select Normal Mode, pressing osfSelect has the same effect as pressing Btn1; osfExtend and shifted navigation have the same effect as pressing ShiftBtn1 following a Btn1 action. In Extended Select Add Mode, pressing osfSelect has the same effect as pressing CtrlBtn1; osfExtend and shifted navigation have the same effect as pressing ShiftBtn1 following a CtrlBtn1 action.

Normal Mode is indicated by a solid location cursor, and Add Mode is indicated by a dashed location cursor.

Data Transfer Behavior

List supports dragging of items from the List and transfer of items to the clipboard. When the user presses **BTransfer** on a selected item, the widget transfers all selected items. When the user presses **BTransfer** on an unselected item, the widget transfers only that item. Depending on the value of **XmNprimaryOwnership**, List can also support primary selection.

When the **XmNconvertCallback** procedures are called, the **location_data** member of the **XmConvertCallbackStruct** member is NULL if the selected items are being transferred. If the selected items are not being transferred, this member has the following value: If a single item is being transferred, the value is an integer representing the position of the item in the List. A value of 1 transfers the first item in the List; a value of 2 transfers the second item; and so on. If the entire contents of the List are being transferred, the value is −1.

As a source of data, List supports the following targets and associated conversions of data to these targets:

locale If the *locale* target matches the widget's locale, the widget transfers the selected list items in the encoding of the locale. Each item transferred except the last includes a trailing separator.

COMPOUND_TEXT
The widget transfers the selected list items as type *COMPOUND_TEXT*. Each item transferred except the last includes a trailing separator.

STRING The widget transfers the selected list items as type *STRING*. Each item transferred except the last includes a trailing separator.

TEXT If the selected list items are fully convertible to the encoding of the locale, the widget transfers the selected list items in the encoding of the locale. Otherwise, the widget transfers the selected list items as type *COMPOUND_TEXT*. Each item transferred except the last includes a trailing separator.

_MOTIF_CLIPBOARD_TARGETS
The widget transfers, as type *ATOM*, a list of the targets it supports for immediate transfer for the *CLIPBOARD* selection. These include _MOTIF_COMPOUND_STRING. If the selected list items are fully convertible to *STRING*, these also include *STRING*; otherwise, they also include *COMPOUND_TEXT*.

_MOTIF_COMPOUND_STRING
: The widget transfers the selected list items as a compound string in Byte Stream format. Each item transferred except the last includes a trailing separator.

_MOTIF_DEFERRED_CLIPBOARD_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets it supports for delayed transfer for the *CLIPBOARD* selection. This widget currently supplies no targets for _MOTIF_DEFERRED_CLIPBOARD_TARGETS.

_MOTIF_EXPORT_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets to be used as the value of the DragContext's **XmNexportTargets** in a drag-and-drop transfer. These include _MOTIF_COMPOUND_STRING, *COMPOUND_TEXT*, the encoding of the locale, *STRING*, *TEXT*, *BACKGROUND*, and *FOREGROUND*.

_MOTIF_LOSE_SELECTION
: When the widget loses the selection, it deselects all list items.

As a source of data, List also supports the following standard Motif targets:

BACKGROUND
: The widget transfers **XmNbackground** as type *PIXEL*.

CLASS
: The widget finds the first shell in the widget hierarchy that has a WM_CLASS property and transfers the contents as text in the current locale.

CLIENT_WINDOW
: The widget finds the first shell in the widget hierarchy and transfers its window as type *WINDOW*.

COLORMAP
: The widget transfers **XmNcolormap** as type *COLORMAP*.

FOREGROUND
: The widget transfers **XmNforeground** as type *PIXEL*.

NAME
: The widget finds the first shell in the widget hierarchy that has a WM_NAME property and transfers the contents as text in the current locale.

TARGETS The widget transfers, as type *ATOM*, a list of the targets it supports. These include the standard targets in this list. These also include _MOTIF_COMPOUND_STRING, *COMPOUND_TEXT*, the encoding of the locale, *STRING*, and *TEXT*.

TIMESTAMP
The widget transfers the timestamp used to acquire the selection as type *INTEGER*.

_MOTIF_RENDER_TABLE
The widget transfers **XmNrenderTable** if it exists, or else the default text render table, as type *STRING*.

_MOTIF_ENCODING_REGISTRY
The widget transfers its encoding registry as type *STRING*. The value is a list of NULL separated items in the form of tag encoding pairs. This target symbolizes the transfer target for the Motif Segment Encoding Registry. Widgets and applications can use this Registry to register text encoding formats for specified render table tags. Applications access this Registry by calling **XmRegisterSegmentEncoding** and **XmMapSegmentEncoding**.

List has no widget class destination procedure. Subclasses and the **XmNdestinationCallback** procedures are responsible for any data transfers to the widget.

Classes

List inherits behavior, resources, and traits from **Core** and **XmPrimitive**.

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**	**Type**	**Default**	**Access**
XmNautomatic- Selection	XmCAutomatic- Selection	XtEnum	False	CSG
XmNbrowseSelection- Callback	XmCCallback	XtCallbackList	NULL	C
XmNdefaultAction- Callback	XmCCallback	XtCallbackList	NULL	C
XmNdestination- Callback	XmCCallback	XtCallbackList	NULL	C
XmNdouble- ClickInterval	XmCDoubleClick- Interval	int	dynamic	CSG
XmNextendedSelection- Callback	XmCCallback	XtCallbackList	NULL	C
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNitemCount	XmCItemCount	int	0	CSG
XmNitems	XmCItems	XmStringTable	NULL	CSG
XmNlistMarginHeight	XmCList- MarginHeight	Dimension	0	CSG
XmNlistMarginWidth	XmCList- MarginWidth	Dimension	0	CSG
XmNlistSizePolicy	XmCListSizePolicy	unsigned char	XmVARIABLE	CG
XmNlistSpacing	XmCListSpacing	Dimension	0	CSG
XmNmatchBehavior	XmCMatchBehavior	unsigned char	XmQUICK_- NAVIGATE	CSG
XmNmultipleSelection- Callback	XmCCallback	XtCallbackList	NULL	C
XmNprimaryOwnership	XmCPrimary- Ownership	unsigned char	XmOWN_NEVER	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNscrollBarDisplay- Policy	XmCScrollBar- DisplayPolicy	unsigned char	XmAS_NEEDED	CSG
XmNselectColor	XmCSelectColor	XmRSelectColor	XmREVERSED_- GROUND_COLORS	CSG
XmNselectedItemCount	XmCSelected- ItemCount	int	0	CSG

XmNselectedItems	XmCSelectedItems	XmStringTable	NULL	CSG
XmNselectedPosition-Count	XmCSelected-PositionCount	int	0	CSG
XmNselectedPositions	XmCSelected- Positions	unsigned int *	NULL	CSG
XmNselectionMode	XmCSelectionMode	unsigned char	dynamic	CSG
XmNselectionPolicy	XmCSelectionPolicy	unsigned char	XmBROWSE_-SELECT	CSG
XmNsingleSelection-Callback	XmCCallback	XtCallbackList	NULL	C
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CSG
XmNtopItemPosition	XmCTopItemPosition	int	1	CSG
XmNvisibleItemCount	XmCVisibleI- temCount	int	dynamic	CSG

XmNautomaticSelection
Invokes either **XmNbrowseSelectionCallback** or **XmNextendedSelectionCallback** when Btn1 is pressed and the items that are shown as selected change if the value is True (or **XmAUTO**) and the selection mode is either **XmBROWSE_SELECT** or **XmEXTENDED_SELECT** respectively. If False (**XmNO_AUTO_SELECT**), 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 osfActivate is pressed. The reason is **XmCR_DEFAULT_ACTION**.

XmNdestinationCallback
Specifies a list of callbacks called when the List is the destination of a transfer operation. The type of the structure whose address is passed to these callbacks is **XmDestinationCallbackStruct**. The reason is **XmCR_OK**.

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. **XmNfontList** is obsolete and exists only for compatibility with previous releases. You should now use **XmNrenderTable** instead of **XmNfontList**. If both are specified, the render table will take precedence. The font list 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 a widget that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the font list is initialized to the **XmTEXT_RENDER_TABLE** value of the ancestor widget. If no such widget is found, the default is implementation dependent. Refer to **XmFontList**(3) 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**(3) 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 List whose parent is a ScrolledWindow. 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 List whose parent is a ScrolledWindow 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.

XmNmatchBehavior

Specifies the matching behavior followed by XmList. The current values are **XmNONE** and **XmQUICK_NAVIGATE**, as follows:

XmNONE Specifies that the typed in characters are ignored.

XmQUICK_NAVIGATE

Specifies that 1-character navigation shall be supported when List has focus. If the typed character is the initial character of some set of items in List, the first of those items following the current item will be navigated to (become the current item). If all such items precede the current item, the first such item becomes the current item. Subsequently, typing the same character will cyclically navigate among the items with the same first character.

XmNmultipleSelectionCallback

Specifies a list of callbacks that is called when an item is selected in multiple selection mode. The reason is **XmCR_MULTIPLE_SELECT**.

XmNprimaryOwnership

Specifies whether XmContainer takes ownership of the Primary selection when a selection is made inside it. This resource can take the following values:

XmOWN_NEVER
Never takes ownership.

XmOWN_ALWAYS
Always takes ownership.

XmOWN_MULTIPLE
Only takes ownership is more than one element has been selected.

XmOWN_POSSIBLE_MULTIPLE
Only takes ownership if more than one element can be selected at a time.

XmNrenderTable
Specifies the render table associated with the list items. The render table is used in conjunction with the **XmNvisibleItemCount** resource to determine the height of the List widget. If this value is NULL at initialization, List searches its parent hierarchy for a widget that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the render table is initialized to the **XmTEXT_RENDER_TABLE** value of the ancestor widget. If no such widget is found, the default is implementation dependent. If a font list and a render table are both specified, the render table will take precedence. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

XmNscrollBarDisplayPolicy
Controls the display of vertical ScrollBars in a List whose parent is a ScrolledWindow. 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.

XmNselectColor
Allows the application to specify the color of the background rectangle that indicates selected text. It takes two values:

XmDEFAULT_SELECT_COLOR
Causes the select color to be set to a color between the background and the bottom shadow color.

XmREVERSED_GROUND_COLORS
Forces the select color to the foreground color and the color of any text rendered over the select color to be in the background color.

HIGHLIGHT_COLOR
Forces the fill color to use the highlight color.

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 or the array.

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 **XmListDeleteAllItems**, **XmListDeleteItem**, **XmListDeleteItems**, **XmListDeleteItemsPos**, **XmListDeletePos**, **XmListDeletePositions**, **XmListDeselectAllItems**, **XmListDeselectItem**, **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**.

XmNselectedPositionCount
Specifies the number of positions in the selected positions list. The value must be the number of items in **XmNselectedPositions**

XmNselectedPositions

Points to an array of the positions of the selected items in the 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 or the array.

XmNselectionMode

Defines what effect keyboard navigations have on selection. The valid modes are:

XmADD_MODE

Allows no navigation operations to have effect on selection, and osfSelect toggles the selection state of the item at the location cursor without deselecting any other selected items, except in Single Select. However, the widget cannot be put into add mode if the **XmNselectionPolicy** resource is an incompatible mode (**XmNselectionPolicy** cannot be **XmBROWSE_SELECT**).

XmNORMAL_MODE

Allows navigation operations and osfSelect to select the item at the location cursor and deselect any other selected items. However, the widget cannot be put into normal mode if the **XmNselectionPolicy** resource is an incompatible mode (**XmNselectionPolicy** cannot be **XmSINGLE_SELECT** or **XmMULTIPLE_SELECT**).

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

Is a synthetic resource for setting **XmNlayoutDirection**. The values for this resource are **XmSTRING_DIRECTION_L_TO_R** and **XmSTRING_DIRECTION_R_TO_L**. Refer to the **XmNlayoutDirection** resource description. The **XmNstringDirection** resource is obsoleted by **XmNlayoutDirection**, but is kept here for backward compatibility.

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**	**Type**	**Default**	**Access**
XmNbottom-ShadowColor	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottom-ShadowPixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvert- Callback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C

XmNhighlightColor	XmCHighlight-Color	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight-OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlight-Pixmap	Pixmap	dynamic	CSG
XmNhighlight- Thickness	XmCHighlight-Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayout-Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigation-Type	XmNavigationType	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadow-Thickness	Dimension	2	CSG
XmNtopShadowColor	XmCTop-ShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTop-ShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestor- Sensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground-Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefault-Foreground	CSG

XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitial-ResourcesPersistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMapped-WhenManaged	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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;
        unsigned char auto_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*. This member is obsolete and exists for compatibility with earlier releases.

item_position
The position (plus one) of *item* in the List's **XmNitems** array. An *item_position* value of one symbolizes the first element in the list.

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 non-negative.

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**).

auto_selection_type
Indicates the type of automatic selection callback. The types of callbacks include the following:

XmAUTO_BEGIN
Indicates the beginning of automatic selection.

XmAUTO_MOTION
Indicates that there is a button drag selection.

XmAUTO_CANCEL
Indicates that the new selection is cancelled.

XmAUTO_NO_CHANGE
Indicates that the currently selected item matches the initial item.

XmAUTO_CHANGE
Indicates that the currently selected item does not match the initial item.

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*

A pointer to the following callback structure is passed to the **XmNdestinationCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent *event;
        Atom selection;
        XtEnum operation;
        int flags;
        XtPointer transfer_id;
        XtPointer destination_data;
        XtPointer location_data;
        Time time;
} XmDestinationCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which data transfer is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

operation Indicates the type of transfer operation requested.

- When the selection is *PRIMARY*, possible values are **XmMOVE**, **XmCOPY**, and **XmLINK**.
- When the selection is *SECONDARY* or *CLIPBOARD*, possible values are **XmCOPY** and **XmLINK**.
- When the selection is _MOTIF_DROP, possible values are **XmMOVE**, **XmCOPY**, **XmLINK**, and **XmOTHER**. A value of **XmOTHER** means that the callback procedure must get further information from the **XmDropProcCallbackStruct** in the *destination_data* member.

flags Indicates whether or not the destination widget is also the source of the data to be transferred. Following are the possible values:

XmCONVERTING_NONE
The destination widget is not the source of the data to be transferred.

XmCONVERTING_SAME
The destination widget is the source of the data to be transferred.

transfer_id Serves as a unique ID to identify the transfer transaction.

destination_data
Contains information about the destination. When the selection is _MOTIF_DROP, the callback procedures are called by the drop site's **XmNdropProc**, and *destination_data* is a pointer to the **XmDropProcCallbackStruct** passed to the **XmNdropProc** procedure. When the selection is *SECONDARY*, *destination_data* is an Atom representing a target recommmended by the selection owner for use in converting the selection. Otherwise, *destination_data* is NULL.

location_data
Contains information about the location where data is to be transferred. The value is always NULL when the selection is *SECONDARY* or *CLIPBOARD*. If the value is NULL, the data is to be inserted at the widget's cursor position. Otherwise, the value is an integer representing the position in the List where the items are to be transferred. 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. Once *XmTransferDone* procedures start to be called, **location_data** will no longer be stable.

time Indicates the time when the transfer operation began.

Translations

XmList includes translations from Primitive. The **XmList** translations are described in the following list.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<Btn1>Motion:
ListButtonMotion()

s ≈m ≈a <Btn1Down>:
ListBeginExtend()

s ≈m ≈a <Btn1Up>:
ListEndExtend()

c ≈s ≈m ≈a <Btn1Down>:
ListBeginToggle()

c ≈s ≈m ≈a <Btn1Up>:
ListEndToggle()

≈s ≈c ≈m ≈a <Btn1Down>:
ListBeginSelect()

≈s ≈c ≈m ≈a <Btn1Up>:
ListEndSelect()

<Btn2Down>:
ListProcessDrag()

:s c <Key>osfBeginLine:
ListBeginDataExtend()

:c <Key>osfBeginLine:
ListBeginData()

:<Key>osfBeginLine:
ListBeginLine()

:s c <Key>osfEndLine:
ListEndDataExtend()

:c <Key>osfEndLine:
ListEndData()

:<Key>osfEndLine:
ListEndLine()

:<Key>osfPageLeft:
ListLeftPage()

:c <Key>osfPageUp:
ListLeftPage()

:<Key>osfPageUp:
ListPrevPage()

:<Key>osfPageRight:
ListRightPage()

:c <Key>osfPageDown:
ListRightPage()

:<Key>osfPageDown:
ListNextPage()

s <KeyDownosfSelect:
    ListKbdBeginExtend()

:<KeyDownosfSelect:
    ListKbdBeginSelect()

:s <KeyUposfSelect:
    ListKbdEndExtend()

:<KeyUposfSelect:
    ListKbdEndSelect()

:<Key>osfSelectAll:
    ListKbdSelectAll()

:<Key>osfDeselectAll:
    ListKbdDeSelectAll()

:<Key>osfActivate:
    ListKbdActivate()

:<Key>osfAddMode:
    ListAddMode()

:<Key>osfHelp:
    PrimitiveHelp()

:<Key>osfCancel:
    ListKbdCancel()

:c <Key>osfLeft:
    ListLeftPage()

:<Key>osfLeft:
    ListLeftChar()

:c <Key>osfRight:
    ListRightPage()

:<Key>osfRight:
    ListRightChar()

:s <Key>osfUp:
    ListExtendPrevItem()

:<Key>osfUp:
    ListPrevItem()

:s <Key>osfDown:
ListExtendNextItem()

:<Key>osfDown:
ListNextItem()

:c <Key>osfInsert:
ListCopyToClipboard()

:<Key>osfCopy:
ListCopyToClipboard()

≈s c ≈m ≈a <Key>slash:
ListKbdSelectAll()

≈s c ≈m ≈a <Key>backslash:
ListKbdDeSelectAll()

s ≈m ≈a <Key>Tab:
PrimitivePrevTabGroup()

≈m ≈a <Key>Tab:
PrimitiveNextTabGroup()

≈s ≈m ≈a <Key>Return:
ListKbdActivate()

≈s ≈m ≈a <KeyDownspace:
ListKbdBeginSelect()

≈s ≈m ≈a <KeyUpspace:
ListKbdEndSelect()

s ≈m ≈a <KeyDownspace:
ListKbdBeginExtend()

s ≈m ≈a <KeyUpspace:
ListKbdEndExtend()

<Key>: **ListQuickNavigate()**

The List button event translations are modified when Display's **XmNenableBtn1Transfer** resource does not have a value of **XmOFF** (in other words, it is either *XmBUTTON2_TRANSFER* or *XmBUTTON2_ADJUST*). This option allows the actions for selection and transfer to be integrated on Btn1, and the actions for extending the selection can be bound to Btn2. The actions for Btn1 that are defined above still apply when the Btn1 event occurs over text that is not selected. The following actions apply when the Btn1 event occurs over text that is selected:

<Btn1Motion>:
ListProcessBtn1(*ListButtonMotion***)**

s ≈m ≈a <Btn1Down>
ListProcessBtn1(*ListBeginExtend***)**

s ≈m ≈a <Btn1Up>
ListProcessBtn1(*ListEndExtend***)**

c ≈s ≈m ≈a <Btn1Down>
ListProcessBtn1(*ListBeginToggle***)**

c ≈s ≈m ≈a <Btn1Up>
ListProcessBtn1(*ListEndToggle***)**

≈s ≈c ≈m ≈a <Btn1Down>
ListProcessBtn1(*ListBeginSelect***)**

≈s ≈c ≈m ≈a <Btn1Up>
ListProcessBtn1(*ListEndSelect***)**

When Display’s **XmNenableBtn1Transfer** resource has a value of *XmBUTTON2_ADJUST*, the following actions apply:

<Btn2Down>:
ListProcessBtn2(*ListBeginExtend***)**

<Btn2Motion>:
ListProcessBtn2(*ListButtonMotion***)**

<Btn2Up>: **ListProcessBtn2(***ListEndExtend***)**

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. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_BEGIN**.

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, calls the **XmNbrowseSelectionCallback** callbacks. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_BEGIN**.

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. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_BEGIN**.

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.

Otherwise, the list takes keyboard focus. No other action occurs. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_BEGIN**.

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. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_MOTION**.

ListCopyToClipboard()
: Copies the content of the selected items to the clipboard as a single compound string with each item separated by a newline. This action calls the **XmNconvertCallback** procedures, possibly multiple times, for the *CLIPBOARD* selection.

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 calls the **XmNextendedSelectionCallback** callbacks.

  If **XmNautomaticSelection** is set to True, then the **auto_selection_type** field of the callback will be valid. If **XmNselectionPolicy** is set to **XmBROWSE_SELECT** and the currently selected item position matches the position of the item that was selected before the browse selection began, or if **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** and the set of currently selected item positions matches the set of item positions selected before the extended selection began, the callback will be called with **auto_selection_type** set to **XmAUTO_NO_CHANGE**. Otherwise, it will be set to **XmAUTO_CHANGE**.

ListEndLine():

Moves the horizontal scroll region to the end of the line.

ListEndSelect():

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**, **XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to **XmBROWSE_SELECT**, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to **XmMULTIPLE_SELECT**, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**).

If **XmNautomaticSelection** is set to True, then the **auto_selection_type** field of the callback will be valid. If **XmNselectionPolicy** is set to **XmBROWSE_SELECT** and the currently selected item position matches the position of the item that was selected before the brose selection began, or if **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** and the set of currently selected item positions matches the set of item positions selected before the extended selection began, the callback will be called with **auto_selection_type** set to **XmAUTO_NO_CHANGE**. Otherwise, it will be set to **XmAUTO_CHANGE**.

ListEndToggle():

If **XmNselectionPolicy** is set to **XmEXTENDED_SELECT**, moves the location cursor to the last item selected or deselected and calls the **XmNextendedSelectionCallback** callbacks.

If **XmNautomaticSelection** is set to True, then the **auto_selection_type** field of the callback will be valid. If **XmNselectionPolicy** is set to **XmBROWSE_SELECT** and the currently selected item position matches the position of the item that was selected before the browse selection began, or if **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** and the set of currently selected item positions matches the set of item positions selected before the extended selection began, the callback will be called with **auto_selection_type** set to **XmAUTO_NO_CHANGE**. Otherwise, it will be set to **XmAUTO_CHANGE**.

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. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_BEGIN**".

ListKbdBeginSelect():

If the **XmNselectionPolicy** is set to **XmSINGLE_SELECT**, deselects any current selection and toggles the state of the item at the location cursor. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_BEGIN**".

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, 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. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_BEGIN**".

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. The **auto_selection_type** component of the callback structure will be set to **XmAUTO_CANCEL**".

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**, this action calls the **XmNextendedSelectionCallback** callbacks.

If **XmNautomaticSelection** is set to True, then the **auto_selection_type** field of the callback will be valid. If **XmNselectionPolicy** is set to **XmBROWSE_SELECT** and the currently selected item position matches the position of the item that was selected before the browse selection began, or if **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** and the set of currently selected item positions matches the set of item positions selected before the extended selection began, the callback will be called with **auto_selection_type** set to **XmAUTO_NO_CHANGE**. Otherwise, it will be set to **XmAUTO_CHANGE**.

ListKbdEndSelect():

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**).

If **XmNautomaticSelection** is set to True, then the **auto_selection_type** field of the callback will be valid. If **XmNselectionPolicy** is set to **XmBROWSE_SELECT** and the currently selected item position matches the position of the item that was selected before the browse selection began, or if **XmNselectionPolicy** is set to **XmEXTENDED_SELECT** and the set of currently selected item positions matches the set of item positions selected before the extended selection began, the callback will be called with **auto_selection_type** set to **XmAUTO_NO_CHANGE**. Otherwise, it will be set to **XmAUTO_CHANGE**.

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.

ListProcessBtn1(*string*)
When Display’s **XmNenableBtn1Transfer** resource is not **XmOFF**, the actions for selection and transfer are integrated on Btn1. When the button is not performing a transfer or drag, the action that is performed depends on the value of *string*, which can be one of the following actions:

- **ListButtonMotion**
- **ListBeginExtend**
- **ListEndExtend**

- **ListBeginToggle**
- **ListEndToggle**
- **ListBeginSelect**
- **ListEndSelect**

ListProcessBtn2()
When Display's **XmNenableBtn1Transfer** resource has a value of *XmBUTTON2_TRANSFER*, the actions for extending selection are bound on Btn2, and a drag starts immediately. When Display's **XmNenableBtn1Transfer** resource has a value of *XmBUTTON2_ADJUST*, the action that is performed depends on the value of *string*, which can be one of the following actions:

- **ListBeginExtend**
- **ListButtonMotion**
- **ListEndExtend**

ListProcessDrag():
Drags the content of one or more selected list items. Each item is separated by a newline. If **BTransfer** is pressed on an unselected item, it drags only that item, excluding any other selected items. This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures, possibly multiple times, for the _MOTIF_DROP selection.

ListQuickNavigate()
Navigates to an item. When List's **XmNmatchBehavior** resource is **XmQUICK_NAVIGATE**, this action uses 1-character navigation to navigate. Refer to the **XmNmatchBehavior** resource for a description of how this navigation works.

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**(3).

Related Information

Core(3), **XmCreateList**(3), **XmCreateScrolledList**(3), **XmFontListCreate**(3), **XmFontListAppendEntry**(3), **XmListAddItem**(3), **XmListAddItems**(3), **XmListAddItemUnselected**(3), **XmListAddItemsUnselected**(3), **XmListDeleteAllItems**(3), **XmListDeleteItem**(3), **XmListDeleteItems**(3), **XmListDeleteItemsPos**(3), **XmListDeletePos**(3), **XmListDeletePositions**(3), **XmListDeselectAllItems**(3), **XmListDeselectItem**(3), **XmListDeselectPos**(3), *XmListGetKbdItemPos* **XmListGetMatchPos**(3), **XmListGetSelectedPos**(3), **XmListItemExists**(3), **XmListItemPos**(3), **XmListPosToBounds**(3),

XmListReplaceItems(3), **XmListReplaceItemsPos**(3), **XmListReplaceItemsPos**(3), **XmListReplaceItemsPosUnselected**(3), **XmListReplaceItemsUnselected**(3), **XmListSelectItem**(3), **XmListSelectPos**(3), **XmListSetAddMode**(3), **XmListSetBottomItem**(3), **XmListSetBottomPos**(3), **XmListSetHorizPos**(3), **XmListSetItem**(3), **XmListSetKbdItemPos**(3), **XmListSetPos**(3), **XmListUpdateSelectedList**(3), **XmListYToPos**(3), **XmPrimitive**(3) and **XmStringCreate**(3).

XmMainWindow

Purpose

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**.

MainWindow also provides the following three child types for layout at creation time:

- **XmMENU_BAR**
- **XmCOMMAND_WINDOW**

- **XmMESSAGE_WINDOW**

MainWindow can use these child types at creation time instead of their associated resource values. MainWindow uses the *XmQTmenuSystem* trait.

Descendants

MainWindow automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
=		
HorScrollBar	**XmScrollBar**	horizontal scroll bar
Separator1	**XmSeparatorGadget**	optional first separator
Separator2	**XmSeparatorGadget**	optional second separator
Separator3	**XmSeparatorGadget**	optional third separator
VertScrollBar	**XmScrollBar**	vertical scroll bar

Classes

MainWindow inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XmScrolledWindow**.

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**	**Type**	**Default**	**Access**
XmNcommandWindow	XmCCommandWindow	Widget	NULL	CSG
XmNcommandWindow-Location	XmCCommandWindow-Location	unsigned char	ABOVE (SeeDesc.)	CG
XmNmainWindow-MarginHeight	XmCMainWindow-MarginHeight	Dimension	0	CSG
XmNmainWindow-MarginWidth	XmCMainWindow-MarginWidth	Dimension	0	CSG
XmNmenuBar	XmCMenuBar	Widget	NULL	CSG
XmNmessageWindow	XmCMessageWindow	Widget	NULL	CSG
XmNshowSeparator	XmCShowSeparator	Boolean	False	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**	**Type**	**Default**	**Access**
XmNautoDrag- Model	XmCAutoDragModel	XtEnum	XmAUTO_DRAG_ ENABLED	CSG
XmNclipWindow	XmCClipWindow	Widget	dynamic	G
XmNhorizontal- ScrollBar	XmCHorizontal- ScrollBar	Widget	dynamic	CSG
XmNscrollBarDisplay- Policy	XmCScrollBar- DisplayPolicy	unsigned char	dynamic	CSG
XmNscrollBar- Placement	XmCScrollBar- Placement	unsigned char	XmBOTTOM_- RIGHT	CSG
XmNscrolledWindow- MarginHeight	XmCScrolledWindow- MarginHeight	Dimension	0	N/A
XmNscrolledWindow- MarginWidth	XmCScrolledWindow- MarginWidth	Dimension	0	N/A
XmNscrolling- Policy	XmCScrollingPolicy	unsigned char	XmAPPLICATION_- DEFINED	CG
XmNspacing	XmCSpacing	Dimension	4	CSG
XmNtraverseObscured- Callback	XmCCallback	XtCallback- List	NULL	CSG
XmNverticalScrollBar	XmCVerticalScrollBar	Widget	dynamic	CSG

XmNvisualPolicy	XmCVisualPolicy	unsigned char	dynamic	G
XmNworkWindow	XmCWorkWindow	Widget	NULL	CSG

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlight- Pixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	XmCReadOnly	WidgetList	NULL	G

XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

MainWindow inherits translations from ScrolledWindow.

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateMainWindow**(3), **XmMainWindowSep1**(3), **XmMainWindowSep2**(3), **XmMainWindowSep3**(3), **XmMainWindowSetAreas**(3), **XmManager**(3), and **XmScrolledWindow**(3)

XmManager

Purpose

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**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG

XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlight- Color	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlight-Pixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayout-Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigation-Type	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadow-Thickness	Dimension	0	CSG
XmNstringDirection	XmCString-Direction	XmString- Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadow-Color	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	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.

XmNlayoutDirection

Specifies the direction in which components of the manager (including strings) are laid out. The values are of type **XmDirection**. If the widget's parent is a manager or shell, the value is inherited from the widget's parent. Otherwise, it is inherited from the closest ancestor vendor or menu shell. Refer to the **XmDirection**(3) reference page for the possible direction values.

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**.

XmNpopupHandlerCallback
Allows the application to control which popup menu will be automatically posted. The reason can either be **XmCR_POST** or **XmCR_REPOST:**

XmCR_POST
Indicates that this is a regular posting request.

XmCR_REPOST
Indicates that the menu was just unposted and that this callback was invoked on a replay.

This callback uses the **XmPopupHandlerCallbackStruct** structure to pass information.

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
Is a synthetic resource for setting **XmNlayoutDirection**. The values for this resource are **XmSTRING_DIRECTION_L_TO_R** and **XmSTRING_DIRECTION_R_TO_L**. Refer to the **XmNlayoutDirection** resource description. The **XmNstringDirection** resource is obsoleted by **XmNlayoutDirection**, but is kept here for backward compatibility.

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**.

The unit type can also be specified in resource files, with the following format:

```
<floating value><unit>
```

where:

unit	is <" ", pixels, inches, centimeters, millimeters, points, font units>
pixels	is <*pix*, *pixel*, *pixels*>
inches	is <*in*, *inch*, *inches*>
centimeter	is <*cm*, *centimeter*, *centimeters*>
millimeters	is <*mm*, *millimeter*, *millimeters*>

points is <*pt*, *point*, *points*>

font units is <*fu*, *font_unit*, *font_units*>

float is {"+"|"-"}{{<"0"-"9">*}.}<"0"-"9">*

Note that the type Dimension must always be positive.

For example,

```
xmfonts*XmMainWindow.height: 10.4cm
*PostIn.width: 3inches
```

XmNunitType can have the following values:

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

XmMILLIMETERS
All values provided to the widget are treated as normal millimeter values.

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

XmCENTIMETERS
All values provided to the widget are treated as normal centimeter values.

XmINCHES
All values provided to the widget are treated as normal inch values.

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

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

XmFONT_UNITS
All values provided to the widget are treated as normal font units. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

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 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**	**Type**	**Default**	**Access**
XmNchildren	XmCReadOnly	WidgetList	NULL	G
XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestor- Sensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefault- Foreground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroy- Callback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback Information

A pointer to the following structure is passed to each callback for **XmNhelpCallback**:

```
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.

A pointer to the following structure is passed to each callback for **XmNpopupHandlerCallback**:

```
typedef struct
{
        int reason;
        XEvent * xevent;
        Widget menuToPost;
        Boolean postIt;
        Widget target;
} XmPopupHandlerCallbackStruct;
```

reason	Indicates why the callback was invoked.
xevent	Points to the *XEvent* that triggered the handler.
menuToPost	Specifies the popup menu that the menu system believes should be posted. The application may modify this field.
postIt	Indicates whether the posting process should continue. The application may modify this field.
target	Specifies the most specific widget or gadget that the menu sytem found from the event that matches the event.

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.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<BtnMotion>:
ManagerGadgetButtonMotion()

c<Btn1Down>:
ManagerGadgetTraverseCurrent()

≈c<Btn1Down>:
ManagerGadgetArm()

≈c<Btn1Down> ,≈c<Btn1Up>:
ManagerGadgetActivate()

≈c<Btn1Up>:
ManagerGadgetActivate()

≈c<Btn1Down> (2+):
ManagerGadgetMultiArm()

≈c<Btn1Up> (2+):
ManagerGadgetMultiActivate()

<Btn2Down>:
ManagerGadgetDrag()

:<Key>osfActivate:
ManagerParentActivate()

:<Key>osfCancel:
ManagerParentCancel()

:<Key>osfSelect:
ManagerGadgetSelect()

:<Key>osfHelp:
ManagerGadgetHelp()

≈s ≈m ≈a <Key>Return:
ManagerParentActivate()

≈s ≈m ≈a <Key>space:
ManagerGadgetSelect()

<Key>: **ManagerGadgetKeyInput()**

:<Key>osfBeginLine:
ManagerGadgetTraverseHome()

:<Key>osfUp:
ManagerGadgetTraverseUp()

:<Key>osfDown:
ManagerGadgetTraverseDown()

:<Key>osfLeft:
ManagerGadgetTraverseLeft()

:<Key>osfRight:
ManagerGadgetTraverseRight()

s ≈m ≈a <Key>Tab:
ManagerGadgetPrevTabGroup()

≈m ≈a <Key>Tab:
ManagerGadgetNextTabGroup()

Action Routines

The **XmManager** action routines are

GadgetTakeFocus():
Causes the current gadget to take keyboard focus when **Ctrl<Btn1Down>** is pressed.

ManagerGadgetActivate():
Causes the current gadget to be activated.

ManagerGadgetArm():
Causes the current gadget to be armed.

ManagerGadgetButtonMotion():
Causes the current gadget to process a mouse motion event.

ManagerGadgetDrag():
Causes the current gadget to begin a drag operation. 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():

This action depends on the value of the Display resource **XmNenableButtonTab**. When **XmNenableButtonTab** is False (default), this action 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.

When **XmNenableButtonTab** is True, this action moves to the next item within the current tab group, unless it is the last item in the tab group. When the item is the last in the group, the action traverses to the first item in the next tab group. The **XmNenableButtonTab** behavior applies only to PushButton, ArrowButton, and DrawnArrow.

ManagerGadgetPrevTabGroup():

This action depends on the value of the Display resource **XmNenableButtonTab**. When **XmNenableButtonTab** is False (default), this action 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.

When **XmNenableButtonTab** is True, this action moves to the previous item within the current tab group unless it is the first item in the tab group. When the item is the first in the group, the action traverses to the first item in the previous tab group. The **XmNenableButtonTab** behavior applies only PushButton, ArrowButton, and DrawnButton.

ManagerGadgetSelect():

Causes the current gadget to be armed and activated.

ManagerGadgetTraverseCurrent()
Causes the current gadget to take keyboard focus when **Ctrl<Btn1Down>** is pressed. Gadget is not activated.

ManagerGadgetTraverseDown():
Traverses to the next item below the current gadget in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

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. The wrapping direction depends on the layout direction of the widget tab group.

ManagerGadgetTraverseNext():
Traverses to the next item in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

ManagerGadgetTraversePrev():
Traverses to the previous item in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

ManagerGadgetTraverseRight()
Traverses to the next item to the right of the current gadget in the current tab, wrapping if necessary. widget tab group. The wrapping direction depends on the layout direction of the widget tab group.

ManagerGadgetTraverseUp():
Traverses to the next item above the current gadget in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

ManagerParentActivate():
If the parent is a manager, passes the osfActivate event received by the current widget/gadget to its parent.

ManagerParentCancel():
If the parent is a manager, passes the osfCancel event received by the current widget/gadget to its parent.

Additional Behavior

This widget has the additional behavior described below:

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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmDirection**(3), **XmChangeColor**(3), **XmGadget**(3), and **XmScreen**(3).

XmMenuShell

Purpose

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 menu panes.

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 menu pane. The convenience functions automatically create a MenuShell widget as the parent of the menu pane. 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 menu pane, the MenuShell's parent must be the widget from which the Popup menu pane is popped up.
- If the MenuShell is for a menu pane that is pulled down from a Popup or another Pulldown menu pane, the MenuShell's parent must be the Popup or Pulldown menu pane.
- If the MenuShell is for a menu pane that is pulled down from a MenuBar, the MenuShell's parent must be the MenuBar.
- If the MenuShell is for a Pulldown menu pane 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**.

MenuShell uses the *XmQTmenuSystem* trait and holds the *XmQTspecifyRenderTable* trait.

Classes

MenuShell inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	NULL	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	XmLEFT_TO_RIGHT	CG

XmNbuttonFontList
Specifies the font list used for button descendants. See the **XmNbuttonRenderTable** resource.

XmNbuttonRenderTable
Specifies the render table used for MenuShell’s button descendants. If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNbuttonRenderTable** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, **XmNbuttonRenderTable** is initialized by looking up the parent hierarchy of the widget for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, **XmNbuttonRenderTable** is initialized to the **XmBUTTON_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

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 label descendants. See the **XmNlabelRenderTable** resource.

XmNlabelRenderTable
Specifies the render table 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, **XmNlabelRenderTable** 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 holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, **XmNlabelRenderTable** is initialized to the **XmLABEL_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

XmNlayoutDirection
Specifies the direction in which the subwidgets, children of a widget, or other visual components are to be laid out. This policy will apply as the default layout policy for all descendants of this MenuShell.

Inherited Resources

MenuShell 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. 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**	**Type**	**Default**	**Access**
XmNallowShell- Resize	XmCAllowShell- Resize	Boolean	True	G
XmNcreatePopup- ChildProc	XmCCreatePopup- ChildProc	XtCreatePopup- ChildProc	NULL	CSG
XmNgeometry	XmCGeometry	String	NULL	CSG
XmNoverrideRedirect	XmCOverride- Redirect	Boolean	True	CSG
XmNpopdown- Callback	XmCCallback	XtCallbackList	NULL	C
XmNpopupCallback	XmCCallback	XtCallbackList	NULL	C
XmNsaveUnder	XmCSaveUnder	Boolean	True	CSG
XmNvisual	XmCVisual	Visual *	CopyFrom- Parent	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

The XmMenuShell translations are described in the following list.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<BtnDown>:
: **ClearTraversal()**

<BtnUp>:
: **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 **XmEXPLICIT**, 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 **XmEXPLICIT**, 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**(3).

Related Information

Composite(3), **Core**(3), **OverrideShell**(3), **Shell**(3), **XmCreateMenuShell**(3), **XmCreatePopupMenu**(3), **XmCreatePulldownMenu**(3), and **XmRowColumn**(3).

XmMessageBox

Purpose

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 BulletinBoard 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. The *XmQTmenuSystem* trait is used to check that it is the first MenuBar child.

- All widgets or gadgets are placed after the *OK* button in the order of their creation (this order is checked using the *XmQTactivatable* trait).
- 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**(3) for a list of the paths that are searched for these files.

MessageBox uses the *XmQTactivatable* and *XmQTmenuSystem* traits.

Descendants

MessageBox automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
Cancel	**XmPushButtonGadget**	Cancel button
Help	**XmPushButtonGadget**	Help button
Message	**XmLabelGadget**	displayed message
OK	**XmPushButtonGadget**	OK button
Separator	**XmSeparatorGadget**	dividing line between message and buttons
Symbol	**XmLabelGadget**	icon symbolizing message type

Classes

MessageBox inherits behavior, resources, and traits 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	Type	Default	Access
XmNcancelCallback	XmCCallback	XtCallbackList	NULL	C
XmNcancelLabel- String	XmCCancelLabel- String	XmString	dynamic	CSG
XmNdefaultButton- Type	XmCDefaultButtonT- ype	unsigned char	XmDIALOG_OK_- BUTTON	CSG
XmNdialogType	XmCDialogType	unsigned char	XmDIALOG_- MESSAGE	CSG
XmNhelpLabelString	XmCHelpLabel- tring	XmString	dynamic	CSG
XmNmessage- Alignment	XmCAlignment	unsigned char	XmALIGNMENT_- BEGINNING	CSG
XmNmessageString	XmCMessageString	XmString	""	CSG
XmNminimizeButtons	XmCMinimize- Buttons	Boolean	False	CSG
XmNokCallback	XmCCallback	XtCallbackList	NULL	C
XmNokLabelString	XmCOkLabelString	XmString	dynamic	CSG
XmNsymbolPixmap	XmCPixmap	Pixmap	dynamic	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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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:

XmDIALOG_ERROR

Indicates an ErrorDialog.

XmDIALOG_INFORMATION

Indicates an InformationDialog.

XmDIALOG_MESSAGE

Indicates a MessageDialog. This is the default MessageBox dialog type. It does not have an associated message symbol.

XmDIALOG_QUESTION

Indicates a QuestionDialog.

XmDIALOG_TEMPLATE

Indicates a TemplateDialog. The TemplateDialog contains only a separator child. It does not have an associated message symbol.

XmDIALOG_WARNING
Indicates a WarningDialog.

XmDIALOG_WORKING
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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

XmNmessageAlignment
Controls the alignment of the message Label. Possible values include the following:

- **XmALIGNMENT_BEGINNING** (default)
- **XmALIGNMENT_CENTER**
- **XmALIGNMENT_END**

See the description of **XmNalignment** in the **XmLabel** reference page for an explanation of these values.

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 this resource is 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*.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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**	**Type**	**Default**	**Access**
XmNallowOverlap	XmCAllowOverlap	Boolean	True	CSG
XmNautoUnmanage	XmCAutoUnmanage	Boolean	True	CG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNcancelButton	XmCWidget	Widget	Cancel button	SG
XmNdefaultButton	XmCWidget	Widget	dynamic	SG
XmNdefaultPosition	XmCDefaultPosition	Boolean	True	CSG
XmNdialogStyle	XmCDialogStyle	unsigned char	dynamic	CSG
XmNdialogTitle	XmCDialogTitle	XmString	NULL	CSG
XmNfocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNmapCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	10	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNnoResize	XmCNoResize	Boolean	False	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_ANY	CSG

XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_OUT	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNtextTranslations	XmCTranslations	XtTranslations	NULL	C
XmNunmapCallback	XmCCallback	XtCallbackList	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlight- Pixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_- GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Composite Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNchildren	XmCReadOnly	WidgetList	NULL	G

XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	G

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	N/A
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen- -Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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:

KeyosfCancel:
Calls the activate callbacks for the cancel button if it is sensitive.

KeyosfActivate:
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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmBulletinBoard**(3), **XmCreateErrorDialog**(3), **XmCreateInformationDialog**(3), **XmCreateMessageBox**(3), **XmCreateMessageDialog**(3), **XmCreateQuestionDialog**(3), **XmCreateTemplateDialog**(3), **XmCreateWarningDialog**(3), **XmCreateWorkingDialog**(3), **XmManager**(3), and **XmMessageBoxGetChild**(3).

XmNotebook

Purpose

The Notebook widget class

Synopsis

#include <Xm/Notebook.h>

Description

Notebook is a manager widget that organizes its children into pages, tabs, status areas, and page scrollers to simulate a real notebook. It stacks its page children so that all page children occupy the same area like real book pages. Notebook displays visuals that look like the binding of a book and the edges of other pages around the page that is shown. Tab children simulate notebook tabs. Major tabs divide Notebook into several sections, and minor tabs subdivide these sections. Status area children provide additional information about pages such as page numbers. The page scroller child allows the user to move from page to page. Notebook also provides tab scrollers for scrolling major and minor tabs if it cannot display all tabs within its edges.

The application creates pages, tabs, status areas, and page scroller as children of the Notebook widget. Notebook creates tab scrollers when the Notebook is created.

The **XmNnotebookChildType** constraint resource of Notebook determines whether a child widget is a page, tab, status area, or page scroller. Any Motif widget can be a page of the Notebook. When the application creates a child of the Notebook widget without setting the child type constraint, the child becomes a page by default, unless it has the *XmQTactivatable*, *XmQTaccessTextual*, or *XmQTnavigator* trait. Children with the *XmQTactivatable*, *XmQTaccessTextual*, or *XmQTnavigator* trait become major tabs, status areas, and page scrollers, respectively.

Notebook uses the *XmQTaccessTextual*, *XmQTactivatable*, *XmQTjoinSide*, and *XmQTnavigator* traits, and installs the *XmQTscrollFrame* trait.

The application attaches a tab to a page by creating a tab child of the Notebook and setting the **XmNpageNumber** constraint to the page number of the targeted page. By the same method, a status area widget can be attached to a page. The page scroller child, on the other hand, is associated with the Notebook, not with a specific page. Therefore, there is only one valid page scroller for each Notebook.

Pages

Only one child of type **XmPAGE** is displayed at a time by Notebook. Other page children are hidden off-screen. When Notebook displays a particular page, it positions the previously-displayed page off-screen and puts the new page in its place. The page is resized to fit into the dimensions that Notebook has allocated to display pages.

Page Numbers

Notebook uses the **XmNcurrentPageNumber**, **XmNfirstPageNumber**, and **XmNlastPageNumber** resources to determine the current page and available page number range. Only those pages whose page numbers are within the range can be displayed. Other pages cannot be displayed until the range between **XmNfirstPageNumber** and **XmNlastPageNumber** is changed to include them or their page numbers are changed to a number within the range.

If **XmNfirstPageNumber** and **XmNlastPageNumber** are not set explicitly by the application, they are set to 1 by default; Notebook sets **XmNlastPageNumber** to the largest page number assigned by the application thereafter by default. However, once **XmNlastPageNumber** is set by the application, Notebook no longer changes it even when a page with a higher page number is managed.

The **XmNpageNumber** constraint resource is used for specifying the page number of a page widget. It can be set to any integer. For tab and status area children, the resource is used for linking the child widget to a page. For the page scroller child, the resource has no meaning and is ignored by the Notebook.

When a page without a page number is managed, Notebook assigns it the smallest unallocated page number that is not less than the first page number and greater than the last allocated page number. When a tab or a status area without a page number is managed, the newly managed widget is assigned the page number of the most recently managed page, unless the page already has the same type of child. If the page does have the same type of child, Notebook assigns the newly managed widget a page number one greater than the most recently managed page; this new page number is now occupied. Notebook may generate a default page number greater than **XmNlastPageNumber**, making those pages inaccessible to the user.

Duplicate and Empty Pages

Since an application can create or change page numbers, it is possible to have duplicate page numbers and empty pages. When two pages with the same page number are managed, only the more recently managed page can be displayed. Inserting a page with an existing page number does not cause a warning. The old page widget cannot be displayed until the new page widget is removed from the Notebook or until the page number of the old page widget is changed to some other number.

An empty page is a page slot where no page is inserted. Empty pages occur when a tab or status area is associated with a page number that has no matching page widget. Empty pages display the blank Notebook background unless the application provides visual information to this empty area while processing **XmNpageChangedCallback**.

Notebook Visuals

Notebook draws lines around two sides of the top page to simulate the edges of other pages that are behind the top page. The **XmNbackPagePlacement** and **XmNorientation** resources determine which two sides have the lines drawn around them. By default, they are drawn on the bottom and right sides of the top page. The application can set resources to control how many lines are drawn and how wide the area that they are drawn in is. Applications can also choose from three styles of binding visual that simulates the binding of a Notebook. Solid or spiral bindings can be drawn by Notebook, or the application can supply a pixmap that is tiled into the binding.

Tabs

A major or minor tab is a Motif widget with the *XmQTactivatable* trait. If a widget without the trait is created for a tab, Notebook does not provide the page activation callback. As a result, even though the tab is displayed, it cannot automatically move the associated page to the top.

Major tabs divide the Notebook pages into sections. Minor tabs subdivide these sections. Only minor tabs associated with the current section are displayed, where a section consists of the group of pages between the current major tab and the next major tab, including the current major tab but not including the page containing the next major tab. The exception to this is when there is no preceding major tab, in which case the section starts from the **XmNfirstPageNumber** value. A user in one major tab section does not see the minor tabs in other sections. However, all tabs are used in computing the size of the Notebook.

Unlike regular notebook tabs, tabs in the Notebook are not attached to a physical page (a widget). They are, instead, attached to a logical page (a page number). Therefore, it is possible to have a tab with an empty page. When a page with a tab is removed from the Notebook, the tab is not removed because it is still bound to a logical page. Destroying or unmanaging a page widget only erases the page and leaves an empty page. It does not tear the page out of the Notebook. To remove the tab, the application must explicitly destroy or unmanage it.

Notebook supports the *XmQTjoinSide* trait. A widget that has the *XmQTjoinSide* trait can be added to the Notebook as a Major or Minor tab and will appear to be attached to its associated page with no margins or shadows between them.

Status Areas

A status area is any widget that is used for describing the associated page. For example, the Label widget as a status area child can hold a simple string or a pixmap that describes a page. A status area widget is also attached to a page by the page number constraint resource. Therefore, it is possible to have multiple status area widgets for one page. Only the most recently managed status area widget for that page can be displayed. All others for that page are not unmanaged, but their sizes are used for computing the size of the Notebook. If no status area widget is provided, the Notebook displays its blank background in the status area's reserved space. Notebook does not create any default status area widget.

Page Scrollers

The page scroller of the Notebook is any widget that the application creates for scrolling pages. If the application does not create one when the Notebook is realized, Notebook creates a SpinBox widget as the default page scroller. If the application creates a new page scroller, the default page scroller is destroyed. If the application creates multiple page scrollers, only the most recently managed one can be displayed and used. All others are unmanaged.

The default SpinBox page scroller grays out one of the arrow visuals if the current page is a boundary page. If the current page is the first page, the previous arrow of the SpinBox is grayed. If the current page is the last page, the next arrow of the SpinBox is grayed.

Tab Scrollers

Tab scrollers are created by the Notebook for scrolling major tabs and minor tabs. When Notebook is initialized, it creates four ArrowButtonGadgets for scrolling to the next major tab, the previous major tab, the next minor tab, and the previous minor tab. The application cannot replace these tab scrollers. The application can change all resources of these widgets except the position and the arrow direction. Tab scrollers are only visible and enabled when there is not enough space to display all the major or minor tabs appropriate to the page. Tab scrollers are also grayed out when scrolling is inappropriate. The following lists the tab scrollers that are created:

Child Widgets that XmNotebook Creates		
Child	**Name**	**Widget Class**
Page Scroller	PageScroller	XmSpinBox
Next Major Tab Scroller	MajorTabScrollerNext	XmArrowButtonGadget
Previous Major Tab Scroller	MajorTabScrollerPrevious	XmArrowButtonGadget
Next Minor Tab Scroller	MinorTabScrollerNext	XmArrowButtonGadget
Previous Minor Tab Scroller	MinorTabScrollerPrevious	XmArrowButtonGadget

When the user selects the page scroller, a major tab, or a minor tab, the value of **XmNcurrentPageNumber** is changed to the selected page number and **XmNpageChangedCallback** is invoked. After the application returns from the callback, the Notebook displays the last page child whose page number is equal to the current page number. It also displays the last matched status area child. All other pages and status areas are automatically hidden. Major tabs and minor tabs that can fit into the Notebook's edges are displayed and positioned appropriately. All other tabs are also hidden. The application can also cause a page change by calling **XtSetValues** on **XmNcurrentPageNumber** and then calling **XtCallCallbacks** on **XmNpageChangedCallback**.

Orientation

The Notebook has eight different visual configurations, depending on the value of **XmNbackPagePlacement** and **XmNorientation**. These two resources determine the placement of back pages, the binding, major tabs, minor tabs, the status area, and the page scroller. The location of the binding is determined by **XmNorientation**. Major tabs are always placed on the back page side opposite to the binding; Minor tabs are placed on the back page display area that is visually connected to the binding. Both Major and Minor tabs are ordered so that the page numbers they access increase as they get closer to the corner where the back pages meet. The status area and the page scroller are always located on the bottom of the Notebook, inside the frame. The page scroller is always placed adjacent to a back page side. The following table shows the possible configurations and the locations of each Notebook component within the configuration. The default back page value and the default orientation are based upon **XmNlayoutDirection**.

Notebook Configurations				
XmNbackPage-Placement	**XmNorientation**	**Major Tabs**	**Status Area Minor Tabs**	**Binding Page Scroller**
XmBOTTOM_RIGHT	XmHORIZONTAL	RIGHT	BOTTOM LEFT BOTTOM	LEFT BOTTOM RIGHT
XmBOTTOM_RIGHT	XmVERTICAL	BOTTOM	BOTTOM LEFT RIGHT	TOP BOTTOM RIGHT
XmBOTTOM_LEFT	XmHORIZONTAL	LEFT	BOTTOM RIGHT BOTTOM	RIGHT BOTTOM LEFT
XmBOTTOM_LEFT	XmVERTICAL	BOTTOM	BOTTOM RIGHT LEFT	TOP BOTTOM LEFT
XmTOP_RIGHT	XmHORIZONTAL	RIGHT	BOTTOM LEFT TOP	LEFT BOTTOM RIGHT
XmTOP_RIGHT	XmVERTICAL	TOP	BOTTOM LEFT RIGHT	BOTTOM BOTTOM RIGHT
XmTOP_LEFT	XmHORIZONTAL	LEFT	BOTTOM RIGHT TOP	RIGHT BOTTOM LEFT
XmTOP_LEFT	XmVERTICAL	TOP	BOTTOM RIGHT LEFT	BOTTOM BOTTOM LEFT

There are three tab groups for tab group traversal inside the Notebook: major tabs, minor tabs, and the page scroller. The application can also create additional types of tab groups within the Notebook; for example, each page added by the application is treated as a separate tab group by the traversal actions.

Classes

Notebook inherits behavior, resources, and traits from **Core**, **Composite**, **Constraint**, and **XmManager** classes.

The class pointer is *xmNotebookWidgetClass*.

The class name is **XmNotebook**.

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).

XmNotebook Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbackPage- Background	XmCBackPage- Background	Pixel	dynamic	CSG
XmNbackPage- Foreground	XmCBackPage- Foreground	Pixel	dynamic	CSG
XmNbackPage- Number	XmCBackPage- Number	Cardinal	2	CSG
XmNbackPage- Placement	XmCBackPage- Placement	unsigned char	dynamic	CSG
XmNbackPageSize	XmCBackPageSize	Dimension	8	CSG
XmNbindingPixmap	XmCBindingPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNbindingType	XmCBindingType	unsigned char	XmSPIRAL	CSG
XmNbindingWidth	XmCBindingWidth	Dimension	25	CSG
XmNcurrentPage- Number	XmCCurrentPage- Number	int	dynamic	CSG
XmNfirstPageNumber	XmCFirstPageNumber	int	1	CSG
XmNframeBackground	XmCFrameBackground	Pixel	dynamic	CSG
XmNframeShadow- Thickness	XmCShadowThickness	Dimension	0	CSG
XmNinnerMarginHeight	XmCInnerMargin- Height	Dimension	0	CSG
XmNinnerMarginWidth	XmCInnerMargin- Width	Dimension	0	CSG
XmNlastPageNumber	XmCLastPageNumber	int	dynamic	CSG
XmNminorTabSpacing	XmCMinorTabSpacing	Dimension	3	CSG
XmNmajorTabSpacing	XmCMajorTabSpacing	Dimension	3	CSG
XmNorientation	XmCOrientation	unsigned char	XmHORIZONTAL	CSG
XmNpageChanged- Callback	XmCCallback	XtCallbackList	NULL	C

XmNbackPageBackground
Specifies the background color for drawing back pages. The default is a lower-intensity version of **XmNframeBackground**.

XmNbackPageForeground
Specifies the forground color for drawing back pages. The default is taken from the application's default foreground color.

XmNbackPageNumber
Specifies the number of lines to draw for back pages. The minimum value is 1, and the maximum value is (**XmNbackPageSize** / 2).

XmNbackPagePlacement
Specifies where to place the back pages. The default is dependent on the **XmNlayoutDirection** resource of the Notebook's instance parents. It can have one of the following values:

XmBOTTOM_RIGHT
Displays back pages on the Notebook's bottom and right sides.

XmBOTTOM_LEFT
Displays back pages on the Notebook's bottom and left sides.

XmTOP_RIGHT
Displays back pages on the Notebook's top and right sides.

XmTOP_LEFT
Displays back pages on the Notebook's top and left sides.

XmNbackPageSize
Specifies the thickness of the back page rendering.

XmNbindingPixmap
Specifies the pixmap or bitmap for stippling or tiling the binding when **XmNbindingType** is **XmPIXMAP** or **XmPIXMAP_OVERLAP_ONLY**.

XmNbindingType
Specifies the binding type. It can have one of the following values:

XmNONE Displays no binding.

XmSOLID Displays a solid binding in the foreground color of the Notebook within the binding area specified by **XmNbindingWidth**.

XmSPIRAL Displays a spiral binding in the foreground color of the Notebook within the area specified by **XmNbindingWidth** and within the area outside of the frame equal to the area specified by **XmNbindingWidth**.

XmPIXMAP
Displays the binding with the pixmap or bitmap specified by **XmNbindingPixmap** as a stipple or tile. It uses the foreground color of the Notebook for stippling. The binding width is decided by the larger value of **XmNbindingWidth** and the width of the pixmap or bitmap.

XmPIXMAP_OVERLAP_ONLY
Displays the binding with the pixmap or bitmap specified by **XmNbindingPixmap** as a stipple or tile. It uses the foreground color of the Notebook for stippling. The binding is displayed only within the binding area specified by **XmNbindingWidth**.

XmNbindingWidth
Specifies the width of the Notebook binding. If **XmNbindingType** is **XmPIXMAP** and the width of the pixmap specified in **XmNbindingPixmap** is greater than **XmNbindingWidth**, then this resource is ignored and the width of the pixmap is used as the width of the Notebook binding instead.

XmNcurrentPageNumber
Specifies the page number of the currently displayed page. Initially, it is set to **XmNfirstPageNumber**. If it is set to less than **XmNfirstPageNumber**, then it is set to **XmNfirstPageNumber**. If it is set to **XmNlastPageNumber**, then it is set to **XmNlastPageNumber**.

XmNfirstPageNumber
Specifies the page number for the first page of the Notebook. The Notebook does not scroll to any page numbers below this value.

XmNframeBackground
Specifies the background color for drawing the Notebook's frame.

XmNframeShadowThickness

Specifies the shadow thickness around the Notebook's frame.

XmNinnerMarginHeight

Specifies the margin on the top and bottom sides of the page, status area, and page scroller widgets.

XmNinnerMarginWidth

Specifies the margin on the left and right sides of the page, status area, and page scroller widgets.

XmNlastPageNumber

Specifies the page number for the last page of the Notebook. The Notebook does not scroll to any page numbers above this value. The default page number is the largest page number of managed page, major tab, or minor tab widgets. If this is set to a value that is less than **XmNfirstPageNumber**, the behavior of the Notebook is undefined.

XmNmajorTabSpacing

Specifies the spacing distance between major tabs. If **XmNframeShadowThickness** is greater than **XmNmajorTabSpacing**, then this resource is ignored and the size of **XmNframeShadowThickness** is used as the spacing distance between major tabs.

XmNminorTabSpacing

Specifies the spacing distance between minor tabs. If **XmNframeShadowThickness** is greater than **XmNminorTabSpacing**, then this resource is ignored and the size of **XmNframeShadowThickness** is used as the spacing distance between minor tabs.

XmNorientation

Specifies the orientation of the Notebook. It can have one of the following values:

XmHORIZONTAL

Places the binding beside the pages, in the left or right side of the frame.

XmVERTICAL

Places the binding above or below the pages, in the top or the bottom of the frame.

XmNpageChangedCallback

Specifies the list of callbacks to call whenever the **XmNcurrentPageNumber**, representing the current page number, is changed. This includes the point when the widget is realized and the page number is initialized. The callback structure is **XmNotebookCallbackStruct**. The reason is **XmCR_MAJOR_TAB**, **XmCR_MINOR_TAB**, **XmCR_PAGE_SCROLLER_INCREMENT**, **XmCR_PAGE_SCROLLER_DECREMENT**, or **XmCR_NONE**, depending upon what action caused the Notebook to display a new page.

XmNotebook Constraint Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNnotebookChildType	XmCNotebookChildType	unsigned char	dynamic	CG
XmNpageNumber	XmCPageNumber	int	dynamic	CSG
XmNresizable	XmCResizable	Boolean	True	CSG

XmNnotebookChildType

Specifies the child type of the Notebook. It can be one of the following types:

XmPAGE The child is a page of the Notebook. This is the default when the child does not have the *XmQTactivatable*, *XmQTaccessTextual*, or *XmQTnavigator* trait.

XmMAJOR_TAB

The child is a major tab. This is the default when the child has the *XmQTactivatable* trait.

XmMINOR_TAB

The child is a minor tab.

XmSTATUS_AREA

The child is a status area. This is the default when the child has the *XmQTaccessTextual* trait and does not have the *XmQTactivatable* trait.

XmPAGE_SCROLLER

The child is the page scroller. The default page scroller is destroyed, if it exists. Any previously created page scrollers are unmanaged. This is the default when the child has the *XmQTnavigator* trait and does have the *XmQTactivatable* trait or the *XmQTaccessTextual* trait.

XmNpageNumber

Specifies the page number associated with the widget. If the widget is a page, the number specifies the page number of the widget. If the widget is not a page, the number specifies the page number of the associated page. If none is supplied by the application, Notebook generates the smallest unallocated page number when the child is managed. This resource is ignored for the page scroller.

XmNresizable

Specifies whether this child can request a resize.

Inherited Resources

Notebook 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**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG

XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefault-Foreground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG

XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback

A pointer to the following structure is passed to callbacks for **XmNpageChangedCallback**.

```
typedef struct
{
        int reason;
        XEvent *event;
        int page_number;
        Widget page_widget;
        int prev_page_number;
        Widget prev_page_widget;
} XmNotebookCallbackStruct;
```

reason Specifies the reason for the callback.

event Points to the *XEvent* that triggered the callback. It can be NULL.

page_number
Indicates the page number to be displayed.

page_widget
Indicates the page widget that has the new page number. It is NULL if no page widget with the page number is found.

prev_page_number
Indicates the page number of the currently displayed page. If the callback procedure is being called at widget initialization, this page number will be returned as **XmUNSPECIFIED_PAGE_NUMBER**.

prev_page_widget
Indicates the currently displayed page widget. If the callback procedure is being called at widget initialization, NULL will be returned.

Translations

Notebook inherits translations from Manager.

Accelerators

Notebook accelerators are added to all major tab and minor tab children of XmNotebook. Notebook accelerators are listed below. These accelerators might not directly correspond to a translation table.

<osfBeginLine>:
: **TraverseTab(***Home***)**

<osfEndLine>:
: **TraverseTab(***End***)**

<osfLeft>:
: **TraverseTab(***Previous***)**

<osfRight>:
: **TraverseTab(***Next***)**

<osfUp>:
: **TraverseTab(***Previous***)**

<osfDown>:
: **TraverseTab(***Next***)**

Action Routines

Notebook action routines are described below:

TraverseTab(*Home*|*End*|*Next*|*Previous*)
: Moves the focus on major or minor tabs.

Additional Behavior

The Notebook widget has the additional behavior described below:

Tab
: Notebook intercepts tab group traversal when traversal is entering or leaving major or minor tabs. It does this to support major tabs and minor tabs as two separate tab groups when they are actually treated as one by traversal. If a minor tab has keyboard focus and a user or program action specifies **XmTRAVERSE_PREV_TAB_GROUP**, keyboard focus will go to a major tab. If a major tab has keyboard focus and a user or program action specifies **XmTRAVERSE_NEXT_TAB_GROUP**, keyboard focus will go to a minor tab.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateNotebook**(3), **XmManager**(3), and **XmNotebookGetPageInfo**(3).

XmPanedWindow

Purpose

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 all panes to appear first in the **XmNchildren** list and all sashes to appear after the panes. For a pane child, it returns the value of **XmNpositionIndex** if one has been specified for the child. Otherwise, it returns the number of *panes* in the PanedWindow's **XmNchildren** list. Other than the fact that all sashes appear after all panes, the insertion order of sashes is unspecified. This procedure also causes nonsash widgets to be inserted after other nonsash children but before any sashes. The behavior of PanedWindow is undefined if **XmNinsertPosition** is set to a procedure other than the default.

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**.

Descendants

PanedWindow automatically creates for each paned window the descendants shown in the following table. An application can use **XtName** and the child list information to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
Sash	**subclass of XmPrimitive**	sash
Separator	**XmSeparatorGadget**	dividing line between window panes

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**	**Type**	**Default**	**Access**
XmNmarginHeight	XmCMarginHeight	Dimension	3	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	3	CSG

XmNorientation	XmCOrientation	unsigned char	XmVERTICAL	CSG
XmNrefigureMode	XmCBoolean	Boolean	True	CSG
XmNsashHeight	XmCSashHeight	Dimension	10	CSG
XmNsashIndent	XmCSashIndent	Position	-10	CSG
XmNsashShadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNsashWidth	XmCSashWidth	Dimension	10	CSG
XmNseparatorOn	XmCSeparatorOn	Boolean	True	CSG
XmNspacing	XmCSpacing	Dimension	8	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.

XmNorientation
Specifies the layout as either vertical (with the **XmVERTICAL** value) or horizontal (with the **XmHORIZONTAL** value). In the vertical layout, the children are laid out in a vertically tiled format. In the horizontal layout, the children are laid out in a horizontal layout, with the sash moveable along the horizontal axis.

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 depends on the **XmNlayoutDirection** resource of the widget.

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	Type	Default	Access
XmNallowResize	XmCBoolean	Boolean	False	CSG
XmNpaneMaximum	XmCPaneMaximum	Dimension	1000	CSG
XmNpaneMinimum	XmCPaneMinimum	Dimension	1	CSG
XmNpositionIndex	XmCPositionIndex	short	XmLAST_POSITION	CSG
XmNskipAdjust	XmCBoolean	Boolean	False	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**	**Type**	**Default**	**Access**
XmNbottomShadow-Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG

XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadow- Pixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG

XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

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

Translations

XmPanedWindow inherits translations from **XmManager**.

The translations for sashes within the PanedWindow are described in the following table.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

≈c ≈s ≈m ≈a <Btn1Down>:
SashAction(*Start***)**

≈c ≈s ≈m ≈a <Btn1Motion>:
SashAction(*Move***)**

≈c ≈s ≈m ≈a <Btn1Up>:
SashAction(*Commit***)**

≈c ≈s ≈m ≈a <Btn2Down>:
SashAction(*Start***)**

≈c ≈s ≈m ≈a <Btn2Motion>:
SashAction(*Move***)**

≈c ≈s ≈m ≈a <Btn2Up>:
SashAction(*Commit***)**

:<Key>osfActivate:
PrimitiveParentActivate()

:<Key>osfCancel:
PrimitiveParentCancel()

:<Key>osfHelp:
Help()

:c <Key>osfUp:
SashAction(*Key,LargeIncr,Up***)**

:<Key>osfUp:
SashAction(*Key,DefaultIncr,Up***)**

:c <Key>osfRight:
SashAction(*Key,LargeIncr,Right***)**

:<Key>osfRight:
SashAction(*Key,DefaultIncr,Right***)**

:c <Key>osfDown:
SashAction(*Key,LargeIncr,Down***)**

:<Key>osfDown:
SashAction(*Key,DefaultIncr,Down***)**

:c <Key>osfLeft:
SashAction(*Key,LargeIncr,Left***)**

:<Key>osfLeft:
SashAction(*Key,DefaultIncr,Left***)**

≈s ≈m ≈a <Key>Return:
PrimitiveParentActivate()

s ≈m ≈a <Key>Tab:
PrevTabGroup()

≈m ≈a <Key>Tab:
NextTabGroup()

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**, **Down**, **Left**, or **Right**.

  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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreatePanedWindow**(3), and **XmManager**(3).

XmPrimitive

Purpose

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. Primitive and all its subclasses hold the *XmQTcareParentVisual* trait.

Data Transfer Behavior

Primitive has no widget class conversion or destination procedure. Subclasses and the **XmNconvertCallback** procedures are responsible for any conversion of selections. Subclasses and the subclass **XmNdestinationCallback** procedures are responsible for any data transfers to the widget.

Classes

Primitive inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottom- ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow- Pixmap	XmCBottom- ShadowPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlight- Color	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight- OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlight- Pixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight- Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayout- Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmNONE	CSG
XmNpopupHandler- Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadow- Thickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadow- Color	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	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.

XmNconvertCallback
Specifies a list of callbacks called when the widget is asked to convert a selection. The type of the structure whose address is passed to these callbacks is **XmConvertCallbackStruct**. The reason is **XmCR_OK**.

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.

XmNlayoutDirection
Specifies the direction in which components of the primitive (including strings) are laid out. The values are of type **XmDirection**. If the widget's parent is a primitive or shell, the value is inherited from the widget's parent. Otherwise, it is inherited from the closest ancestor vendor or menu shell. Refer to the **XmDirection**(3) reference page for the possible direction values.

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**.

XmNpopupHandlerCallback

Allows the application to control which popup menu will be automatically posted. The reason can either be **XmCR_POST** or **XmCR_REPOST:**

XmCR_POST

Indicates that this is a regular posting request.

XmCR_REPOST
Indicates that the menu was just unposted and that this callback was invoked on a replay.

This callback uses the **XmPopupHandlerCallbackStruct** structure to pass information.

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. If a default top shadow pixmap exists, it will need to be removed for the **XmNtopShadowColor** to take effect.

XmNtopShadowPixmap
Specifies the pixmap to use to draw the top and left sides of the border shadow. A Primitive top shadow pixmap is created in two situations. In either of these situations, a default 50-foreground top shadow pixmap is created.

- If the Primitive top shadow color is the same as the Core background pixel color.
- If the depth of the screen is only one.

For example, if a widget with the same top shadow and background color is created, a default shadow pixmap is generated. Such a pixmap needs to be removed for the **XmNtopShadowColor** resource to take effect.

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**.

The unit type can also be specified in resource files, with the following format:

```
<floating value><unit>
```

where:

unit	is <" ", pixels, inches, centimeters, millimeters, points, font units>
pixels	is <*pix*, *pixel*, *pixels*>
inches	is <*in*, *inch*, *inches*>
centimeter	is <*cm*, *centimeter*, *centimeters*>
millimeters	is <*mm*, *millimeter*, *millimeters*>
points	is <*pt*, *point*, *points*>
font units	is <*fu*, *font_unit*, *font_units*>
float	is {"+"|"-"}{{<"0"-"9">*}.}<"0"-"9">*
	Note that the type Dimension must always be positive.

For example,

```
xmfonts*XmMainWindow.height: 10.4cm
*PostIn.width: 3inches
```

XmNunitType can have the following values:

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

XmMILLIMETERS
All values provided to the widget are treated as normal millimeter values.

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

XmCENTIMETERS
All values provided to the widget are treated as normal centimeter values.

XmINCHES
All values provided to the widget are treated as normal inch values.

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

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

XmFONT_UNITS
All values provided to the widget are treated as normal font units. A font unit has horizontal and vertical components. These are the values of the XmScreen resources **XmNhorizontalFontUnit** and **XmNverticalFontUnit**.

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**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources- Persistent	XmCInitialResources- Persistent	Boolean	True	C
XmNmappedWhen- Managed	XmCMappedWhen- Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG

XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback Information

A pointer to the following structure is passed to each callback for **XmNhelpCallback**:

```
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.

A pointer to the following callback structure is passed to the **XmNconvertCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent *event;
        Atom selection;
        Atom target;
        XtPointer source_data;
        XtPointer location_data;
        int flags;
        XtPointer parm;
        int parm_format;
        unsigned long parm_length;
        Atom parm_type;
        int status;
        XtPointer value;
        Atom type;
        int format;
        unsigned long length;
} XmConvertCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which conversion is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

target Indicates the conversion target.

source_data Contains information about the selection source. When the selection is _MOTIF_DROP, *source_data* is the DragContext. Otherwise, it is NULL.

location_data
Contains information about the location of data to be converted. If the value is NULL, the data to be transferred consists of the widget's current selection. Otherwise, the type and interpretation of the value are specific to the widget class.

flags Indicates the status of the conversion. Following are the possible values:

XmCONVERTING_NONE
This flag is currently unused.

XmCONVERTING_PARTIAL
The target widget was able to be converted, but some data was lost.

XmCONVERTING_SAME
The conversion target is the source of the data to be transferred.

XmCONVERTING_TRANSACT
This flag is currently unused.

parm Contains parameter data for this target. If no parameter data exists, the value is NULL.

When *selection* is *CLIPBOARD* and *target* is _MOTIF_CLIPBOARD_TARGETS or _MOTIF_DEFERRED_CLIPBOARD_TARGETS, the value is the requested operation (**XmCOPY**, **XmMOVE**, or **XmLINK**).

parm_format
Specifies whether the data in *parm* should be viewed as a list of *char*, *short*, or *long* quantities. Possible values are 0 (when *parm* is NULL), 8 (when the data in *parm* should be viewed as a list of *char*s), 16

(when the data in *parm* should be viewed as a list of *short*s), or 32 (when the data in *parm* should be viewed as a list of *long*s). Note that *parm_format* symbolizes a data type, not the number of bits in each list element. For example, on some machines, a *parm_format* of 32 means that the data in *parm* should be viewed as a list of 64-bit quantities, not 32-bit quantities.

parm_length Specifies the number of elements of data in *parm*, where each element has the size specified by *parm_format*. When *parm* is NULL, the value is 0.

parm_type Specifies the parameter type of *parm*.

status An IN/OUT member that specifies the status of the conversion. The initial value is **XmCONVERT_DEFAULT**. The callback procedure can set this member to one of the following values:

XmCONVERT_DEFAULT
This value means that the widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it overwrites the data provided by the callback procedures in the *value* member.

XmCONVERT_MERGE
This value means that the widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it appends its data to the data provided by the callback procedures in the *value* member. This value is intended for use with targets that result in lists of data, such as *TARGETS*.

XmCONVERT_DONE
This value means that the callback procedure has successfully finished the conversion. The widget class conversion procedure, if any, is not called after the callback procedures return.

XmCONVERT_REFUSE
This value means that the callback procedure has terminated the conversion process without completing the requested conversion. The widget class conversion

procedure, if any, is not called after the callback procedures return.

value	An IN/OUT parameter that contains any data that the callback procedure produces as a result of the conversion. The initial value is NULL. If the callback procedure sets this member, it must ensure that the *type*, *format*, and *length* members correspond to the data in *value*. The callback procedure is responsible for allocating memory when it sets this member. The toolkit frees this memory when it is no longer needed.
type	An IN/OUT parameter that indicates the type of the data in the *value* member. The initial value is *INTEGER*.
format	An IN/OUT parameter that specifies whether the data in *value* should be viewed as a list of *char*, *short*, or *long* quantities. The initial value is 8. The callback procedure can set this member to 8 (for a list of *char*), 16 (for a list of *short*), or 32 (for a list of *long*).
length	An IN/OUT member that specifies the number of elements of data in *value*, where each element has the size symbolized by *format*. The initial value is 0.

A pointer to the following structure is passed to each callback for **XmNpopupHandlerCallback**:

```
typedef struct
{
        int reason;
        XEvent * xevent;
        Widget menuToPost;
        Boolean postIt;
        Widget target;
} XmPopupHandlerCallbackStruct;
```

reason	Indicates why the callback was invoked.
xevent	Points to the *XEvent* that triggered the handler.
menuToPost	Specifies the popup menu that the menu system believes should be posted. The application may modify this field.
postIt	Indicates whether the posting process should continue. The application may modify this field.

target Specifies the most specific widget or gadget that the menu sytem found from the event that matches the event.

Translations

The **XmPrimitive** translations are listed below.

Note that for buttons in menus, altering translations in **#override** or **#augment** mode is undefined.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

:<Key>osfActivate:
PrimitiveParentActivate()

:<Key>osfCancel:
PrimitiveParentCancel()

:<Key>osfBeginLine:
PrimitiveTraverseHome()

:<Key>osfUp:
PrimitiveTraverseUp()

:<Key>osfDown:
PrimitiveTraverseDown()

:<Key>osfLeft:
PrimitiveTraverseLeft()

:<Key>osfRight:
PrimitiveTraverseRight()

≈s ≈m ≈a <Key>Return:
PrimitiveParentActivate()

s ≈m ≈a <Key>Tab:
PrimitivePrevTabGroup()

≈m ≈a <Key>Tab:
PrimitiveNextTabGroup()

<Key>osfHelp:
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():
: This action depends on the value of the Display resource **XmNenableButtonTab**. When **XmNenableButtonTab** is False (default), this action 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.

  When **XmNenableButtonTab** is True, this action moves to the next item within the current tab group, unless it is the last item in the tab group. When the item is the last in the group, the action traverses to the first item in the next tab group. The **XmNenableButtonTab** behavior applies only to PushButton, ArrowButton, and DrawnArrow.

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():
: This action depends on the value of the Display resource **XmNenableButtonTab**. When **XmNenableButtonTab** is False (default), this action 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.

  When **XmNenableButtonTab** is True, this action moves to the previous item within the current tab group unless it is the first item in the tab group. When the item is the first in the group, the action traverses to the first item in the previous tab group. The **XmNenableButtonTab** behavior applies only PushButton, ArrowButton, and DrawnButton.

PrimitiveTraverseDown():
: Traverses to the next item below the current widget in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

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. The wrapping direction depends on the layout direction of the widget tab group.

PrimitiveTraverseNext():
: Traverses to the next item in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

PrimitiveTraversePrev():
: Traverses to the previous item in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

PrimitiveTraverseRight():
: Traverses to the next item to the right of the current gadget in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

PrimitiveTraverseUp():
: Traverses to the next item above the current gadget in the current tab group, wrapping if necessary. The wrapping direction depends on the layout direction of the widget tab group.

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**(3).

Related Information

Core(3), **XmDirection**(3), **XmChangeColor**(3), and **XmScreen**(3).

XmPushButton

Purpose

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, Btn1 controls the behavior of the PushButton. In addition, Btn3 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**.

PushButton uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits, and holds the *XmQactivatable*, *XmQTmenuSavvy*, and *XmQTtakesDefault* traits.

Classes

PushButton inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallback- List	NULL	C
XmNarmCallback	XmCCallback	XtCallback- List	NULL	C
XmNarmColor	XmCArmColor	Pixel	dynamic	CSG
XmNarmPixmap	XmCArmPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNdefaultButton-ShadowThickness	XmCDefaultButton-ShadowThickness	Dimension	dynamic	CSG
XmNdisarmCallback	XmCCallback	XtCallback- List	NULL	C
XmNfillOnArm	XmCFillOnArm	Boolean	True	CSG
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	CSG
XmNshowAsDefault	XmCShowAsDefault	Dimension	0	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**. This callback uses the *XmQTactivatable* trait.

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.

This resource is also used when the button is armed in a menu and the **XmNenableEtchedInMenu** resource is **True**.

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. When the Display resource **XmNdefaultButtonEmphasis** has a value of **XmEXTERNAL_HIGHLIGHT** (the default), PushButton draws the location cursor outside of the outer shadow. When this resource has a value of **XmINTERNAL_HIGHLIGHT**, the highlight is drawn between the inner and outer shadows. 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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	CSG
XmNacceleratorText	XmCAccelerator- Text	XmString	NULL	CSG
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	dynamic	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	dynamic	CSG

XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	dynamic	CSG
XmNmarginRight	XmCMarginRight	Dimension	dynamic	CSG
XmNmarginTop	XmCMarginTop	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonicCharSet	XmCMnemonic-CharSet	String	XmFONTLIST_-DEFAULT_TAG	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString-Direction	dynamic	CSG

XmPrimitive Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallback- List	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallback- List	NULL	C
XmNhighlightColor	XmCHighlight-Color	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight-OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlight-Pixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayout-Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigation-Type	XmNavigation-Type	XmNONE	CSG

XmNpopupHandler- Callback	XmCCallback	XtCallback- List	NULL	C
XmNshadowThickness	XmCShadow- Thickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadow- Color	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources- Persistent	XmCInitialResources- Persistent	Boolean	True	C
XmNmappedWhen- Managed	XmCMappedWhen- Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG

XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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 *Primitive*.

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.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

c<Btn1Down>:
ButtonTakeFocus()

≈c<Btn1Down>:
  Arm()

≈c<Btn1Down>,≈cBtn1Up:
  Activate() Disarm()

≈c<Btn1Down>(2+):
  MultiArm()

≈c<Btn1Up>(2+):
  MultiActivate()

≈c<Btn1Up>:
  Activate() Disarm()

≈c<Btn2Down>:
  ProcessDrag()

:<Key>osfActivate:
  PrimitiveParentActivate()

:<Key>osfCancel:
  PrimitiveParentCancel()

:<Key>osfSelect:
  ArmAndActivate()

:<Key>osfHelp:
  Help()

≈s ≈m ≈a <Key>Return:
  PrimitiveParentActivate()

≈s ≈m ≈a <Key>space:
  ArmAndActivate()

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, Btn3 also performs the Btn1 actions.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<Btn2Down>:
ProcessDrag()

c<Btn1Down>:
MenuButtonTakeFocus()

c<Btn1Up>:
MenuButtonTakeFocusUp()

≈c<BtnDown>:
BtnDown()

≈c<'BtnUp>:
BtnUp()

:<Key>osfSelect:
ArmAndActivate()

:<Key>osfActivate:
ArmAndActivate()

:<Key>osfCancel:
MenuEscape()

:<Key>osfHelp:
Help()

≈s ≈m ≈a <Key>Return:
ArmAndActivate()

≈s ≈m ≈a <Key>space:
ArmAndActivate()

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.

ButtonTakeFocus():
: Causes the PushButton to take keyboard focus when **Ctrl<Btn1Down>** is pressed, without activating the widget.

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 **XmEXPLICIT**, 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 **XmEXPLICIT**, 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 sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures, possibly multiple times, for the _MOTIF_DROP selection. 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**(3).

Related Information

Core(3), **XmCreatePushButton**(3), **XmLabel**(3), **XmPrimitive**(3), and **XmRowColumn**(3).

XmPushButtonGadget

Purpose

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, Btn1 controls the behavior of the PushButtonGadget. In addition, Btn3 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**.

PushButtonGadget uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits, and holds the *XmQactivatable*, *XmQTmenuSavvy*, and *XmQTtakesDefault* traits.

Classes

PushButtonGadget inherits behavior, resources, and traits 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 Resource Set

Name	**Class**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNarmColor	XmCArmColor	Pixel	dynamic	CSG
XmNarmPixmap	XmCArmPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNdefaultButton-ShadowThickness	XmCdefaultButton-ShadowThickness	Dimension	dynamic	CSG
XmNdisarmCallback	XmCCallback	XtCallbackList	NULL	C
XmNfillOnArm	XmCFillOnArm	Boolean	True	CSG
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	CSG
XmNshowAsDefault	XmCShowAsDefault	Dimension	0	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**. This callback uses the *XmQTactivatable* trait.

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 0 (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 the Display resource **XmNdefaultButtonEmphasis** has a value of **XmEXTERNAL_HIGHLIGHT** (the default), PushButton draws the location cursor outside of the outer shadow. When this resource has a value of **XmINTERNAL_HIGHLIGHT**, the highlight is drawn between the inner and outer shadows. 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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	CSG
XmNacceleratorText	XmCAcceleratorText	XmString	NULL	CSG
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	dynamic	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG

XmNmarginBottom	XmCMarginBottom	Dimension	dynamic	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	dynamic	CSG
XmNmarginRight	XmCMarginRight	Dimension	dynamic	CSG
XmNmarginTop	XmCMarginTop	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonic- CharSet	XmCMnemonic- CharSet	String	dynamic	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CSG

XmGadget Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNbottom- ShadowColor	XmCBottom- ShadowColor	Pixel	dynamic	CSG
XmNbottom- ShadowPixmap	XmCBottom- ShadowPixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOn- Enter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlight- Pixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight- Thickness	Dimension	2	CSG
XmNlayoutDirection	XmNCLayout- Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	XmNONE	CSG
XmNshadowThickness	XmCShadow- Thickness	Dimension	2	CSG

XmNtopShadowColor	XmCTopShadow- Color	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

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

Btn2Down: Drags the contents of a PushButtonGadget label, identified when Btn2 is pressed. This action is undefined for PushButtonGadgets used in a menu system.

Btn1Down: 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.

Btn1**(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.

Btn1Up: 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**.

Btn1Up**(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**.

KeyosfActivate:

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.

KeyosfSelect:

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.

KeyosfHelp: 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.

KeyosfCancel:

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**(3).

Related Information

Object(3), **RectObj**(3), **XmCreatePushButtonGadget**(3), **XmGadget**(3), **XmLabelGadget**(3), and **XmRowColumn**(3).

XmRendition

Purpose

The Rendition registry

Synopsis

#include <Xm/Xm.h>
XmRendition

Description

XmRendition is a pseudo widget used for the rendering of **XmString**s. **XmRendition** has two parts: **XmStringTag** and rendering information. The **XmStringTag** part can be matched with an **XmStringTag** associated with a *LOCALE*, *CHARSET*, or **RENDITION[BEGIN|END]** component within **XmString**. The rendering information contains information about the font or fontset, colors, tabs, and lines to be used in rendering a text component.

If a resource in a rendition is unspecified, usually by setting it to **XmAS_IS** or **XmUNSPECIFIED_PIXEL**, then the value to be used for that resource is the value of the immediately preceeding rendition in **XmString**. If that value is unspecified, then the preceding value is used, and so on. If no renditions specify a value for a resource, then a default value will be used.

Classes

XmRendition 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 **XmRenditionUpdate** (S), retrieved by using **XmRenditionRetrieve** (G), or is not applicable (N/A).

XmRendition Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNrendition-Background	XmCRendition-Background	Pixel	XmUNSPECIFIED_-PIXEL	CSG
XmNfont	XmCFont	XtPointer	XmAS_IS	CSG
XmNfontName	XmCFontName	String	XmAS_IS	CSG
XmNfontType	XmCFontType	XmFontType	XmAS_IS	CSG
XmNrendition-Foreground	XmCRendition-Foreground	Pixel	XmUNSPECIFIED_-PIXEL	CSG
XmNloadModel	XmCLoadModel	unsigned char	XmAS_IS	CSG
XmNstrikethruType	XmCStrikethruType	unsigned char	XmAS_IS	CSG
XmNtabList	XmCTabList	XmTabList	XmAS_IS	CSG
XmNtag	XmCTag	XmStringTag	""	G
XmNunderlineType	XmCUnderlineType	unsigned char	XmAS_IS	CSG

XmNrenditionBackgroundSpecifies the background drawing color. A value of **XmUNSPECIFIED_PIXEL** indicates that the background is not specified for this rendition.

XmNfont Specifies the actual font or fontset to be used by this rendition. The value of this resource, if set to other than **XmAS_IS**, will be used regardless of the settings of the other font resources. Setting this resource will force **XmNloadModel** to be **XmLOAD_IMMEDIATE**. If this resource is not initially set, then it will be set subsequently by the rendition whenever the font or fontset specified by **XmNfontName** is loaded. If both **XmNfontName** and **XmNfont** are specified in a resource file, the **XmNfont** specification will take precedence.

XmNfontName

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. A value of **XmAS_IS** indicates that the font is not specified for this rendition. If both **XmNfontName** and **XmNfont** are specified in a resource file, the **XmNfont** specification will take precedence.

XmNfontType
Specifies whether the **XmNfontName** resource refers to a font name or to a base font name list. Valid values are **XmFONT_IS_FONT** and **XmFONT_IS_FONTSET**.

XmNrenditionForeground
Specifies the foreground drawing color. A value of **XmUNSPECIFIED_PIXEL** indicates that the foreground is not specified for this rendition.

XmNloadModel
Specifies whether the font or fontset specified by **XmNfontName** is to be loaded when the rendition is created (**XmLOAD_IMMEDIATE**) or only when the font is required to render an **XmString** segment (**XmLOAD_DEFERRED**). Note that specifying **XmLOAD_IMMEDIATE** for **XmNloadModel** is valid only if **XmNfontName** is specified, in which case the specified font will be loaded on creation, or if **XmNfont** is specified, in which case the font is already loaded. **XmLOAD_DEFERRED** is only valid when **XmNfontName** is specified.

XmNstrikethruType
Specifies the type of line to be used to strike through a text segment. Valid values are **XmNO_LINE**, **XmSINGLE_LINE**, **XmDOUBLE_LINE**, **XmSINGLE_DASHED_LINE**, and **XmDOUBLE_DASHED_LINE**. A value of **XmAS_IS** indicates that the strike-through type is not specified for this rendition.

XmNtabList
Specifies the tab list to be used in rendering compound strings containing tab components.

XmNtag
Specifies the tag string to be used in matching the renditions with other renditions or with **XmStringTag** components in *XmStrings*. This resource must always be specified. That is, NULL is not a legal value but the empty string is. This resource is automatically set to the value of the *tag* parameter in the **XmRenditionCreate** call.

XmNunderlineType
Specifies the type of line to be used to underline a text segment. Valid values are **XmNO_LINE**, **XmSINGLE_LINE**, **XmDOUBLE_LINE**, **XmSINGLE_DASHED_LINE**, and **XmDOUBLE_DASHED_LINE**.

A value of **XmAS_IS** indicates that the underline type is not specified for this rendition.

Related Information

XmRenditionCreate(3), **XmRenditionFree**(3), **XmRenditionRetrieve**(3), **XmRenditionUpdate**(3), and **XmString**(3).

XmRowColumn

Purpose

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 menu panes, and OptionMenu. RowColumn uses the *XmQTmenuSavvy* trait and holds the *XmQTmenuSystem* trait.

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 specific layout (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.

The default **XmNinsertPosition** procedure for the RowColumn returns the value of **XmNpositionIndex** if one has been specified for the child. Otherwise, this procedure returns the number of children in the RowColumn’s **XmNnumChildren** list. In a MenuBar, Pulldown menu pane, or Popup menu pane 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 menu pane, or Popup menu pane 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 menu pane, or Popup menu pane 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

For the Popup and Pulldown Menupanes, popup and pulldown menus have particular behaviors when the Btn1 button is pressed outside the menus. These behaviors are summarized here.

When there is already a popped up menu, a user can either press Btn1 in the same area as the popped up menu, or can press Btn1 in another area that should have a menu popped up. When Btn1 is pressed in the same area as the already popped up menu, that menu is unposted. If Btn1 is pressed in a different area, the associated popup menu is posted for the new area. Note, however, that if the **XmNpopupHandlerCallback** of either **XmManager** or **XmPrimitive** is available, then the callback may override these default behaviors.

For pulldown menus, a user can press Btn1 on the Cascade button to post the pulldown menu, then click on it again. Upon the second click, the pulldown menu is unposted.

Popup menus are not allowed to have NULL parents.

Tear-off Menus

Pulldown and Popup menu panes support tear-off menus, which enable the user to retain a menu pane on the display to facilitate subsequent menu selections. A menu pane that can be torn-off is identified by a tear-off button that spans the width of the menu pane and displays a dashed line. A torn-off menu pane 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 menu pane also displays an empty title. Tear-off menus should not be shared.

The user can tear off a menu pane using the mouse or keyboard. Clicking Btn1 or pressing osfActivate (or osfSelect) on the tear-off button, tears off the menu pane at the current position. Pressing Btn2 on the tear-off button tears off the menu pane and allows the user to drag the torn-off menu to a new position designated by releasing the mouse button. Tearing off a menu pane unposts the current active menu. Only one tear-off copy for each menu pane is allowed. Subsequent tear-off actions of a torn menu pane 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. The **XmNtearOffTitle** resource enables the application to specify the tear-off menu's title if the menu is torn off.

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.

Descendants

RowColumn automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	Class	Identity
OptionButton	**XmCascadeButtonGadget**	option menu button
OptionLabel	**XmLabelGadget**	option menu label
TearOffControl	subclass of **XmPrimitive**	tear-off button of torn-off menu pane

Classes

RowColumn inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNadjustLast	XmCAdjustLast	Boolean	True	CSG
XmNadjustMargin	XmCAdjustMargin	Boolean	True	CSG
XmNentryAlignment	XmCAlignment	unsigned char	XmALIGNMENT_-BEGINNING	CSG

XmNentryBorder	XmCEntryBorder	Dimension	0	CSG
XmNentryCallback	XmCCallback	XtCallbackList	NULL	C
XmNentryClass	XmCEntryClass	WidgetClass	dynamic	CSG
XmNentryVertical-Alignment	XmCVertical- Alignment	unsigned char	XmALIGNMENT_-CENTER	CSG
XmNisAligned	XmCIsAligned	Boolean	True	CSG
XmNisHomogeneous	XmCIsHomogeneous	Boolean	dynamic	CG
XmNlabelString	XmCXmString	XmString	NULL	C
XmNmapCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	dynamic	CSG
XmNmenuAccelerator	XmCAccelerators	String	dynamic	CSG
XmNmenuHelpWidget	XmCMenuWidget	Widget	NULL	CSG
XmNmenuHistory	XmCMenuWidget	Widget	NULL	CSG
XmNmenuPost	XmCMenuPost	String	NULL	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonicCharSet	XmCMnemonic- CharSet	String	XmFONTLIST_-DEFAULT_TAG	CSG
XmNnumColumns	XmCNumColumns	short	1	CSG
XmNorientation	XmCOrientation	unsigned char	dynamic	CSG
XmNpacking	XmCPacking	unsigned char	dynamic	CSG
XmNpopupEnabled	XmCPopupEnabled	int	XmPOPUP_-KEYBOARD	CSG
XmNradioAlwaysOne	XmCRadioAlwaysOne	Boolean	True	CSG
XmNradioBehavior	XmCRadioBehavior	Boolean	False	CSG
XmNresizeHeight	XmCResizeHeight	Boolean	True	CSG
XmNresizeWidth	XmCResizeWidth	Boolean	True	CSG
XmNrowColumnType	XmCRowColumn- Type	unsigned char	XmWORK_AREA	CG
XmNspacing	XmCSpacing	Dimension	dynamic	CSG
XmNsubMenuId	XmCMenuWidget	Widget	NULL	CSG
XmNtearOffMenu-ActivateCallback	XmCCallback	XtCallbackList	NULL	C

XmNtearOffMenu-DeactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNtearOffModel	XmCTearOffModel	unsigned char	XmTEAR_OFF_-DISABLED	CSG
XmNtearOffTitle	XmCTearOffTitle	XmString	NULL	CSG
XmNunmapCallback	XmCCallback	XtCallbackList	NULL	C
XmNwhichButton	XmCWhichButton	unsigned int	dynamic	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**(3) 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 menu pane or a Pulldown menu pane, alignment is not performed; the Label is treated as the title within the menu pane, 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
When **XmNrowColumnType** is set to **XmMENU_OPTION**, this resource points to a text string that displays the label with respect to the selection area. The positioning of the label relative to the selection area depends on the layout direction in the horizontal orientation. 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 menu panes, 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 menu panes, 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 menu pane 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 menu pane, when the accelerator is typed by the user, the Popup menu pane 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 menu pane is osfMenu. The default for a MenuBar is osfMenuBar. 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**. Which corner of the MenuBar the Help widget is placed at depends on the **XmNlayoutDirection** resource of the widget. 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 Btn1 **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. Therefore, this resource cannot be **BTransfer Press**.

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 menu pane. 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 within the widget. In a row-major layout the children of the RowColumn are laid out in rows. The direction of the horizontal layout in the row-major layout (from left or right), and the wrapping in the column-major layout (vertical), depend on the **XmNlayoutDirection** resource of the widget. 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 menu pane and any of its submenus. The Popup menu pane needs to be informed whenever its accessibility to the user changes because posting of the Popup menu pane is controlled by the application. This resource can take four values, including:

XmPOPUP_KEYBOARD

Specifies that the keyboard input—accelerators and mnemonics—defined for the Popup menu pane and any of its submenus is enabled. This is the default.

XmPOPUP_DISABLED
Specifies that the keyboard input is disabled.

XmPOPUP_AUTOMATIC
Specifies that the keyboard is enabled for automatic popup menus.

XmPOPUP_AUTOMATIC_RECURSIVE
Specifies that the keyboard is enabled for recursive automatic popup menus.

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 menu pane 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 menu pane is about to be activated. It precedes the tear-off's map callback.

Use this resource when your application has shared menu panes 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 menu pane is about to be deactivated. It follows the tear-off's unmap callback.

Use this resource when your application has shared menu panes 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.

XmNtearoffTitle
Used by the TearOff shell to set the title for the window manager to display.

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 Motif. This resource cannot be 2.

XmRowColumn Constraint Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNpositionIndex	XmCPositionIndex	short	XmLAST_POSITION	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**	**Type**	**Default**	**Access**
XmNbuttonAccelerators	XmCButtonAccelerators	StringTable	NULL	C
XmNbuttonAccelerator- Text	XmCButton-AcceleratorText	XmStringTable	NULL	C
XmNbuttonCount	XmCButtonCount	int	0	C
XmNbuttonMnemonic-CharSets	XmCButtonMnemonic-CharSets	XmStringCharSetTable	NULL	C
XmNbuttonMnemonics	XmCButtonMnemonics	XmKeySymTable	NULL	C
XmNbuttons	XmCButtons	XmStringTable	NULL	C
XmNbuttonSet	XmCButtonSet	int	−1	C
XmNbuttonType	XmCButtonType	XmButtonTypeTable	NULL	C
XmNoptionLabel	XmCOptionLabel	XmString	NULL	C
XmNoptionMnemonic	XmCOptionMnemonic	KeySym	NULL	C
XmNpostFromButton	XmCPostFromButton	int	−1	C
XmNsimpleCallback	XmCCallback	XtCallbackProc	NULL	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 menu pane, or Pulldown menu pane.

XmCHECKBUTTON

Specifies a ToggleButtonGadget for a CheckBox, Popup menu pane, or Pulldown menu pane.

XmDOUBLE_SEPARATOR
Specifies a SeparatorGadget for a Popup menu pane, Pulldown menu pane, or OptionMenu Pulldown submenu. The separator type is **XmDOUBLE_LINE**.

XmPUSHBUTTON
Specifies a PushButtonGadget for a Popup menu pane, Pulldown menu pane, or OptionMenu Pulldown submenu.

XmRADIOBUTTON
Specifies a ToggleButtonGadget for a RadioBox, Popup menu pane, or Pulldown menu pane.

XmSEPARATOR
Specifies a SeparatorGadget for a Popup menu pane, Pulldown menu pane, or OptionMenu Pulldown submenu.

XmTITLE
Specifies a LabelGadget used as a title for a Popup menu pane or Pulldown menu pane.

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 menu pane for an OptionMenu.

XmNpostFromButton
This resource is for use with the simple menu creation routines. For a Pulldown menu pane, 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 menu pane. 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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottomShadow- Color	Pixel	dynamic	CSG
XmNbottomShadow- Pixmap	XmCBottomShadow- Pixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigationType	dynamic	CSG
XmNpopupHandler- Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadow- Thickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	dynamic	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources- Persistent	XmCInitialResources- Persistent	Boolean	True	C
XmNmappedWhen- Managed	XmCMappedWhen- Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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, osfActivate, and osfCancel translations from **XmManager** and has the following additional translations.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

<Btn2Down>:
MenuGadgetDrag()

c<Btn1Down>:
MenuGadgetTraverseCurrent()

c<Btn1Up>:
MenuGadgetTraverseCurrentUp()

≈c<BtnDown>:
MenuBtnDown()

≈c<BtnUp>:
MenuBtnUp()

:<Key>osfHelp:
MenuHelp()

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, Btn3 also performs the Btn1 actions.

:<Key>osfHelp:
MenuHelp()

:<Key>osfLeft:
MenuGadgetTraverseLeft()

:<Key>osfRight:
MenuGadgetTraverseRight()

:<Key>osfUp:
MenuGadgetTraverseUp()

:<Key>osfDown:
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 osfSelect. 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 Btn1Down. 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 Btn1Up. 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. The direction of the wrapping depends on the **XmNlayoutDirection** resource.

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.

The preceding description applies when the **XmNlayoutDirection** horizontal direction is **XmLEFT_TO_RIGHT**. If the **XmNlayoutDirection** horizontal direction is **XmRIGHT_TO_LEFT**, then the following applies.

If 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 a CascadeButton, then this action posts its associated submenu. If the current menu item is not a CascadeButton and is not at the left edge of a MenuPane, this action disarms the current item and arms the item to its left, wrapping if necessary. If the current menu item is not a CascadeButton and is at the left 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 left. 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 left edge of a row (except the bottom row), this action wraps to the rightmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, leftmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, rightmost menu item of 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.

The preceding description applies when the **XmNlayoutDirection** horizontal direction is **XmLEFT_TO_RIGHT**. If the **XmNlayoutDirection** horizontal direction is **XmRIGHT_TO_LEFT**, then the following applies. 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 right edge of a MenuPane, this action disarms the current item and arms the item to its right. If the current menu item is at the right edge of a submenu attached to a MenuBar item, then this action unposts the submenu and traverses to the MenuBar item to the right, 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 right 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 right edge, this action wraps within the MenuPane. If the current menu item is at the right edge of the MenuPane and not in the top row, this action wraps to the leftmost menu item in the row above. If the current menu item is in the upper, rightmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, leftmost menu item in 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. The direction of the wrapping depends on the **XmNlayoutDirection** resource.

Related Behavior

The following menu functions are available:

osfMenuBar:
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.

osfMenu: 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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateMenuBar**(3), **XmCreateOptionMenu**(3), **XmCreatePopupMenu**(3), **XmCreatePulldownMenu**(3), **XmCreateRadioBox**(3), **XmCreateRowColumn**(3), **XmCreateSimpleCheckBox**(3), **XmCreateSimpleMenuBar**(3), **XmCreateSimpleOptionMenu**(3), **XmCreateSimplePopupMenu**(3), **XmCreateSimplePulldownMenu**(3), **XmCreateSimpleRadioBox**(3), **XmCreateWorkArea**(3), **XmGetMenuCursor**(3), **XmGetPostedFromWidget**(3), **XmGetTearOffControl**, **XmLabel**(3), **XmManager**(3), **XmMenuPosition**(3), **XmOptionButtonGadget**(3), **XmOptionLabelGadget**(3), **XmRepTypeInstallTearOffModelConverter**, **XmSetMenuCursor**(3), **XmUpdateDisplay**(3), **XmVaCreateSimpleCheckBox**(3), **XmVaCreateSimpleMenuBar**(3), **XmVaCreateSimpleOptionMenu**(3), **XmVaCreateSimplePopupMenu**(3), **XmVaCreateSimplePulldownMenu**(3), and **XmVaCreateSimpleRadioBox**(3).

XmScale

Purpose
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. The placement of this label depends on the **XmNlayoutDirection** resource of the widget.

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 **XmScale** resource **XmNeditable** 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**. The placement of the title depends on the **XmNlayoutDirection** resource of the widget. The direction of the title is based on the widget's layout direction.

Scale uses the *XmQTspecifyRenderTable* trait, and holds the *XmQTtransfer* trait.

Data Transfer Behavior

Scale supports dragging of the representation of the Scale value from the Scale when the value is displayed and when the value of the **XmNenableUnselectableDrag** resource of **XmDisplay** is set to True.

As a source of data, Scale supports the following targets and associated conversions of data to these targets:

COMPOUND_TEXT
: The widget transfers a string representation of **XmNvalue** as type *COMPOUND_TEXT*.

STRING
: The widget transfers a string representation of **XmNvalue** as type *STRING*.

_MOTIF_CLIPBOARD_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets it supports for the *CLIPBOARD* selection. These include *STRING* and *COMPOUND_TEXT*.

_MOTIF_EXPORT_TARGETS
: The widget transfers, as type *ATOM*, a list of the targets to be used as the value of the DragContext's **XmNexportTargets** in a drag-and-drop transfer. These include *STRING* and *COMPOUND_TEXT*.

As a source of data, Scale also supports the following standard Motif targets:

BACKGROUND
: The widget transfers **XmNbackground** as type *PIXEL*.

CLASS
: The widget finds the first shell in the widget hierarchy that has a WM_CLASS property and transfers the contents as text in the current locale.

CLIENT_WINDOW
: The widget finds the first shell in the widget hierarchy and transfers its window as type *WINDOW*.

COLORMAP
: The widget transfers **XmNcolormap** as type *COLORMAP*.

FOREGROUND
: The widget transfers **XmNforeground** as type *PIXEL*.

NAME
: The widget finds the first shell in the widget hierarchy that has a WM_NAME property and transfers the contents as text in the current locale.

TARGETS
: The widget transfers, as type *ATOM*, a list of the targets it supports. These include the standard targets in this list. These also include *STRING* and *COMPOUND_TEXT*.

TIMESTAMP

The widget transfers the timestamp used to acquire the selection as type *INTEGER*.

_MOTIF_RENDER_TABLE

The widget transfers **XmNrenderTable** if it exists, or else the default text render table, as type *STRING*.

_MOTIF_ENCODING_REGISTRY

The widget transfers its encoding registry as type *STRING*. The value is a list of NULL separated items in the form of tag encoding pairs. This target symbolizes the transfer target for the Motif Segment Encoding Registry. Widgets and applications can use this Registry to register text encoding formats for specified render table tags. Applications access this Registry by calling **XmRegisterSegmentEncoding** and **XmMapSegmentEncoding**.

Descendants

Scale automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
Scrollbar	**XmScrollBar**	scroll bar
Title	**XmLabelGadget**	title of scale

Classes

Scale inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNdecimalPoints	XmCDecimalPoints	short	0	CSG
XmNdragCallback	XmCCallback	XtCallbackList	NULL	C
XmNeditable	XmCEditable	Boolean	True	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOnEnter	Boolean	False	CSG
XmNhighlightThickness	XmCHighlight- Thickness	Dimension	2	CSG
XmNmaximum	XmCMaximum	int	100	CSG
XmNminimum	XmCMinimum	int	0	CSG
XmNorientation	XmCOrientation	unsigned char	XmVERTICAL	CSG
XmNprocessing- Direction	XmCProcessing- Direction	unsigned char	dynamic	CSG
XmNscaleHeight	XmCScaleHeight	Dimension	0	CSG
XmNscaleMultiple	XmCScaleMultiple	int	dynamic	CSG
XmNscaleWidth	XmCScaleWidth	Dimension	0	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNshowArrows	XmCShowArrows	XtEnum	XmNONE	CSG
XmNshowValue	XmCShowValue	XtEnum	XmNONE	CSG
XmNsliderMark	XmCSliderMark	XtEnum	dynamic	CSG
XmNsliderVisual	XmCSliderVisual	XtEnum	dynamic	CSG
XmNslidingMode	XmCSlidingMode	XtEnum	XmSLIDER	CSG
XmNtitleString	XmCTitleString	XmString	NULL	CSG
XmNvalue	XmCValue	int	dynamic	CSG
XmNvalueChanged- Callback	XmCCallback	XtCallbackList	NULL	C

XmNconvertCallback
Specifies a list of callbacks called when the Scale is asked to convert a selection. The type of the structure whose address is passed to these callbacks is **XmConvertCallbackStruct**. The reason is **XmCR_OK**.

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**.

XmNeditable
Specifies how the Scale scrollbar will react to user input. This resource can be True or False values, as follows:

True Allows the scrollbar to be sensitive to user input. This is the default value.

False Makes the Scale scrollbar insensitive to user input. The visual is not greyed out. This value would mostly be used in **XmTHERMOMETER** mode.

When **XmNeditable** is used on a widget it sets the dropsite to **XmDROP_SITE_ACTIVE**.

XmNfontList
Specifies the font list to use for the title text string specified by **XmNtitleString**, and the label displayed when **XmNshowValue** is True. The font list is an obsolete structure, and is retained only for compatibility with earlier releases of Motif. See the **XmNrenderTable** resource.

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 Scale is oriented vertically, the default value is **XmMAX_ON_TOP**. If the XmScale is oriented horizontally, the default value depends on the **XmNlayoutDirection** resource of the widget.

XmNrenderTable

Specifies the render table 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 holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the render table is initialized to the **XmLABEL_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent. If a font list (**XmNfontList**) and a render table are both specified, the render table will take precedence. Refer to **XmRenderTable**(3) for more information on the creation and structure of a render table.

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.

XmNshowArrows
Specifies whether the arrows are displayed and how they are to be displayed. This resource can take the following values:

XmEACH_SIDE
Indicates that one arrow is displayed on each end of the ScrollBar slider.

XmMAX_SIDE
Indicates that one arrow is displayed on the **XmNmaximum** side of the ScrollBar slider.

XmMIN_SIDE
Indicates that one arrow is displayed on the **XmNminimum** side of the ScrollBar slider.

XmNONE Indicates that no arrows are displayed.

XmNONE is the default value.

XmNshowValue
Specifies whether a label for the current slider value should be displayed next to the slider. If the value is **XmNEAR_SLIDER**, the current slider value is displayed. If the value is **XmNONE**, no slider value is displayed. If the value is **XmNEAR_BORDER**, the current slider value is displayed near the border.

XmNsliderMark

Specifies the shape the slider is to be displayed in. This resource can take the following values:

XmETCHED_LINE
: Specifies the slider as an etched line. This is the default when **XmNslidingMode** is **XmSLIDER**.

XmNONE
: Specifies the slider as a foregrounded rectangle. This is the default when **XmNslidingMode** is **XmTHERMOMETER** and the Scale scrollbar is insensitive to user input (**XmNeditable** is **False**.

XmROUND_MARK
: Specifies the slider as a shadowed circle. This is the default when **XmNslidingMode** is **XmTHERMOMETER** and the Scale scrollbar is sensitive to user input (**XmNeditable** is **True**.

XmTHUMB_MARK
: Specifies the slider as a series of three etched lines centered in the middle of the slider.

XmNslidingMode

Specifies the mode the slider works in. There are two possible modes:

XmSLIDER
: Allows the slider to move freely between the minimum and maximum ends of the scale. This is the default value.

XmTHERMOMETER
: Forces the slider to be anchored to one side of the trough area.

XmNsliderVisual

Specifies the color of the slider visual. This resource can take the following values:

XmBACKGROUND_COLOR
: Specifies that the slider visual is in the background color.

XmFOREGROUND_COLOR
: Specifies that the slider visual is in the foreground color.

XmSHADOWED_BACKGROUND
Specifies that the slider visual is in the background color, with a shadow. This is the default when the *XmNslidingModel* resource is **XmSLIDER**.

XmTROUGH_COLOR
Specifies that the slider visual is in the trough color. This is the default when the *XmNslidingModel* resource is **XmTHERMOMETER**.

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 (zero) 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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottomShadow-Color	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG

XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler- Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG
XmNstringDirection	XmCStringDirection	XmString- Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG

XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Callback Information

A pointer to the following structure is passed to the **XmNdragCallback** and **XmNvalueChangedCallback** procedures:

```
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

A pointer to the following structure is passed to the **XmNconvertCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent *event;
        Atom selection;
        Atom target;
        XtPointer source_data;
        XtPointer location_data;
        int flags;
        XtPointer parm;
        int parm_format;
        unsigned long parm_length;
        int status;
        XtPointer value;
        Atom type;
        int format;
        unsigned long length;
} XmConvertCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which conversion is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

target Indicates the conversion target.

source_data Contains information about the selection source. When the selection is _MOTIF_DROP, *source_data* is the DragContext. Otherwise, it is NULL.

location_data
Contains information about the location of data to be converted. If the value is NULL, the data to be transferred consists of the widget's current selection.

flags Indicates the status of the conversion. Following are the possible values:

XmCONVERTING_NONE
This flag is currently unused.

XmCONVERTING_PARTIAL
The target widget was able to be converted, but some data was lost.

XmCONVERTING_SAME
The conversion target is the source of the data to be transferred.

XmCONVERTING_TRANSACT
This flag is currently unused.

parm Contains parameter data for this target. If no parameter data exists, the value is NULL.

When *selection* is *CLIPBOARD* and *target* is _MOTIF_CLIPBOARD_TARGETS or _MOTIF_DEFERRED_CLIPBOARD_TARGETS, the value is the requested operation (**XmCOPY**, **XmMOVE**, or **XmLINK**).

parm_format
Specifies whether the data in *parm* should be viewed as a list of *char*, *short*, or *long* quantities. Possible values are 0 (when *parm* is NULL), 8 (when the data in *parm* should be viewed as a list of *char*s), 16 (when the data in *parm* should be viewed as a list of *short*s), or 32 (when the data in *parm* should be viewed as a list of *long*s). Note that *parm_format* symbolizes a data type, not the number of bits in each list element. For example, on some machines, a *parm_format* of 32 means that the data in *parm* should be viewed as a list of 64-bit quantities, not 32-bit quantities.

parm_length Specifies the number of elements of data in *parm*, where each element has the size specified by *parm_format*. When *parm* is NULL, the value is 0.

status An IN/OUT member that specifies the status of the conversion. The initial value is **XmCONVERT_DEFAULT**. The callback procedure can set this member to one of the following values:

XmCONVERT_DEFAULT
This value means that the widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it overwrites the data provided by the callback procedures in the *value* member.

XmCONVERT_MERGE
This value means that the widget class conversion procedure, if any, is called after the callback procedures return. If the widget class conversion procedure produces any data, it appends its data to the data provided by the callback procedures in the *value* member. This value is

intended for use with targets that result in lists of data, such as *TARGETS*.

XmCONVERT_DONE
This value means that the callback procedure has successfully finished the conversion. The widget class conversion procedure, if any, is not called after the callback procedures return.

XmCONVERT_REFUSE
This value means that the callback procedure has terminated the conversion process without completing the requested conversion. The widget class conversion procedure, if any, is not called after the callback procedures return.

value An IN/OUT parameter that contains any data that the callback procedure produces as a result of the conversion. The initial value is NULL. If the callback procedure sets this member, it must ensure that the *type*, *format*, and *length* members correspond to the data in *value*. The callback procedure is responsible for allocating, but not for freeing, memory when it sets this member.

type An IN/OUT parameter that indicates the type of the data in the *value* member. The initial value is *INTEGER*.

format An IN/OUT parameter that specifies whether the data in *value* should be viewed as a list of *char*, *short*, or *long* quantities. The initial value is 8. The callback procedure can set this member to 8 (for a list of *char*), 16 (for a list of *short*), or 32 (for a list of *long*).

length An IN/OUT member that specifies the number of elements of data in *value*, where each element has the size symbolized by *format*. The initial value is 0.

Behavior

XmScale has the following behavior:

Btn1Down or Btn2Down:
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.

Btn2Down in value label:
Drags the contents of the label showing the current slider value. This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures, possibly multiple times, for the _MOTIF_DROP selection.

Btn1Motion or Btn2Motion:
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**.

Btn1Up or Btn2Up:
If the button press occurs within the slider and the slider position is changed, the callbacks for **XmNvalueChangedCallback** are called.

CtrlBtn1Down:
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.

KeyosfUp: 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.

KeyosfDown:
: 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.

KeyosfLeft:
: 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.

KeyosfRight:
: 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.

CtrlKeyosfUp or KeyosfPageUp:
: 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.

CtrlKeyosfDown or KeyosfPageDown:
: 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.

CtrlKeyosfLeft or KeyosfPageLeft:
: 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.

CtrlKeyosfRight or KeyosfPageRight:
: 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.

KeyosfBeginLine or KeyosfBeginData:
: Moves the slider to the minimum value and calls the **XmNvalueChangedCallback** callbacks.

KeyosfEndLine or KeyosfEndData:
: Moves the slider to the maximum value and calls the **XmNvalueChangedCallback** callbacks.

KeyosfNextField:
: 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.

KeyosfPrevField:
: 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.

KeyosfHelp:
: 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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateScale**(3), **XmManager**(3), **XmScaleGetValue**(3), and **XmScaleSetValue**(3).

XmScreen

Purpose

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**	**Type**	**Default**	**Access**
XmNbitmapConversion-Model	XmCBitmap-ConversionModel	XtEnum	XmPIXMAP	CSG??
XmNcolorAllocation- Proc	XmCColor-AllocationProc	XtProc	NULL	CSG??
XmNcolorCalculation- Proc	XmCColor-CalculationProc	XtProc	NULL	CSG??
XmNdarkThreshold	XmCDarkThreshold	int	dynamic	C
XmNdefaultCopy-CursorIcon	XmCDefaultCopy-CursorIcon	Widget	NULL	CSG
XmNdefaultInvalid-CursorIcon	XmCDefaultInvalid-CursorIcon	Widget	NULL	CSG
XmNdefaultLink-CursorIcon	XmCDefaultLink- -CursorIcon	Widget	NULL	CSG
XmNdefaultMove-CursorIcon	XmCDefaultMove-CursorIcon	Widget	NULL	CSG
XmNdefaultNone-CursorIcon	XmCDefaultNone-CursorIcon	Widget	NULL	CSG
XmNdefaultSource-CursorIcon	XmCDefaultSource-CursorIcon	Widget	NULL	CSG
XmNdefaultValid-CursorIcon	XmCDefaultValid-CursorIcon	Widget	NULL	CSG
XmNfont	XmCFont	XFontStruct *	NULL	CSG
XmNforeground- Threshold	XmCForeground-Threshold	int	dynamic	C
XmNhorizontalFontUnit	XmCHorizontal-FontUnit	int	dynamic	CSG

XmNinsensitiveStipple-Bitmap	XmCinsensitiveStipple-Bitmap	Bitmap	"50_foreground"	CSG
XmNlightThreshold	XmCLightThreshold	int	dynamic	C
XmNmenuCursor	XmCCursor	Cursor	arrow	C
XmNmoveOpaque	XmCMoveOpaque	Boolean	False	CSG
XmNunpostBehavior	XmCUnpostBehavior	unsigned char	XmUNPOST_-AND_REPLAY	CSG
XmNuseColorObject	XmCUseColorObject	Boolean	False	C
XmNuserData	XmCUserData	XtPointer	NULL	CSG
XmNverticalFontUnit	XmCVertical- FontUnit	int	dynamic	CSG

XmNbitmapConversionModel
Provides a policy for the conversion of xbm and xpm files to the **Pixmap** type. This resource takes the following values:

XmMATCH_DEPTH
From a supplied xbm or xpm file, generates a converted pixmap file having the same depth as the widget.

XmDYNAMIC_DEPTH
Converts an input xbm file to a **Pixmap** of depth 1, or converts an input xpm file to a **Pixmap** having the same depth as the widget.

XmNcolorAllocationProc
Identifies the procedure to be used for color allocation. Normally, this procedure is an application-defined color allocation procedure. However, if no application-defined color allocation procedure is set, the system uses Motif's predefined color allocation procedure.

XmNcolorCalculationProc
Identifies the procedure to be used for per-widget color calculation. Normally, this procedure is an application-defined color calculation procedure. However, if no application-defined color calculation procedure is set, the system uses Motif's predefined color calculation procedure.

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. The system default icon is determined by the Display resource **XmNenableDragIcon**.

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. The system default icon is determined by the Display resource **XmNenableDragIcon**.

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. The system default icon is determined by the Display resource **XmNenableDragIcon**.

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. The system default icon is determined by the Display resource **XmNenableDragIcon**.

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. The system default icon is determined by the Display resource **XmNenableDragIcon**.

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. The system default icon is determined by the Display resource **XmNenableDragIcon**.

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. The system default icon is determined by the Display resource **XmNenableDragIcon**.

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**. Note that this resource is used only for the calculation of the font unit values. To specify a font to be used to display text, use a widget's render table resource (**XmNrenderTable**).

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.

XmNinsensitiveStippleBitmap
Provides widgets with the bitmap to use when generating the insensitive visual. This bitmap is to be used as the stipple for the rendering of insensitive visuals.

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	**leftbutton**
arrow	**ll_angle**
based_arrow_down	**lr_angle**
based_arrow_up	**man**
boat	**middlebutton**
bogosity	**mouse**
bottom_left_corner	**pencil**
bottom_right_corner	**pirate**
bottom_side	**plus**
bottom_tee	**question_arrow**
box_spiral	**right_ptr**
center_ptr	**right_side**
circle	**right_tee**
clock	**rightbutton**
coffee_mug	**rtl_logo**
cross	**sailboat**
cross_reverse	**sb_down_arrow**
crosshair	**sb_h_double_arrow**
diamond_cross	**sb_left_arrow**
dot	**sb_right_arrow**
dotbox	**sb_up_arrow**
double_arrow	**sb_v_double_arrow**
draft_large	**shuttle**
draft_small	**sizing**
draped_box	**spider**
exchange	**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.

XmNuseColorObject
Enables and disables the sharing of colors between widgets, and the dynamic changing of colors. A value of False disables this, and a value of True enables it.

XmNuserData
Allows the application to attach any necessary specific data to the widget. This is an internally unused resource.

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(3), **XmDisplay**(3), **XmGetXmScreen**(3), and **XmSetMenuCursor**(3),

XmScrollBar

Purpose

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.

If the ScrollBar parent is an automatic ScrolledWindow, the **XmNtraversalOn** default is True. Otherwise, the default is False.

ScrollBar holds the *XmQTnavigator* traits.

Classes

ScrollBar inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNdecrementCallback	XmCCallback	XtCallbackList	NULL	C
XmNdragCallback	XmCCallback	XtCallbackList	NULL	C
XmNeditable	XmCEditable	Boolean	True	CSG
XmNincrement	XmCIncrement	int	1	CSG
XmNincrementCallback	XmCCallback	XtCallbackList	NULL	C
XmNinitialDelay	XmCInitialDelay	int	250 ms	CSG
XmNmaximum	XmCMaximum	int	100	CSG
XmNminimum	XmCMinimum	int	0	CSG
XmNorientation	XmCOrientation	unsigned char	XmVERTICAL	CSG
XmNpageDecrement-Callback	XmCCallback	XtCallbackList	NULL	C
XmNpageIncrement	XmCPageIncrement	int	10	CSG
XmNpageIncrement-Callback	XmCCallback	XtCallbackList	NULL	C
XmNprocessingDirection	XmCProcessing-Direction	unsigned char	dynamic	CSG
XmNrepeatDelay	XmCRepeatDelay	int	50 ms	CSG
XmNshowArrows	XmCShowArrows	XtEnum	XmEACH_SIDE	CSG
XmNsliderSize	XmCSliderSize	int	dynamic	CSG
XmNsliderMark	XmCSliderMark	XtEnum	dynamic	CSG
XmNsliderVisual	XmCSliderVisual	XtEnum	XmSHADOWED	CSG

XmNslidingMode	XmCSlidingMode	XtEnum	XmSLIDER	CSG
XmNsnapBackMultiple	XmCSnapBack-Multiple	unsigned short	MaxValue	CSG
XmNtoBottomCallback	XmCCallback	XtCallbackList	NULL	C
XmNtoTopCallback	XmCCallback	XtCallbackList	NULL	C
XmNtroughColor	XmCTroughColor	Pixel	dynamic	CSG
XmNvalue	XmCValue	int	dynamic	CSG
XmNvalueChanged-Callback	XmCCallback	XtCallbackList	NULL	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**.

XmNeditable

Specifies how ScrollBar will react to user input. This resource can be True or False values, as follows:

True Allows the scrollbar to be sensitive to user input. This is the default value.

False Makes the Scale scrollbar insensitive to user input. The visual is not greyed out. This value would mostly be used in **XmTHERMOMETER** mode.

When **XmNeditable** is used on a widget it sets the dropsite to **XmDROP_SITE_ACTIVE**.

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. **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 ScrollBar is oriented vertically, the default value is **XmMAX_ON_BOTTOM**. If the ScrollBar is oriented horizontally, the default value depends on the **XmNlayoutDirection** resource of the widget.

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 and how they are to be displayed. This resource can take the following values:

XmEACH_SIDE

Indicates that one arrow is displayed on each end of the ScrollBar slider. This corresponds to a value of True in previous releases.

XmMAX_SIDE

Indicates that both arrows are displayed on the **XmNmaximum** side of the ScrollBar slider.

XmMIN_SIDE

Indicates that both arrows are displayed on the **XmNminimum** side of the ScrollBar slider.

XmNONE Indicates that no arrows are displayed. This corresponds to a value of False in previous releases.

XmEACH_SIDE is the default value.

XmNsliderMark
Specifies the shape the slider is to be displayed in. This resource can take the following values:

XmETCHED_LINE
Specifies the slider as an etched line.

XmNONE
Specifies the slider as a foregrounded rectangle. This is the default for a regular slider.

XmROUND_MARK
Specifies the slider as a shadowed circle. This is the default when the slider is a thermometer.

XmTHUMB_MARK
Specifies the slider as a series of three etched lines centered in the middle of the slider.

XmNslidingMode
Specifies the mode the slider works in. There are two possible modes:

XmSLIDER
Allows the slider to move freely between the minimum and maximum ends of the scale. This is the default value.

XmTHERMOMETER
Forces the slider to be anchored to one side of the trough area.

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.

XmNsliderVisual
Specifies the color of the slider visual. This resource can take the following values:

XmBACKGROUND_COLOR
Specifies that the slider visual is in the background color.

XmFOREGROUND_COLOR
Specifies that the slider visual is in the foreground color.

XmSHADOWED_BACKGROUND
Specifies that the slider visual is in the background color, with a shadow. This is the default for a regular slider.

XmTROUGH_COLOR
Specifies that the slider visual is in the trough color. This is the default when the slider is a thermometer.

XmNsnapBackMultiple
Specifies the distance over which the scrollbar slider snaps back to its original position when the user drags the mouse outside the ScrollBar edge. This distance is defined in terms of multiples of the width of the slider. For example, a multiple of 0 (zero) causes the slider to snap back as soon as the pointer moves out of the ScrollBar frame, a multiple of 1 causes the slider to snap back as soon as the pointer moves beyond 1 ScrollBar width of the ScrollBar edge. Whenever the slider snaps back, the ScrollBar **dragCallback** is called if there is one.

The default value is large enough to prevent unwanted snapBack activity if the mouse is moved within the boundaries of any reasonable screen. To reset the default, set this resource to a large value, such as 10000.

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. This color defaults to the color used for selections.

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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOn-Enter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmSTICKY_TAB_-GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadow-Thickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadow-Color	Pixel	dynamic	CSG

XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	dynamic	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

≈s ≈c ≈m ≈a <Btn1Down>:
Select()

≈s ≈c ≈m ≈a <Btn1Up>:
Release()

≈s ≈c ≈m ≈a Button1PtrMoved:
Moved()

≈s ≈c ≈m ≈a <Btn2Down>:
Select()

≈s ≈c ≈m ≈a <Btn2Up>:
: **Release()**

≈s ≈c ≈m ≈a Button2PtrMoved:
: **Moved()**

≈s c ≈m ≈a <Btn1Down>:
: **TopOrBottom()**

≈s c ≈m ≈a <Btn1Up>:
: **Release()**

:<Key>osfActivate:
: **PrimitiveParentActivate()**

:<Key>osfCancel:
: **CancelDrag()**

:<Key>osfBeginLine:
: **TopOrBottom()**

:<Key>osfEndLine:
: **TopOrBottom()**

:<Key>osfPageLeft:
: **PageUpOrLeft(***Left***)**

:c <Key>osfPageUp:
: **PageUpOrLeft(***Left***)**

:<Key>osfPageUp:
: **PageUpOrLeft(***Up***)**

:<Key>osfPageRight:
: **PageDownOrRight(***Right***)**

:c <Key>osfPageDown:
: **PageDownOrRight(***Right***)**

:<Key>osfPageDown:
: **PageDownOrRight(***Down***)**

:<Key>osfHelp:
: **PrimitiveHelp()**

:c <Key>osfUp:
: **PageUpOrLeft(***Up***)**

:<Key>osfUp:
IncrementUpOrLeft(*Up***)**

:c <Key>osfDown:
PageDownOrRight(*Down***)**

:<Key>osfDown:
IncrementDownOrRight(*Down***)**

:c <Key>osfLeft:
PageUpOrLeft(*Left***)**

:<Key>osfLeft:
IncrementUpOrLeft(*Left***)**

:c <Key>osfRight:
PageDownOrRight(*Right***)**

:<Key>osfRight:
IncrementDownOrRight(*Right***)**

≈s ≈m ≈a <Key>Return:
PrimitiveParentActivate()

s ≈m ≈a <Key>Tab:
PrimitivePrevTabGroup()

≈m ≈a <Key>Tab:
PrimitiveNextTabGroup()

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(*Down*|*Right*):
With an argument of **Down**, or 0 (zero) for compatibility, moves the slider down by one increment. With an argument of **Right**, or 1 for compatibility, 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(*Up*|*Left*):
: With an argument of **Up**, or 0 (zero) for compatibility, moves the slider up by one increment. With an argument of **Left**, or 1 for compatibility, 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(*Down*|*Right*):
: With an argument of **Down**, or 0 (zero) for compatibility, moves the slider down by one page increment. With an argument of **Right**, or 1 for compatibility, 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(*Up*|*Left*):
: With an argument of **Up**, or 0 (zero) for compatibility, moves the slider up by one page increment. With an argument of **Left**, or 1 for compatibility, it moves 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():
CtrlBtn1Down 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 KeyosfBeginLine or KeyosfBeginData moves the slider to the minimum value and invokes the **XmNtoTopCallback**. Pressing KeyosfEndLine or KeyosfEndData 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**(3).

Related Information

Core(3), **XmCreateScrollBar**(3), **XmPrimitive**(3), **XmScrollBarGetValues**(3), and **XmScrollBarSetValues**(3).

XmScrolledWindow

Purpose
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 and the clipping area of the ScrolledWindow widget. The names of these widgets are **HorScrollBar**, **VertScrollBar**, and **ClipWindow** respectively, and remain consistent whether created by **XmCreateScrolledList**, **XmCreateScrolledText** or **XmCreateScrolledWindow**.

ScrolledWindow uses the *XmQTnavigator* trait, and holds the *XmQTscrollFrame* trait.

Descendants

ScrolledWindow automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
VertScrollBar	**XmScrollBar**	vertical scroll bar
HorScrollBar	**XmScrollBar**	horizontal scroll bar
ClipWindow	**XmClipWindow**	clip window

Classes

ScrolledWindow inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNautoDragModel	XmCAutoDragModel	XtEnum	XmAUTO_DRAG_- ENABLED	CSG
XmNclipWindow	XmCClipWindow	Widget	dynamic	G
XmNhorizontal- ScrollBar	XmCHorizontal- ScrollBar	Widget	dynamic	CSG
XmNscrollBarDisplay- Policy	XmCScrollBar- DisplayPolicy	unsigned char	dynamic	CSG
XmNscrollBarPlacement	XmCScrollBarPlacement	unsigned char	XmBOTTOM_- RIGHT	CSG
XmNscrolledWindow- MarginHeight	XmCScrolledWindow- MarginHeight	Dimension	0	CSG
XmNscrolledWindow- MarginWidth	XmCScrolledWindow- MarginWidth	Dimension	0	CSG
XmNscrollingPolicy	XmCScrollingPolicy	unsigned char	XmAPPLICATION_- DEFINED	CG
XmNspacing	XmCSpacing	Dimension	4	CSG
XmNtraverseObscured- Callback	XmCCallback	XtCallbackList	NULL	CSG
XmNverticalScrollBar	XmCVerticalScroll- Bar	Widget	dynamic	CSG
XmNvisualPolicy	XmCVisualPolicy	unsigned char	dynamic	G
XmNworkWindow	XmCWorkWindow	Widget	NULL	CSG

XmNautoDragModel

Indicates whether automatic drag is enabled (**XmAUTO_DRAG_ENABLED**) or disabled (**XmAUTO_DRAG_DISABLED**). By default it is enabled.

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 depends on the value of the **XmNlayoutDirection** resource of the widget.

XmNscrolledWindowMarginHeight

Specifies the margin height on the top and bottom of the ScrolledWindow. In order to use the autoscroll drag feature of the Motif drag and drop facility, a user must be able to hold a drag icon over the margin of a scrolled window. Though drag and drop will work with the default margin size of zero, a user may find it difficult to position the icon precisely enough to use the feature easily. The application programmer should ensure that the window margins are set to an adequate size, if the use of the autoscroll drag feature is desired.

XmNscrolledWindowMarginWidth

Specifies the margin width on the right and left sides of the ScrolledWindow. Please refer to the warning concerning the default margin size for the **XmNscrolledWindowMarginHeight** resource, above.

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.

XmScrolledWindow Constraint Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNscrolledWindow-ChildType	XmCScrolledWindow-ChildType	unsigned char	RESOURCE_-DEFAULT	CSG

XmNscrolledWindowChildType
Specifies what the child is. ScrolledWindow supports a number of child types. The possible values are:

XmWORK_AREA
: Indicates a work area child. This specifies that both ScrollBars are limited to moving the child inside the clipping window. If the scrolling policy is **XmAUTOMATIC**, the work area child can move in both directions.

XmHOR_SCROLLBAR
: Indicates a horizontal child widget; the child must have the *XmQTnavigator* trait installed. For example, the **XmScrollBar** widget has the *XmQTnavigator* trait installed.

XmVERT_SCROLLBAR
: Indicates a vertical child widget; the child must have the *XmQTnavigator* trait installed.

XmSCROLL_HOR
: Indicates that only the horizontal ScrollBar moves the child. This value is only meaningful if the scrolling policy is **XmAUTOMATIC**.

XmSCROLL_VERT
: Indicates that only the vertical ScrollBar moves the child. This value is only meaningful if the scrolling policy is **XmAUTOMATIC**.

XmNO_SCROLL
: Indicates that the child does not move with the ScrollBars. This value is only meaningful if the scrolling policy is **XmAUTOMATIC**.

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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottom- ShadowColor	Pixel	dynamic	CSG

XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallback-List	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlight- Pixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallback-List	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString-Direction	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G

XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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 these conditions to be 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 so by virtue of ancestral **XmNtraverseObscuredCallback** routines; 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;
        XmTraversalDirection direction;
} 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:

KeyosfPageUp:
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.

KeyosfPageDown:
: 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.

KeyosfPageLeft:
: 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.

KeyosfPageRight:
: 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.

KeyosfBeginLine:
: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window horizontally to the edge corresponding to the horizontal ScrollBar's minimum value.

KeyosfEndLine:
: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window horizontally to the edge corresponding to the horizontal ScrollBar's maximum value.

KeyosfBeginData:
: If **XmNscrollingPolicy** is **XmAUTOMATIC**, scrolls the window vertically to the edge corresponding to the vertical ScrollBar's minimum value.

KeyosfEndData:
: 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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmCreateScrolledWindow**(3), **XmManager**(3), **XmProcessTraversal**(3), **XmScrollBar**(3), **XmScrollVisible**(3), and **XmScrolledWindowSetAreas**(3).

XmSelectionBox

Purpose

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. The *XmQTmenuSystem* trait is used to check that it is the first MenuBar child.

Buttons — All **XmPushButton** widgets or gadgets, and their subclasses are placed after the *OK* button in the order of their creation (this order is checked using the *XmQTactivatable* trait). The layout direction of the buttons depends on the **XmNlayoutDirection** resource.

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** or **TextField**

Selection Separator
: **Separator**

SelectionBox uses the *XmQTaccessTextual*, *XmQTactivatable*, and *XmQTmenuSystem* traits.

Descendants

SelectionBox automatically creates the descendants shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendant. In addition, a user or an application can use the named descendant when specifying resource values.

Named Descendant	**Class**	**Identity**
Apply	**XmPushButtonGadget**	Apply button
Cancel	**XmPushButtonGadget**	Cancel button
Help	**XmPushButtonGadget**	Help button
Items	**XmLabelGadget**	title above the list of items
ItemsList	**XmList**	list of items from which the user will select
ItemsListSW	**XmScrolledWindow**	ScrolledWindow parent of **ItemsList**
OK	**XmPushButtonGadget**	OK button
Selection	**XmLabelGadget**	title above the selection box
Separator	**XmSeparatorGadget**	dividing line between selection box and buttons
Text	**XmTextField**	selection box containing text of selected item

Classes

SelectionBox inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNapplyCallback	XmCCallback	XtCallbackList	NULL	C
XmNapplyLabelString	XmCApplyLabelString	XmString	dynamic	CSG
XmNcancelCallback	XmCCallback	XtCallbackList	NULL	C
XmNcancelLabelString	XmCCancelLabelString	XmString	dynamic	CSG
XmNchildPlacement	XmCChildPlacement	unsigned char	XmPLACE_ABOVE_-SELECTION	CSG
XmNdialogType	XmCDialogType	unsigned char	dynamic	CG
XmNhelpLabelString	XmCHelpLabelString	XmString	dynamic	CSG
XmNlistItemCount	XmCItemCount	int	0	CSG
XmNlistItems	XmCItems	XmStringTable	NULL	CSG
XmNlistLabelString	XmCListLabelString	XmString	dynamic	CSG
XmNlistVisibleItem- Count	XmCVisibleItemCount	int	dynamic	CSG
XmNminimizeButtons	XmCMinimizeButtons	Boolean	False	CSG
XmNmustMatch	XmCMustMatch	Boolean	False	CSG
XmNnoMatchCallback	XmCCallback	XtCallbackList	NULL	C
XmNokCallback	XmCCallback	XtCallbackList	NULL	C
XmNokLabelString	XmCOkLabelString	XmString	dynamic	CSG
XmNselectionLabel- String	XmCSelectionLabel- String	XmString	dynamic	CSG
XmNtextAccelerators	XmCTextAccelerators	XtAccelerators	default	C
XmNtextColumns	XmCColumns	short	dynamic	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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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*.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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**.

Now that some default localized label strings are provided through message catalogs for the children of composite widgets, the **labelString** resources cannot be set on the child through default resource files. Instead, the resource provided at the parent level must be used.

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**	**Type**	**Default**	**Access**
XmNallowOverlap	XmCAllowOverlap	Boolean	True	CSG
XmNautoUnmanage	XmCAutoUnmanage	Boolean	True	CG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNcancelButton	XmCWidget	Widget	dynamic	SG
XmNdefaultButton	XmCWidget	Widget	dynamic	SG
XmNdefaultPosition	XmCDefaultPosition	Boolean	True	CSG
XmNdialogStyle	XmCDialogStyle	unsigned char	dynamic	CSG
XmNdialogTitle	XmCDialogTitle	XmString	NULL	CSG
XmNfocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNmapCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	10	CSG

XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNnoResize	XmCNoResize	Boolean	False	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_-ANY	CSG
XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_-OUT	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNtextTranslations	XmCTranslations	XtTranslations	NULL	C
XmNunmapCallback	XmCCallback	XtCallbackList	NULL	C

XmManager Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottom- ShadowColor	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottom-ShadowPixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmTAB_GROUP	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG

XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	N/A
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMapped-WhenManaged	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG

XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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 This member is obsolete and exists for compatibility with earlier releases.

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.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

:<Key>osfUp:
SelectionBoxUpOrDown(*Previous***)**

:<Key>osfDown:
SelectionBoxUpOrDown(*Next***)**

:<Key>osfBeginLine:
  SelectionBoxUpOrDown(*First***)**

:<Key>osfEndLine:
  SelectionBoxUpOrDown(*Last***)**

:<Key>osfRestore:
  SelectionBoxRestore()

s c ≈m ≈a <Key>space:
  SelectionBoxRestore()

Action Routines

The XmSelectionBox action routines are

SelectionBoxUpOrDown(*Previous*|*Next*|*First*|*Last*):
  When called with an argument of **Previous**, or 0 (zero) for compatibility, selects the previous item in the list and replaces the text with that item.

  When called with an argument of **Next**, or 1 for compatibility, selects the next item in the list and replaces the text with that item.

  When called with an argument of **First**, or 2 for compatibility, selects the first item in the list and replaces the text with that item.

  When called with an argument of **Last**, or 3 for compatibility, 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:

KeyosfCancel:
  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.

KeyosfActivate:
  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 KeyosfActivate is called before the

SelectionBox actions associated with KeyosfActivate. In a multiline Text widget, any KeyosfActivate event except KeyosfEnter calls the Text action associated with KeyosfActivate, then the SelectionBox actions associated with KeyosfActivate. 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**(3).

Related Information

Composite(3), **Constraint**(3), **Core**(3), **XmBulletinBoard**(3), **XmCreateSelectionBox**(3), **XmCreateSelectionDialog**(3), **XmCreatePromptDialog**(3), **XmManager**(3), and **XmSelectionBoxGetChild**(3).

XmSeparator

Purpose

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 **XmNseparatorType** 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. Separator holds the *XmQTmenuSavvy* trait.

Classes

Separator inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNmargin	XmCMargin	Dimension	0	CSG
XmNorientation	XmCOrientation	unsigned char	XmHORIZONTAL	CSG
XmNseparatorType	XmCSeparatorType	unsigned char	XmSHADOW_ETCHED_IN	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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottomShadow- Color	Pixel	dynamic	CSG
XmNbottomShadow- Pixmap	XmCBottomShadow- Pixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallback- List	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallback- List	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlightThickness	Dimension	0	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmNONE	CSG
XmNpopupHandler- Callback	XmCCallback	XtCallback- List	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	False	G
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG

XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

Translations

There are no translations for **XmSeparator**.

Related Information

Core(3), **XmCreateSeparator**(3), and **XmPrimitive**(3).

XmSeparatorGadget

Purpose

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 **XmNseparatorType** 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. SeparatorGadget holds the *XmQTmenuSavvy* trait.

Classes

SeparatorGadget inherits behavior, resources, and traits 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**	**Type**	**Default**	**Access**
XmNmargin	XmCMargin	Dimension	0	CSG
XmNorientation	XmCOrientation	unsigned char	XmHORIZONTAL	CSG
XmNseparatorType	XmCSeparatorType	unsigned char	XmSHADOW_ETCHED_IN	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	Type	Default	Access
XmNbackground	XmCBackground	Pixel	dynamic	CSG

XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNbottomShadow- Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight-OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	0	CSG
XmNlayoutDirection	XmNCLayout-Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmNONE	CSG
XmNshadowThickness	XmCShadow-Thickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	False	G
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

RectObj Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNborderWidth	XmCBorderWidth	Dimension	0	N/A
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNsensitive	XmCSensitive	Boolean	True	CSG

XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

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

Behavior

XmSeparatorGadget has no behavior.

Related Information

Object(3), **RectObj**(3), **XmCreateSeparatorGadget**(3), and **XmGadget**(3).

XmText

Purpose

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.

In some Asian languages, texts are drawn vertically. Also, some characters are displayed with 90-degree clockwise rotation, and other characters are mapped to vertical glyphs that differ from the normal horizontal glyphs. Information about which characters require rotation or mapping to vertical glyphs is specified in the X Locale Database (NLS databases) and handled by X library, depending on **XNOrientation XOC** values. *XmText* widget should also handle the vertically aligned lines as for editing, entering, or selecting texts.

The vertical writing feature of the *XmText* widget is enabled when the *XmTOP_TO_BOTTOM* value is specified for the *XmNlayoutDirection* resource of the *XmText* widget. In that case, the horizontal scroll bar is displayed on the bottom of the *XmText* widget and the vertical scroll bar is displayed on the left side.

Mouse Selection

The Text widget allows text to be edited, inserted, and selected. The user can cut, copy, and paste text by 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.

Text uses the *XmQTnavigator*, *XmQTspecifyRenderTable*, and *XmQTscrollFrame* traits, and holds the *XmQTaccessTextual* and *XmQTtransfer* traits. The widget checks its parent for the *XmQTscrollFrame* trait. If this trait does not exist, then the widget has no scrolling. If the trait does exist, and the ScrollFrame widget has not been initialized, the widget creates two navigators and sets up the scrollbars.

If an application or widget calls the **setValue** trait method of *XmQTaccessTextual*, then **XmText** will call **XmTextSetString** to set the string value.

Classes

Text inherits behavior, resources, and traits from **Core** and **XmPrimitive**.

The class pointer is *xmTextWidgetClass*.

The class name is **XmText**.

Data Transfer Behavior

Text supports transfer of the primary, secondary, and clipboard selections and dragging of selected text from the widget. Text can also be the destination for the primary, secondary, and clipboard selections, and it supports dropping of data being dragged onto the widget.

When the **XmNconvertCallback** procedures are called, the **location_data** member of the **XmConvertCallbackStruct** member is NULL if the selected text is being transferred. If the entire text, not the selected text, is being transferred, the value of this member is the widget ID of the Text widget.

As a source of data, Text supports the following targets and associated conversions of data to these targets:

locale If the *locale* target matches the widget's locale, the widget transfers the selected text in the encoding of the locale.

COMPOUND_TEXT
The widget transfers the selected text as type *COMPOUND_TEXT*.

STRING The widget transfers the selected text as type *STRING*.

TEXT If the selected text is fully convertible to the encoding of the locale, the widget transfers the selected text in the encoding of the locale. Otherwise, the widget transfers the selected text as type *COMPOUND_TEXT*.

DELETE The widget deletes the selected text.

_MOTIF_CLIPBOARD_TARGETS
The widget transfers, as type *ATOM*, a list of the targets to which the widget can convert data to be placed on the clipboard immediately. If the selected text is fully convertible to *STRING*, these include *STRING*; otherwise, they include *COMPOUND_TEXT*.

_MOTIF_DEFERRED_CLIPBOARD_TARGETS
The widget transfers, as type *ATOM*, a list of the targets it supports for delayed transfer for the *CLIPBOARD* selection. This widget currently supplies no targets for *_MOTIF_DEFERRED_CLIPBOARD_TARGETS*.

_MOTIF_EXPORT_TARGETS
The widget transfers, as type *ATOM*, a list of the targets to be used as the value of the DragContext's **XmNexportTargets** in a drag-and-drop transfer. These include *COMPOUND_TEXT*, the encoding of the locale, *STRING*, *TEXT*, *BACKGROUND*, and *FOREGROUND*.

_MOTIF_LOSE_SELECTION
The widget takes the following actions:

- When losing the *PRIMARY* selection, it unhighlights the selected text and calls the **XmNlosePrimaryCallback** procedures.
- When losing the *SECONDARY* selection, it removes the secondary selection highlight.
- When losing the _MOTIF_DESTINATION selection, if the widget does not have focus, it changes the cursor to indicate that the widget is no longer the destination.

As a source of data, Text also supports the following standard Motif targets:

BACKGROUND
The widget transfers **XmNbackground** as type *PIXEL*.

CLASS The widget finds the first shell in the widget hierarchy that has a WM_CLASS property and transfers the contents as text in the current locale.

CLIENT_WINDOW
The widget finds the first shell in the widget hierarchy and transfers its window as type *WINDOW*.

COLORMAP
The widget transfers **XmNcolormap** as type *COLORMAP*.

FOREGROUND
The widget transfers **XmNforeground** as type *PIXEL*.

NAME The widget finds the first shell in the widget hierarchy that has a WM_NAME property and transfers the contents as text in the current locale.

TARGETS The widget transfers, as type *ATOM*, a list of the targets it supports. These include the standard targets in this list. These also include *COMPOUND_TEXT*, the encoding of the locale, *STRING*, and *TEXT*.

TIMESTAMP
The widget transfers the timestamp used to acquire the selection as type *INTEGER*.

_MOTIF_RENDER_TABLE
The widget transfers **XmNrenderTable** if it exists, or else the default text render table, as type *STRING*.

_MOTIF_ENCODING_REGISTRY
The widget transfers its encoding registry as type *STRING*. The value is a list of NULL separated items in the form of tag encoding pairs. This target symbolizes the transfer target for the Motif Segment Encoding Registry. Widgets and applications can use this Registry to register text encoding formats for specified render table tags. Applications access this Registry by calling **XmRegisterSegmentEncoding** and **XmMapSegmentEncoding**.

As a destination for data, Text chooses a target and requests conversion of the selection to that target. If the encoding of the locale is present in the list of available targets, Text chooses a requested target from the available targets in the following order of preference:

1. The encoding of the locale
2. *TEXT*
3. *COMPOUND_TEXT*
4. *STRING*

If the encoding of the locale is not present in the list of available targets, Text chooses a requested target from the available targets in the following order of preference:

1. *COMPOUND_TEXT*
2. *STRING*

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**	**Type**	**Default**	**Access**
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C

XmNautoShow-CursorPosition	XmCAutoShow-CursorPosition	Boolean	True	CSG
XmNcursorPosition	XmCCursorPosition	XmTextPosition	0	CSG
XmNcursor-PositionVisible	XmCCursor-PositionVisible	Boolean	dynamic	CSG
XmNdestinationCallback	XmCCallback	XtCallbackList	NULL	C
XmNeditable	XmCEditable	Boolean	True	CSG
XmNeditMode	XmCEditMode	int	XmSINGLE_LINE_-EDIT	CSG
XmNfocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNgainPrimary-Callback	XmCCallback	XtCallbackList	NULL	C
XmNlosePrimaryCallback	XmCCallback	XtCallbackList	NULL	C
XmNlosingFocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	5	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	5	CSG
XmNmaxLength	XmCMaxLength	int	largest integer	CSG
XmNmodifyVerify-Callback	XmCCallback	XtCallbackList	NULL	C
XmNmodifyVerify-CallbackWcs	XmCCallback	XtCallbackList	NULL	C
XmNmotionVerify-Callback	XmCCallback	XtCallbackList	NULL	C
XmtotalLines	XmCTotalLines	int	dynamic	C
XmNsource	XmCSource	XmTextSource	Default source	CSG
XmNtopCharacter	XmCTopCharacter	XmTextPosition	0	CSG
XmNvalue	XmCValue	String	""	CSG
XmNvalueChanged-Callback	XmCCallback	XtCallbackList	NULL	C
XmNvalueWcs	XmCvalueWcs	wchar_t *	(wchar_t *)""	CSG[1]
XmNverifyBell	XmCVerifyBell	Boolean	dynamic	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() action. 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. Setting this resource to False, however, does not ensure that Text will not scroll.

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).

XmNcursorPositionVisible
If the widget has an *XmPrintShell* as one of its ancestors, then the default value is **False**; otherwise, it is **True**.

XmNdestinationCallback
Specifies a list of callbacks called when the widget is the destination of a transfer operation. The type of the structure whose address is passed to these callbacks is **XmDestinationCallbackStruct**. The reason is **XmCR_OK**.

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.

When **XmNeditable** is used on a widget it sets the dropsite to **XmDROP_SITE_ACTIVE**.

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 by 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 **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.

XmNtotalLines
Indicates the number of lines in the text widget buffer (not necessarily visible). The initial value 1 means the text buffer is empty. The number of lines reported takes into account the **wordWrap** policy (that is, it's not simply the number of newline characters.

XmNvalue Specifies the string value of the Text widget as a **char*** data value. Moves the cursor to position 0 unless a value of **XmNcursorPosition** was explicitly supplied in the argument list. 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. Moves the cursor to position 0 unless a value of **XmNcursorPosition** was explicitly supplied in the argument list.

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**	**Type**	**Default**	**Access**
XmNpendingDelete	XmCPendingDelete	Boolean	True	CSG
XmNselectionArray	XmCSelectionArray	XtPointer	default array	CSG
XmNselectionArrayCount	XmCSelectionArrayCount	int	4	CSG
XmNselectThreshold	XmCSelectThreshold	int	5	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 some time of the previous mouse click increments the index into this array and performs the defined action for that index. (This "multiclick" time is specified by the operating environment, and varies among different systems. In general, it is usually set to some fraction of a second.) The possible actions in the order they occur in the default array are as follows:

1. **XmSELECT_POSITION**, which resets the insert cursor position.
2. **XmSELECT_WORD**, which selects a word.
3. **XmSELECT_LINE**, which selects a line of text. This action sees a line as delimited by *hard* newline characters. In other words, if the word wrap feature is on (**XmNwordWrap** is True), this will ignore the newlines automatically inserted by the widget. This is the default.
4. **XmSELECT_OUT_LINE**, which selects a line of text. This action sees a line as delimited by *hard* or *soft* newline characters. In other words, if the word wrap feature is on (**XmNwordWrap** is True),

the newlines automatically inserted by the widget will be treated as delimiting lines.

5. **XmSELECT_ALL**, which 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. This resource also specifies whether a drag should be started and the number of pixels to start a drag when **Btn2Down** and **Btn1Down** are integrated.

XmText Output Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNblinkRate	XmCBlinkRate	int	500	CSG
XmNcolumns	XmCColumns	short	dynamic	CSG
XmNcursorPosition-Visible	XmCCursorPosition-Visible	Boolean	True	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNresizeHeight	XmCResizeHeight	Boolean	False	CSG
XmNresizeWidth	XmCResizeWidth	Boolean	False	CSG
XmNrows	XmCRows	short	dynamic	CSG
XmNwordWrap	XmCWordWrap	Boolean	False	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 width as derived from the specified font. If the em-space value is available, it is used. If not, the width of the numeral "0" is used. If this is not available, the maximum width is used. 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.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, and if the *XmText* widget resource *XmNeditMode* is *XmSINGLE_LINE_EDIT*, this attribute is ignored. If no width is specified, the default is 1.

XmNcursorPositionVisible

Indicates that the insert cursor position is marked by a text cursor when the Boolean value is True.

XmNfontList

Specifies the font list to be used for Text. The font list is an obsolete structure and is retained only for compatibility with earlier releases of Motif. Use the render table (**XmNrenderTable**) instead of font lists wherever possible. If both are specified, the render table will take precedence. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the font list is initialized to the **XmTEXT_RENDER_TABLE** value 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 **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**(3) for more information on a font list structure.

XmNrenderTable

Specifies the render table to be used in deriving a font set or font for rendering text. If both a render table and a font list are specified, the render table will take precedence. If the value of **XmNrenderTable** is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the font list is initialized to the **XmTEXT_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

Text searches the render table for the first occurrence of a rendition that has the tag **_MOTIF_DEFAULT_LOCALE**. If a default element is not found, the first rendition in the table is used. Refer to **XmRenderTable**(3) for more information on the render table 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 Text widget whose parent is a ScrolledWindow and when **XmNscrollVertical** is True.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, this resource indicates that the text attempts to resize its height to accommodate all the text contained in the widget when the Boolean value is **True**. This attribute is ignored if *XmNwordWrap* 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.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, this attribute is still effective even if *XmNwordWrap* is **True**. Also, this attribute is ignored when the application uses a text widget whose parent is a **ScrolledWindow** and *XmNscrollHorizaontal* 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.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, this attribute is still effective, even if the *XmText* widget resource

XmNeditMode is *XmSINGLE_LINE_EDIT*. If no height is specified, the default is 20.

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 whitespace. Whitespace is defined as a space, tab, or newline. This attribute is ignored if the text widget resource **XmNeditMode** is **XmSINGLE_LINE_EDIT**. Note that this resource is only valid when the widget is not a scroll one, or, if the widget is a scroll widget, that the **XmNscrollHorizontal** resource is False.

Indicates that lines are to be broken at word breaks (that is, when the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, the text does not go off the bottom edge of the window) when the Boolean value is **True**.

The following resources are used only when text is created in a ScrolledWindow. See the reference page for **XmCreateScrolledText**.

XmText Scrolling Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNscrollHorizontal	XmCScroll	Boolean	True	CG
XmNscrollLeftSide	XmCScrollSide	Boolean	False	CG
XmNscrollTopSide	XmCScrollSide	Boolean	False	CG
XmNscrollVertical	XmCScroll	Boolean	True	CG

Note in connection with this table that if the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, the default is **True**.

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**.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, this attribute is ignored if the *XmText* widget resource *XmNeditMode* is *XmSINGLE_LINE_EDIT*.

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**.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, this resource is still effective, even if the XmText widget resource *XmNeditMode* is *XmSINGLE_LINE_EDIT*.

XmNscrollTopSide

Indicates that the horizontal ScrollBar should be placed on the top side of the scrolled text window when the Boolean value is True.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, this attribute is ignored if *XmNscrollHorizontal* is **False** or the **Xmtext** resource *XmNeditMode* is *XmSINGLE_LINE_EDIT*.

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**.

When the *XmNlayoutDirection* resource is *XmTOP_TO_BOTTOM*, this resource is still effective, even if the *XmText* widget resource *XmNeditMode* is *XmSINGLE_LINE_EDIT*.

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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C

XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight- OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlight- Pixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight- Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayout- Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_- GROUP	CSG
XmNpopupHandler- Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadow- - Color	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestor- Sensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG

XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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. Note that not all actions may be negated. For example, **XmCR_LOSING_FOCUS** callbacks may be beyond the control of the widget if they are produced by mouse clicks.
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. Note that not all actions may be negated. For example, **XmCR_LOSING_FOCUS** callbacks may be beyond the control of the widget if they are produced by mouse clicks.
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. Note that the *event* field will never be valid for **XmCR_MOVING_INSERT_CURSOR**.

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, doit, currInsert, newInsert*

A pointer to the following callback structure is passed to the **XmNdestinationCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent  *event;
        Atom selection;
        XtEnum operation;
        int flags;
        XtPointer transfer_id;
        XtPointer destination_data;
        XtPointer location_data;
        Time time;
} XmDestinationCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which data transfer is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

operation Indicates the type of transfer operation requested.

- When the selection is *PRIMARY* or *SECONDARY*, possible values are **XmMOVE**, **XmCOPY**, and **XmLINK**.
- When the selection is *CLIPBOARD*, possible values are **XmCOPY** and **XmLINK**.
- When the selection is _MOTIF_DROP, possible values are **XmMOVE**, **XmCOPY**, **XmLINK**, and **XmOTHER**. A value of **XmOTHER** means that the callback procedure must get further information from the **XmDropProcCallbackStruct** structure in the *destination_data* member.

flags Indicates whether or not the destination widget is also the source of the data to be transferred. Following are the possible values:

XmCONVERTING_NONE
The destination widget is not the source of the data to be transferred.

XmCONVERTING_SAME
The destination widget is the source of the data to be transferred.

transfer_id Serves as a unique ID to identify the transfer transaction.

destination_data
Contains information about the destination. When the selection is _MOTIF_DROP, the callback procedures are called by the drop site's **XmNdropProc**, and *destination_data* is a pointer to the **XmDropProcCallbackStruct** structure passed to the **XmNdropProc** procedure. When the selection is *SECONDARY*, *destination_data* is an Atom representing a target recommmended by the selection owner for use in converting the selection. Otherwise, *destination_data* is NULL.

location_data
Contains information about the location where data is to be transferred. The value is always NULL when the selection is *CLIPBOARD*. If the

value is NULL, the data is to be inserted at the widget's cursor position. Otherwise, the value is a pointer to an *XPoint* structure containing the x- and y- coordinates at the location where the data is to be transferred. Once *XmTransferDone* procedures start to be called, **location_data** will no longer be stable.

time Indicates the time when the transfer operation began.

Translations

The **XmText** translations are described in the following list. The actions represent the effective behavior of the associated events, and they may differ in a right-to-left language environment.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

≈c s ≈m ≈a <Btn1Down>:
extend-start()

c ≈s ≈m ≈a <Btn1Down>:
move-destination()

≈c ≈s ≈m ≈a <Btn1Down>:
grab-focus()

≈c ≈m ≈a <Btn1Motion>:
extend-adjust()

≈c ≈m ≈a <Btn1Up>:
extend-end()

<Btn2Down>:
process-bdrag()

m ≈a <Btn2Motion>:
secondary-adjust()

≈m a <Btn2Motion>:
secondary-adjust()

s c <Btn2Up>:
link-to()

≈s <Btn2Up>:
: **copy-to()**

≈c <Btn2Up>:
: **move-to()**

:m <Key>osfPrimaryPaste:
: **cut-primary()**

:a <Key>osfPrimaryPaste:
: **cut-primary()**

:<Key>osfPrimaryPaste:
: **copy-primary()**

:m <Key>osfCut:
: **cut-primary()**

:a <Key>osfCut:
: **cut-primary()**

:<Key>osfCut:
: **cut-clipboard()**

:<Key>osfPaste:
: **paste-clipboard()**

:m <Key>osfCopy:
: **copy-primary()**

:a <Key>osfCopy:
: **copy-primary()**

:<Key>osfCopy:
: **copy-clipboard()**

:s c <Key>osfBeginLine:
: **beginning-of-file(***extend***)**

:c <Key>osfBeginLine:
: **beginning-of-file()**

:s <Key>osfBeginLine:
: **beginning-of-line(***extend***)**

:<Key>osfBeginLine:
: **beginning-of-line()**

:s c <Key>osfEndLine:
: **end-of-file(***extend***)**

:c <Key>osfEndLine:
: **end-of-file()**

:s <Key>osfEndLine:
: **end-of-line(***extend***)**

:<Key>osfEndLine:
: **end-of-line()**

:s <Key>osfPageLeft:
: **page-left(***extend***)** (ignored in vertical writing)

:<Key>osfPageLeft:
: **page-left()** (**next-page()** in vertical writing)

:s c <Key>osfPageUp:
: **page-left(***extend***)**

:c <Key>osfPageUp:
: **page-left()**

:s <Key>osfPageUp:
: **previous-page(***extend***)** (ignored in vertical writing)

:<Key>osfPageUp:
: **previous-page()** (**page-up()** in vertical writing)

:s <Key>osfPageRight:
: **page-right(***extend***)** (ignored in vertical writing)

:<Key>osfPageRight:
: **page-right()** (**previous-page()** in vertical writing)

s c <Key>osfPageDown:
: **page-right(***extend***)** (ignored in vertical writing)

:c <Key>osfPageDown:
: **page-right()**

:s <Key>osfPageDown:
: **next-page(***extend***)** (ignored in vertical writing)

:<Key>osfPageDown:
: **next-page()** (**page-down()** in vertical writing)

:<Key>osfClear:
  clear-selection()

:<Key>osfBackSpace:
  delete-previous-character()

:s m <Key>osfDelete:
  cut-primary()

:s a <Key>osfDelete:
  cut-primary()

:s <Key>osfDelete:
  cut-clipboard()

:c <Key>osfDelete:
  delete-to-end-of-line()

:<Key>osfDelete:
  delete-next-character()

:c m <Key>osfInsert:
  copy-primary()

:c a <Key>osfInsert:
  copy-primary()

:s <Key>osfInsert:
  paste-clipboard()

:c <Key>osfInsert:
  copy-clipboard()

:s <Key>osfSelect:
  key-select()

:<Key>osfSelect:
  set-anchor()

:<Key>osfSelectAll:
  select-all()

:<Key>osfDeselectAll:
  deselect-all()

:<Key>osfActivate:
  activate()

:<Key>osfAddMode:
toggle-add-mode()

:<Key>osfHelp:
Help()

:<Key>osfCancel:
process-cancel()

:s c <Key>osfLeft:
backward-word(*extend***)** (**forward-paragraph(***extend***)** in vertical writing)

:c <Key>osfLeft:
backward-word() (**forward-paragraph()** in vertical writing)

:s <Key>osfLeft:
key-select(*left***) (process-shift-left()** in vertical writing)

:<Key>osfLeft:
backward-character() (**process-left()** in vertical writing)

:s c <Key>osfRight:
forward-word(*extend***)** (**backward-paragraph(***extend***)** in vertical writing)

:c <Key>osfRight:
forward-word() (**backward-paragraph()** in vertical writing)

:s <Key>osfRight:
key-select(*right***)** (**process-shift-right** in vertical writing)

:<Key>osfRight:
forward-character() (**process-right()** in vertical writing)

:s c <Key>osfUp:
backward-paragraph(*extend***)** (**backward-word(***extend***)** in vertical writing)

:c <Key>osfUp:
backward-paragraph() (**backward-word()** in vertical writing)

:s <Key>osfUp:
process-shift-up() (**key-select(***up***)** in vertical writing)

:<Key>osfUp:
process-up() (**backward-character()** in vertical writing)

:s c <Key>osfDown:
: **forward-paragraph(***extend***)** (**forward-word(***extend***)** in vertical writing)

:c <Key>osfDown:
: **forward-paragraph()** (**forward-word()** in vertical writing)

:s <Key>osfDown:
: **process-shift-down()** (**key-select(***down***)** in vertical writing)

:<Key>osfDown:
: **process-down()** (**forward-character()** in vertical writing)

c ≈m ≈a <Key>slash:
: **select-all()**

c ≈m ≈a <Key>backslash:
: **deselect-all()**

s c ≈m ≈a <Key>Tab:
: **prev-tab-group()**

≈s c ≈m ≈a <Key>Tab:
: **next-tab-group()**

s ≈c ≈m ≈a <Key>Tab:
: **process-tab(***Prev***)**

≈s ≈c ≈m ≈a <Key>Tab:
: **process-tab(***Next***)**

≈s c ≈m ≈a <Key>Return:
: **activate()**

≈s ≈c ≈m ≈a <Key>Return:
: **process-return()**

≈s c ≈m ≈a <Key>space:
: **set-anchor()**

s c ≈m ≈a <Key>space:
: **key-select()**

s ≈c ≈m ≈a <Key>space:
: **self-insert()**

<Key>: **self-insert()**

The Text button event translations are modified when Display's **XmNenableBtn1Transfer** resource does not have a value of **XmOFF** (in other words, it is either **XmBUTTON2_TRANSFER** or **XmBUTTON2_ADJUST**). This option allows the actions for selection and transfer to be integrated on Btn1, and the actions for extending the selection can be bound to Btn2. The actions for Btn1 that are defined above still apply when the Btn1 event occurs over text that is not selected. The following actions apply when the Btn1 event occurs over text that is selected:

<Btn1Down>:
process-bdrag().

<Shift><Btn1Down>:
process-bdrag().

<Ctrl><Btn1Down>:
process-bdrag().

<Btn1Down><Shift><Btn1Up>:
grab-focus(), **extend-end**.

<Shift><Btn1Down><Shift><Btn1Up>:
extend-start(), **extend-end()**.

<Ctrl><Btn1Down> <Shift><Btn1Up>:
move-destination().

When Display's **XmNenableBtn1Transfer** resource has a value of **XmBUTTON2_ADJUST**, the following actions apply:

<Btn2Down>:
extend-start().

<Btn2Motion>:
extend-adjust().

<Btn2Up>: **extend-end()**.

Action Routines

The **XmText** action routines are

activate(): Calls the callbacks for **XmNactivateCallback**. Passes the event to the parent.

backward-character(*extend*):

Moves the insertion cursor one character to the left. This action may have different behavior in a right-to-left language environment.

If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The backward-character() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the backward-character() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

In vertical writing, if *XmNeditMode* is *XmSINGLE_LINE_EDIT* and *XmNnavigationType* is *XmNONE*, traverses to the widget to the left in the tab group. If *XmNeditMode* is *XmMULTI_LINE_EDIT*, moves the insertion cursor to the next line in the same column.

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.

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.

The backward-paragraph() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the backward-paragraph() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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. 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.

The backward-word() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the backward-word() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

beep(): Causes the terminal to beep. The beep() action produces no callbacks.

beginning-of-file(*extend*):
If this action is called with no argument, moves the insertion cursor to the beginning of the text.

If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The beginning-of-file() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the beginning-of-file() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

beginning-of-line(*extend*):
If this action is called with no argument, moves the insertion cursor to the beginning of the line.

If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The beginning-of-line() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the beginning-of-line() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

clear-selection():
Clears the current selection by replacing each character except Return with a space character.

The clear-selection() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE** and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

copy-clipboard():
If this widget owns the primary selection, this action copies the selection to the clipboard. This action calls the **XmNconvertCallback** procedures, possibly multiple times, for the *CLIPBOARD* selection.

copy-primary():
Copies the primary selection to just before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmCOPY** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection.

In addition, the copy-primary() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, to the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

copy-to(): If a secondary selection exists, this action copies the secondary selection to the insertion position of the destination component. If the primary selection is in the destination widget, it will be deselected. Otherwise, there is no effect on the primary selection.

This action calls the destination's **XmNdestinationCallback** procedures for the *SECONDARY* selection and the **XmCOPY** operation. The destination's **XmNdestinationCallback** procedures or the destination component itself invokes the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *SECONDARY* selection.

If no secondary selection exists, this action copies the primary selection to the pointer position. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmCOPY** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection.

In addition, the copy-to() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, to the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If there is no secondary selection, the copy-to() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

cut-clipboard():

If this widget owns the primary selection, this action cuts the selection to the clipboard. This action calls the **XmNconvertCallback** procedures, possibly multiple times, for the *CLIPBOARD* selection. If the transfer is successful, this action then calls the **XmNconvertCallback** procedures for the *CLIPBOARD* selection and the *DELETE* target.

In addition, the cut-clipboard() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

cut-primary():

Cuts the primary selection and pastes it just before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmMOVE** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection. If the transfer is successful, this action then calls the selection owner's **XmNconvertCallback** procedures for the *PRIMARY* selection and the *DELETE* target. The cut-primary() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

In addition, the cut-primary() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**, the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the

XmNvalueChangedCallback procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The delete-next-character() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The delete-next-word() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The delete-previous-character() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The delete-previous-word() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

delete-selection():

Deletes the current selection.

The delete-selection() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The **delete-to-end-of-line()** action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

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.

  The delete-to-start-of-line() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

deselect-all():
: Deselects the current selection. The deselect-all() action produces no callbacks.

end-of-file(*extend*):
: If this action is called with no argument, moves the insertion cursor to the end of the text.

  If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

  The end-of-file() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the end-of-file() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

end-of-line(*extend*):
: If this action is called with no argument, moves the insertion cursor to the end of the line. If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The end-of-line() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the end-of-line() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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.

The extend-adjust() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. The extend-adjust() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

extend-end():

Moves the insertion cursor to the position of the pointer. The extend-end() action is used to commit the selection. After this action has been done, process-cancel() will no longer cancel the selection.

The extend-end() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. The extend-end() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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. The extend-start() action may produce no callbacks, however, the extend-start() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

forward-character(*extend*):

Moves the insertion cursor one character to the right. This action may have different behavior in a right-to-left language environment.

If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The forward-character() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the forward-character() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

In vertical writing, if *XmNeditMode* is *XmSINGLE_LINE_EDIT* and *XmNnavigationType* is *XmNONE*, traverses to the widget to the right in the tab group. If *XmNeditMode* is *XmMULTI_LINE_EDIT*, moves the insertion cursor to the previous line in the same column.

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.

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.

The forward-paragraph() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the forward-paragraph() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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. 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.

The forward-word() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the forward-word() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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.

The grab-focus() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The insert-string() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. Note that, in the case of an empty string, no callbacks will be called, since no modification will have been done. However, if the insertion position is inside the current selection, **insert-string** with an empty string will cause the selection to be deselected, and the **XmNmotionVerifyCallback** procedures to be called with reason value **XmCR_MOVING_INSERT_CURSOR**, **XmCR_MODIFYING_TEXT_VALUE**, and **XmCR_VALUE_CHANGED**.

key-select(*right*|*left*):

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.

Note that after a **key-select** action, the selection will still begin at the original anchor, and will extend to the position indicated in the action call. If this new position is on the opposite side of the selection anchor from the previous selection boundary, the original selection will be deselected.

The key-select() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. The key-select() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

In vertical writing, if called with the argument **left**, and if *XmNeditMode* is *XmMULTI_LINE_EDIT*, moves the insertion cursor to the next line in the same column. In vertical writing, if called with the argument **right**, and if *XmNeditMode* is *XmMULTI_LINE_EDIT*, moves the insertion cursor to the previous line in the same column.)

kill-next-character():

In normal mode, if there is a nonnull selection, kills 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.

The killed text is stored in *CUT_BUFFER0*.

The kill-next-character() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The killed text is stored in *CUT_BUFFER0*.

The kill-next-word() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The killed text is stored in *CUT_BUFFER0*.

The kill-previous-character() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The killed text is stored in *CUT_BUFFER0*.

The kill-previous-word() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

kill-selection():

Kills the currently selected text and stores the text in the cut buffer.

The killed text is stored in *CUT_BUFFER0*.

The kill-selection() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The killed text is stored in *CUT_BUFFER0*.

The kill-to-end-of-line() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value

XmCR_VALUE_CHANGED. In the case where there is a nonnull selection to be deleted, this action may also produce calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The killed text is stored in *CUT_BUFFER0*.

The kill-to-start-of-line() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

link-primary():
: Places a link to the primary selection just before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmLINK** operation. The Text widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the link to the primary selection and for taking any related actions.

link-to():
: If a secondary selection exists, this action places a link to the secondary selection at the insertion position of the destination component. This action calls the destination's **XmNdestinationCallback** procedures for the *SECONDARY* selection and the **XmLINK** operation.

If no secondary selection exists, this action places a link to the primary selection at the pointer position. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmLINK** operation.

The Text widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the link to the primary or secondary selection and for taking any related actions.

move-destination():
Moves the insertion cursor to the pointer position without changing any existing current selection. If there is a current selection, sets the widget as the destination widget. This also moves the widget focus to match the insertion cursor.

The move-destination() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

move-to(): If a secondary selection exists, this action moves the secondary selection to the insertion position of the destination component. If the secondary selection is in the destination widget, and the secondary selection and the primary selection overlap, the result is undefined. This action calls the destination's **XmNdestinationCallback** procedures for the *SECONDARY* selection and the **XmMOVE** operation. The destination's **XmNdestinationCallback** procedures or the destination component itself invokes the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *SECONDARY* selection. If the transfer is successful, this action then calls the selection owner's **XmNconvertCallback** procedures for the *SECONDARY* selection and the *DELETE* target.

If no secondary selection exists, this action moves the primary selection to the pointer position. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmMOVE** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection. If the transfer is successful, this action then calls the selection owner's **XmNconvertCallback** procedures for the *PRIMARY* selection and the *DELETE* target.

The move-to() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If there is no secondary selection, the move-to() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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.

The newline() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_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.

The newline-and-backup() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The newline-and-indent() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

next-line(): Moves the insertion cursor to the next line.

The next-line() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

next-page(*extend*):

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with no argument, moves the insertion cursor forward one page.

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with an argument of *extend*, it moves the insertion cursor as in the case of no argument and extends the current selection.

The next-page() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the next-page() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

In vertical writing, scrolls the viewing window down one page of text.

next-tab-group():

Traverses to the next tab group.

The next-tab-group() action produces no callbacks, unless it results in the widget losing focus, in which case, the **XmNlosingFocusCallback** procedures are called with reason value **XmCR_LOSING_FOCUS**.

page-left(*extend*):

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with no argument, moves the insertion cursor back one page.

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with an argument of *extend*, it moves the insertion cursor as in the case of no argument and extends the current selection.

The page-left() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the page-left() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

page-right(*extend*):

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with no argument, moves the insertion cursor forward one page.

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with an argument of *extend*, it moves the insertion cursor as in the case of no argument and extends the current selection.

The page-right() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the page-right() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

paste-clipboard():

Pastes the contents of the clipboard before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *CLIPBOARD* selection and the **XmCOPY** operation.

The paste-clipboard() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

prev-tab-group():

Traverses to the previous tab group.

The prev-tab-group() action produces no callbacks, unless it results in the widget losing focus, in which case, the **XmNlosingFocusCallback** procedures are called with reason value **XmCR_LOSING_FOCUS**.

previous-line():

Moves the insertion cursor to the previous line.

The previous-line() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

previous-page(*extend*):

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with no argument, moves the insertion cursor back one page.

If **XmNeditMode** is **XmMULTI_LINE_EDIT** and this action is called with an argument of *extend*, it moves the insertion cursor as in the case of no argument and extends the current selection.

The previous-page() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the previous-page() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

In vertical writing, if called without an argument, scrolls the viewing window up one page of text.

process-bdrag()

If the pointer is within the selection, this action starts a drag operation for the selection. This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures, possibly multiple times, for the _MOTIF_DROP selection.

If no selection exists or the pointer is outside the selection, this action prepares to start a secondary selection at the pointer position.

Note: Note that when dragging a secondary selection to a different widget, focus will shift momentarily to the second widget, and then back to the original widget. This will generate callbacks to the **XmNlosingFocusCallback** procedures as focus is lost (by each widget) as well as callbacks to the **XmNfocusCallback** procedures as focus is regained.

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(*extend*):

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** 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.

In this case, the action will produce the **XmNlosingFocusCallback** callbacks with reason value **XmCR_LOSING_FOCUS**.

If **XmNeditMode** is **XmMULTI_LINE_EDIT**, moves the insertion cursor down one line.

The process-down() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

In vertical writing, moves the insertion cursor one character down.

process-home():
Moves the insertion cursor to the beginning of the line.

The process-home() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The process-return() action during single-line edit produces calls to the **XmNactivateCallback** procedures with reason value **XmCR_ACTIVATE**. During multi-line editing, the process-return() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

process-shift-down():
If **XmNeditMode** is **XmMULTI_LINE_EDIT**, moves the insertion cursor down one line and selects. If **XmNeditMode** is **XmSINGLE_LINE_EDIT**, this action behaves like process-up() in **XmSINGLE_LINE_EDIT**. Refer to the process-up() action.

The process-shift-down() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

In vertical writing, if called with the argument **up**, moves the insertion cursor one character up and extends the current selection. If called with the argument **down**, moves the insertion cursor one character down and extends the current selection.

process-shift-up():

If **XmNeditMode** is **XmMULTI_LINE_EDIT**, moves the insertion cursor up one line and selects. If **XmNeditMode** is **XmSINGLE_LINE_EDIT**, this action behaves like process-up() in **XmSINGLE_LINE_EDIT**. Refer to the process-up() action.

The process-shift-up() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

In vertical writing, if called with the argument **up**, moves the insertion cursor one character up and extends the current selection. If called with the argument **down**, moves the insertion cursor one character down and extends the current selection.

process-tab(*Prev*|*Next*):

If **XmNeditMode** is **XmSINGLE_LINE_EDIT**, traverses to the next tab group if the direction argument is **Next**, or to the previous tab group if the direction is **Prev**. If **XmNeditMode** is **XmMULTI_LINE_EDIT**, and the direction is **Next**, the action inserts a tab. The **Prev** direction has no effect with **XmMULTI_LINE_EDIT**. In the Text widget, there is a preset tab stop at every eighth columns.

The process-tab() action under multi-line editing produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. Under single-line editing, the action produces no callbacks, unless it results in the widget losing focus, in which case, the **XmNlosingFocusCallback** procedures are called with reason value **XmCR_LOSING_FOCUS**.

process-up(*extend*):

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.

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.

The process-up() action under multi-line editing produces calls to the **XmNmotionVerifyCallback** procedures with reason value

XmCR_MOVING_INSERT_CURSOR. Under single-line editing, the action produces no callbacks unless it results in the widget losing focus, in which case, the **XmNlosingFocusCallback** procedures are called with reason value **XmCR_LOSING_FOCUS**.

In vertical writing, moves the insertion cursor one character up.

redraw-display():
Redraws the contents of the text window.

The redraw-display() action produces no callbacks.

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.

The **scroll-cursor-vertically** action produces no callbacks.

scroll-one-line-down():
Scrolls the text area down one line.

The scroll-one-line-down() action produces no callbacks.

scroll-one-line-up():
Scrolls the text area up one line.

The scroll-one-line-up() action produces no callbacks.

secondary-adjust():
Extends the secondary selection to the pointer position.

The secondary-adjust() action produces no callbacks.

secondary-notify():
Copies the secondary selection to the insertion cursor of the destination widget.

The secondary-notify() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the

XmNvalueChangedCallback procedures with reason value **XmCR_VALUE_CHANGED**.

secondary-start():
Marks the beginning of a secondary selection.

The secondary-start() action produces no callbacks.

select-adjust():
Moves the current selection. The amount of text selected depends on the number of mouse clicks, as specified by the **XmNselectionArray** resource.

The select-adjust() action may produce no callbacks, however, it may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

select-all(): Selects all text.

The select-all() action may produce calls to the **XmNgainPrimaryCallback** procedures.

select-end(): Moves the current selection. The amount of text selected depends on the number of mouse clicks, as specified by the **XmNselectionArray** resource.

The select-end() action produces no callbacks.

select-start():
Marks the beginning of a new selection region.

The select-start() action may produce calls to the **XmNgainPrimaryCallback** procedures.

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.

The self-insert() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

set-anchor(): Resets the anchor point for extended selections. Resets the destination of secondary selection actions.

The set-anchor() action produces no callbacks.

set-insertion-point():
Sets the insertion position to the position of the mouse pointer.

The set-insertion-point() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. Note that if the mouse pointer is already over the position of the insertion cursor, the cursor will not be moved, and no callbacks will be produced.

set-selection-hint():
Sets the text source and location of the current selection.

The set-selection-hint() action produces no callbacks.

toggle-add-mode():
Toggles the state of Add Mode.

The toggle-add-mode() action produces no callbacks.

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.

The following traversal actions generate no callbacks unless they result in the loss of focus by the widget in question, as when **XmNnavigationType** is **XmNONE**. In this case, they produce calls to the **XmNlosingFocusCallback** procedures, with reason value **XmCR_FOCUS_MOVED**.

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 (or whatever is currently in the *CUT_BUFFER0*). The inserted text appears before the insertion cursor.

The unkill() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_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 and stops the cursor from blinking.

Virtual Bindings

The bindings for virtual keys are vendor specific. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Related Information

Core(3), **XmCreateScrolledText**(3), **XmCreateText**(3), **XmFontList**(3), **XmFontListAppendEntry**(3), **XmPrimitive**(3), **XmTextClearSelection**(3), **XmTextCopy**(3), **XmTextCopyLink**(3), **XmTextCut**(3), **XmTextEnableRedisplay**(3), **XmTextDisableRedisplay**(3), **XmTextField**(3), **XmTextFindString**(3), **XmTextFindStringWcs**(3), **XmTextGetBaseline**(3), **XmTextGetEditable**(3), **XmTextGetInsertionPosition**(3), **XmTextGetLastPosition**(3), **XmTextGetMaxLength**(3), **XmTextGetSelection**(3), **XmTextGetSelectionWcs**(3), **XmTextGetSelectionPosition**(3), **XmTextGetSource**(3), **XmTextGetString**(3), **XmTextGetStringWcs**(3), **XmTextGetSubstring**(3), **XmTextGetSubstringWcs**(3), **XmTextGetTopCharacter**(3), **XmTextInsert**(3), **XmTextInsertWcs**(3), **XmTextPaste**(3), **XmTextPasteLink**(3), **XmTextPosToXY**(3), **XmTextPosition**(3), **XmTextRemove**(3), **XmTextReplace**(3), **XmTextReplaceWcs**(3), **XmTextScroll**(3), **XmTextSetAddMode**(3), **XmTextSetEditable**(3), **XmTextSetHighlight**(3), **XmTextSetInsertionPosition**(3), **XmTextSetMaxLength**(3), **XmTextSetSelection**(3), **XmTextSetSource**(3), **XmTextSetString**(3),

XmTextSetStringWcs(3), **XmTextSetTopCharacter**(3), **XmTextShowPosition**(3), and **XmTextXYToPos**(3).

XmTextField

Purpose

The TextField class

Synopsis

#include <Xm/Xm.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.

TextField uses the *XmQTnavigator*, *XmQTspecifyRenderTable*, and *XmQTscrollFrame* traits, and holds the *XmQTaccessTextual XmQTtransfer* traits.

If an application or widget calls the **setValue** trait method of *XmQTaccessTextual*, then **XmTextField** will call **XmTextFieldSetString** to set the string value.

Data Transfer Behavior

TextField supports transfer of the primary, secondary, and clipboard selections and dragging of selected text from the widget. TextField can also be the destination for the primary, secondary, and clipboard selections, and it supports dropping of data being dragged onto the widget.

When the **XmNconvertCallback** procedures are called, the **location_data** member of the **XmConvertCallbackStruct** member is NULL if the selected text is being transferred. If the entire text, not the selected text, is being transferred, the value of this member is the widget ID of the TextField widget.

As a source of data, TextField supports the following targets and associated conversions of data to these targets:

locale If the *locale* target matches the widget's locale, the widget transfers the selected text in the encoding of the locale.

COMPOUND_TEXT
The widget transfers the selected text as type *COMPOUND_TEXT*.

STRING The widget transfers the selected text as type *STRING*.

TEXT If the selected text is fully convertible to the encoding of the locale, the widget transfers the selected text in the encoding of the locale. Otherwise, the widget transfers the selected text as type *COMPOUND_TEXT*.

DELETE The widget deletes the selected text.

_MOTIF_CLIPBOARD_TARGETS
The widget transfers, as type *ATOM*, a list of the targets to which the widget can convert data to be placed on the clipboard immediately. These include the following targets:

- *COMPOUND_TEXT*
- The encoding of the locale, if the selected text is fully convertible to the encoding of the locale
- *STRING*, if the selected text is fully convertible to *STRING*

_MOTIF_EXPORT_TARGETS
The widget transfers, as type *ATOM*, a list of the targets to be used as the value of the DragContext's **XmNexportTargets** in a drag-and-drop transfer. These include *COMPOUND_TEXT*, the encoding of the locale, *STRING*, *TEXT*, *BACKGROUND*, and *FOREGROUND*.

_MOTIF_LOSE_SELECTION
The widget takes the following actions:

- When losing the *PRIMARY* selection, it unhighlights the selected text and calls the **XmNlosePrimaryCallback** procedures.
- When losing the *SECONDARY* selection, it removes the secondary selection highlight.
- When losing the _MOTIF_DESTINATION selection, if the widget does not have focus it changes the cursor to indicate that the widget is no longer the destination.

As a source of data, TextField also supports the following standard Motif targets:

BACKGROUND
The widget transfers **XmNbackground** as type *PIXEL*.

CLASS
The widget finds the first shell in the widget hierarchy that has a WM_CLASS property and transfers the contents as text in the current locale.

CLIENT_WINDOW
The widget finds the first shell in the widget hierarchy and transfers its window as type *WINDOW*.

COLORMAP
The widget transfers **XmNcolormap** as type *COLORMAP*.

FOREGROUND
The widget transfers **XmNforeground** as type *PIXEL*.

NAME
The widget finds the first shell in the widget hierarchy that has a WM_NAME property and transfers the contents as text in the current locale.

TARGETS
The widget transfers, as type *ATOM*, a list of the targets it supports. These include the standard targets in this list. These also include *COMPOUND_TEXT*, the encoding of the locale, *STRING*, and *TEXT*.

TIMESTAMP
The widget transfers the timestamp used to acquire the selection as type *INTEGER*.

_MOTIF_RENDER_TABLE
The widget transfers **XmNrenderTable** if it exists, or else the default text render table, as type *STRING*.

_MOTIF_ENCODING_REGISTRY
The widget transfers its encoding registry as type *STRING*. The value is a list of NULL separated items in the form of tag encoding pairs. This target symbolizes the transfer target for the Motif Segment Encoding Registry. Widgets and applications can use this Registry to register text encoding formats for specified render table tags. Applications access this Registry by calling **XmRegisterSegmentEncoding** and **XmMapSegmentEncoding**.

As a destination for data, TextField chooses a target and requests conversion of the selection to that target. If the encoding of the locale is present in the list of available targets, TextField chooses a requested target from the available targets in the following order of preference:

1. *TEXT*
2. *COMPOUND_TEXT*
3. The encoding of the locale
4. *STRING*

If the encoding of the locale is not present in the list of available targets, TextField chooses a requested target from the available targets in the following order of preference:

1. *COMPOUND_TEXT*
2. *STRING*

Classes

TextField widget inherits behavior, resources, and traits 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	Type	Default	Access
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNblinkRate	XmCBlinkRate	int	500	CSG
XmNcolumns	XmCColumns	short	dynamic	CSG
XmNcursorPosition	XmCCursorPosition	XmTextPosition	0	CSG
XmNcursorPosition- Visible	XmCCursorPosition- Visible	Boolean	dynamic	CSG
XmNdestination- Callback	XmCCallback	XtCallbackList	NULL	C
XmNeditable	XmCEditable	Boolean	True	CSG
XmNfocusCallback	XmCCallback	XtCallbackList	NULL	C
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNgainPrimary- Callback	XmCCallback	XtCallbackList	NULL	C
XmNlosePrimary- Callback	XmCCallback	XtCallbackList	NULL	C
XmNlosingFocus- Callback	XmCCallback	XtCallbackList	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	5	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	5	CSG
XmNmaxLength	XmCMaxLength	int	largest integer	CSG
XmNmodifyVerify- Callback	XmCCallback	XtCallbackList	NULL	C
XmNmodifyVerify- CallbackWcs	XmCCallback	XtCallbackList	NULL	C
XmNmotionVerify- Callback	XmCCallback	XtCallbackList	NULL	C
XmNpendingDelete	XmCPendingDelete	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNresizeWidth	XmCResizeWidth	Boolean	False	CSG

XmNselectionArray	XmCSelectionArray	XtPointer	default array	CSG
XmNselectionArray Count	XmCSelectionArray-Count	int	3	CSG
XmNselectThreshold	XmCSelectThreshold	int	5	CSG
XmNvalue	XmCValue	String	""	CSG
XmNvalueChanged Callback	XmCCallback	XtCallbackList	NULL	C
XmNvalueWcs	XmCValueWcs	wchar_t *	(wchar_t *)""	CSG[1]
XmNverifyBell	XmCVerifyBell	Boolean	dynamic	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() action. 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 width as derived from the specified font. If the em-space value is available, it is used. If not, the width of the numeral "0" is used. If this is not available, the maximum width is used. For proportionate fonts, the actual number of characters that fit on a given line may be greater than the value specified.

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

If the text widget has an **XmPrintShell** as one of its ancestors (that is, the widget was created on a print server connection) then the default value is **False**; otherwise, it is **True**.

XmNdestinationCallback

Specifies a list of callbacks called when the widget is the destination of a transfer operation. The type of the structure whose address is passed to these callbacks is **XmDestinationCallbackStruct**. The reason is **XmCR_OK**.

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.

When **XmNeditable** is used on a widget it sets the dropsite to **XmDROP_SITE_ACTIVE**.

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. The font list is an obsolete structure, and is retained only for compatibility with earlier releases of Motif. Use the render table (**XmNrenderTable**) instead of font lists wherever possible. If both are specified, the render table will take precedence. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the font list is initialized to the **XmTEXT_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

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 will be used. Refer to **XmFontList**(3) for more information on a font list structure.

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.

XmNrenderTable

Specifies the render table to be used in deriving a font set or font for displaying text. If both a render table and a font list are specified, the render table will take precedence. If the value of **XmNrenderTable** is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that holds the *XmQTspecifyRenderTable* trait. If such an ancestor is found, the font list is initialized to the **XmTEXT_RENDER_TABLE** value of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

TextField searches the render table for the first occurrence of a rendition that has the tag **_MOTIF_DEFAULT_LOCALE**. If a default element is not found, the first rendition in the table is used. Refer to **XmRenderTable**(3) for more information on the render table structure.

XmNresizeWidth
Indicates that the 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 some time of the previous mouse click will increment the index into this array and perform the defined action for that index. (This "multi-click" time is specified by the operating environment, and varies among different systems. In general, it is usually set to some fraction of a second.) 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. This resource also specifies whether a drag should be started and the number of pixels to start a drag when **Btn2Down** and **Btn1Down** are integrated.

XmNvalue
Specifies the string value of the TextField widget as a **char*** data value. Moves the cursor to position 0 unless a value of **XmNcursorPosition** was explicitly supplied in the argument list. 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**. 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. Moves the cursor to position 0 unless a value of **XmNcursorPosition** was explicitly supplied in the argument list.

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**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottomShadow- Color	Pixel	dynamic	CSG
XmNbottomShadow- Pixmap	XmCBottomShadow- Pixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG

XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight- OnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight- Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation- Type	XmTAB_GROUP	CSG
XmNpopupHandler- Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow- Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_- PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C
XmNheight	XmCHeight	Dimension	dynamic	CSG

XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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. Note that not all actions may be negated. For example, **XmCR_LOSING_FOCUS** callbacks may be beyond the control of the widget if they are produced by mouse clicks.

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. Note that not all actions may be negated. For example, **XmCR_LOSING_FOCUS** callbacks may be beyond the control of the widget if they are produced by mouse clicks.
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. Note that the *event* field will never be valid for **XmCR_MOVING_INSERT_CURSOR**.

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, doit, currInsert, newInsert*

A pointer to the following callback structure is passed to the **XmNdestinationCallback** procedures:

```
typedef struct
{
        int reason;
        XEvent *event;
        Atom selection;
        XtEnum operation;
        int flags;
        XtPointer transfer_id;
        XtPointer destination_data;
        XtPointer location_data;
        Time time;
} XmDestinationCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the *XEvent* that triggered the callback. It can be NULL.

selection Indicates the selection for which data transfer is being requested. Possible values are *CLIPBOARD*, *PRIMARY*, *SECONDARY*, and _MOTIF_DROP.

operation Indicates the type of transfer operation requested.

- When the selection is *PRIMARY* or *SECONDARY*, possible values are **XmMOVE**, **XmCOPY**, and **XmLINK**.
- When the selection is *CLIPBOARD*, possible values are **XmCOPY** and **XmLINK**.
- When the selection is _MOTIF_DROP, possible values are **XmMOVE**, **XmCOPY**, **XmLINK**, and **XmOTHER**. A value of **XmOTHER** means that the callback procedure must get further information from the **XmDropProcCallbackStruct** in the *destination_data* member.

flags Indicates whether or not the destination widget is also the source of the data to be transferred. Following are the possible values:

XmCONVERTING_NONE
The destination widget is not the source of the data to be transferred.

XmCONVERTING_SAME
The destination widget is the source of the data to be transferred.

transfer_id Serves as a unique ID to identify the transfer transaction.

destination_data
Contains information about the destination. When the selection is _MOTIF_DROP, the callback procedures are called by the drop site's **XmNdropProc**, and *destination_data* is a pointer to the **XmDropProcCallbackStruct** passed to the **XmNdropProc** procedure. When the selection is *SECONDARY*, *destination_data* is an Atom representing a target recommmended by the selection owner for use in converting the selection. Otherwise, *destination_data* is NULL.

location_data
Contains information about the location where data is to be transferred. The value is always NULL when the selection is *CLIPBOARD*. If the value is NULL, the data is to be inserted at the widget's cursor position. Otherwise, the value is a pointer to an *XPoint* structure containing the

x and y coordinates at the location where the data is to be transferred. Once *XmTransferDone* procedures start to be called, **location_data** will no longer be stable.

time Indicates the time when the transfer operation began.

Translations

The **XmTextField** translations are described in the following list. The actions represent the effective behavior of the associated events, and they may differ in a right-to-left language environment.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

≈c s ≈m ≈a <Btn1Down>:
extend-start()

c ≈s ≈m ≈a <Btn1Down>:
move-destination()

≈c ≈s ≈m ≈a <Btn1Down>:
grab-focus()

≈c ≈m ≈a <Btn1Motion>:
extend-adjust()

≈c ≈m ≈a <Btn1Up>:
extend-end()

<Btn2Down>:
process-bdrag()

m ≈a <Btn2Motion>:
secondary-adjust()

≈m a <Btn2Motion>:
secondary-adjust()

s c <Btn2Up>:
link-to()

≈s <Btn2Up>:
copy-to()

≈c <Btn2Up>:
> **move-to()**

:m <Key>osfPrimaryPaste:
> **cut-primary()**

:a <Key>osfPrimaryPaste:
> **cut-primary()**

:<Key>osfPrimaryPaste:
> **copy-primary()**

:m <Key>osfCut:
> **cut-primary()**

:a <Key>osfCut:
> **cut-primary()**

:<Key>osfCut:
> **cut-clipboard()**

:<Key>osfPaste:
> **paste-clipboard()**

:m <Key>osfCopy:
> **copy-primary()**

:a <Key>osfCopy:
> **copy-primary()**

:<Key>osfCopy:
> **copy-clipboard()**

:s <Key>osfBeginLine:
> **beginning-of-line(***extend***)**

:<Key>osfBeginLine:
> **beginning-of-line()**

:s <Key>osfEndLine:
> **end-of-line(***extend***)**

:<Key>osfEndLine:
> **end-of-line()**

:s <Key>osfPageLeft:
> **page-left(***extend***)**

:<Key>osfPageLeft:
    page-left()

:s c<Key>osfPageUp:
    page-left(*extend***)**

:c <Key>osfPageUp:
    page-left()

:s <Key>osfPageRight:
    page-right(*extend***)**

:<Key>osfPageRight:
    page-right()

s c <Key>osfPageDown:
    page-right(*extend***)**

:c <Key>osfPageDown:
    page-right()

:<Key>osfClear:
    clear-selection()

:<Key>osfBackSpace:
    delete-previous-character()

:s m <Key>osfDelete:
    cut-primary()

:s a <Key>osfDelete:
    cut-primary()

:s <Key>osfDelete:
    cut-clipboard()

:c <Key>osfDelete:
    delete-to-end-of-line()

:<Key>osfDelete:
    delete-next-character()

:c m <Key>osfInsert:
    copy-primary()

:c a <Key>osfInsert:
    copy-primary()

:s <Key>osfInsert:
paste-clipboard()

:c <Key>osfInsert:
copy-clipboard()

:<Key>osfInsert:
toggle-overstrike()

:s <Key>osfSelect:
key-select()

:<Key>osfSelect:
set-anchor()

:<Key>osfSelectAll:
select-all()

:<Key>osfDeselectAll:
deselect-all()

:<Key>osfActivate:
activate()

:<Key>osfAddMode:
toggle-add-mode()

:<Key>osfHelp:
Help()

:<Key>osfCancel:
process-cancel()

:s c <Key>osfLeft:
backward-word(*extend***)**

:c <Key>osfLeft:
backward-word()

:s <Key>osfLeft:
key-select(*left***)**

:<Key>osfLeft:
backward-character()

:s c <Key>osfRight:
forward-word(*extend***)**

:c <Key>osfRight:
forward-word()

:s <Key>osfRight:
key-select(*right*)

:<Key>osfRight:
forward-character()

:<Key>osfUp:
traverse-prev()

:<Key>osfDown:
traverse-next()

c ≈m ≈a <Key>slash:
select-all()

c ≈m ≈a <Key>backslash:
deselect-all()

s ≈m ≈a <Key>Tab:
prev-tab-group()

≈m ≈a <Key>Tab:
next-tab-group()

≈s ≈m ≈a <Key>Return:
activate()

c ≈s ≈m ≈a <Key>space:
set-anchor()

c s ≈m ≈a <Key>space:
key-select()

s ≈c ≈m ≈a <Key>space:
self-insert()

<Key>: **self-insert()**

The TextField button event translations are modified when Display's **XmNenableBtn1Transfer** resource does not have a value of **XmOFF** (in other words, it is either **XmBUTTON2_TRANSFER** or **XmBUTTON2_ADJUST**). This option allows the actions for selection and transfer to be integrated on Btn1. The actions for Btn1 that are defined above still apply when the Btn1 event occurs over text that is not selected. The following actions apply when the Btn1 event occurs over text that is selected:

<Btn1Down>:
    process-bdrag().

<Shift><Btn1Down>:
    process-bdrag().

<Btn1Down><Shift><Btn1Up>:
    grab-focus(), **extend-end**.

<Shift><Btn1Down><Shift><Btn1Up>:
    extend-start(), **extend-end()**.

<Ctrl><Btn1Down><Shift><Btn1Up>:
    move-destination().

<Ctrl><Btn1Down>:
    process-bdrag().

When Display's **XmNenableBtn1Transfer** resource has a value of **XmBUTTON2_ADJUST**, the following actions apply:

<Btn2Down>:
    extend-start().

<Btn2Motion>:
    extend-adjust().

<Btn2Up>:
    extend-end().

Action Routines

The **XmTextField** action routines are

activate(): Calls the callbacks for **XmNactivateCallback**. If the parent is a manager, passes the event to the parent.

backward-character(*extend*):
    Moves the insertion cursor one character to the left. This action may have different behavior in a right-to-left language environment.

    If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The backward-character() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the backward-character() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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. 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.

The backward-word() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the backward-word() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

beginning-of-line(*extend*):

If this action is called with no argument, moves the insertion cursor to the beginning of the line.

If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The beginning-of-line() action produces calls to the **XmNmotionVerifyCallback** with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the beginning-of-line() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

clear-selection():

Clears the current selection by replacing each character except Return with a space character.

The clear-selection() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE** and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

copy-clipboard():
If this widget owns the primary selection, this action copies the selection to the clipboard. This action calls the **XmNconvertCallback** procedures, possibly multiple times, for the *CLIPBOARD* selection.

copy-primary():
Copies the primary selection to just before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmCOPY** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection.

In addition, the copy-primary() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. The copy-primary() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

copy-to(): If a secondary selection exists, this action copies the secondary selection to the insertion position of the destination component. If the primary selection is in the destination widget, it will be deselected. Otherwise, there is no effect on the primary selection.

This action calls the destination's **XmNdestinationCallback** procedures for the *SECONDARY* selection and the **XmCOPY** operation. The destination's **XmNdestinationCallback** procedures or the destination component itself invokes the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *SECONDARY* selection.

If no secondary selection exists, this action copies the primary selection to the pointer position. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmCOPY** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection.

In addition, the copy-to() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If there is no secondary selection, the copy-to() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

cut-clipboard():

If this widget owns the primary selection, this action cuts the selection to the clipboard. This action calls the **XmNconvertCallback** procedures, possibly multiple times, for the *CLIPBOARD* selection. If the transfer is successful, this action then calls the **XmNconvertCallback** procedures for the *CLIPBOARD* selection and the *DELETE* target.

In addition, the cut-clipboard() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

cut-primary():

Cuts the primary selection and pastes it just before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmMOVE** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection. If the transfer is successful, this action then calls the selection owner's **XmNconvertCallback** procedures for the *PRIMARY* selection and the *DELETE* target.

In addition, the cut-primary() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**, the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The delete-next-character() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The delete-next-word() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The delete-previous-character() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value

XmCR_VALUE_CHANGED, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The delete-previous-word() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

delete-selection():

Deletes the current selection.

The delete-selection() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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.

The delete-to-end-of-line() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, and the

XmNvalueChangedCallback procedures with reason value **XmCR_VALUE_CHANGED**.

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.

The delete-to-start-of-line() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

deselect-all():

Deselects the current selection.

The deselect-all() action produces no callbacks.

end-of-line(*extend*):

If this action is called with no argument, moves the insertion cursor to the end of the line. If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

The end-of-line() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the end-of-line() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

extend-adjust():

Selects text from the anchor to the pointer position and deselects text outside that range.

The extend-adjust() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. The extend-adjust() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

extend-end():
: Moves the insertion cursor to the position of the pointer. The extend-end() action is used to commit the selection. After this action has been done, process-cancel() will no longer cancel the selection.

  The extend-end() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. The extend-end() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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.

  The extend-start() action can produce no callbacks, however, it may produce calls to the **XmNgainPrimaryCallback** and **XmNmotionVerifyCallback** procedures. See callback description for more information.

forward-character(*extend*):
: Moves the insertion cursor one character to the right. This action may have different behavior in a right-to-left language environment.

  If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

  The forward-character() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the forward-character() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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. 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.

The forward-word() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. If called with the *extend* argument, the forward-word() action may produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

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, and three clicks to select a line of 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.

The grab-focus() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

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(*right*|*left*):

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.

Note that after a **key-select** action, the selection will still begin at the original anchor, and will extend to the position indicated in the action call. If this new position is on the opposite side of the selection anchor from the previous selection boundary, the original selection will be deselected.

The key-select() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. The key-select() action may also produce calls to the **XmNgainPrimaryCallback** procedures. See callback description for more information.

link-primary():
Places a link to the primary selection just before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmLINK** operation. The TextField widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the link to the primary selection and for taking any related actions.

link-to(): If a secondary selection exists, this action places a link to the secondary selection at the insertion position of the destination component. This action calls the destination's **XmNdestinationCallback** procedures for the *SECONDARY* selection and the **XmLINK** operation.

If no secondary selection exists, this action places a link to the primary selection at the pointer position. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmLINK** operation.

The TextField widget itself performs no transfers; the **XmNdestinationCallback** procedures are responsible for inserting the link to the primary or secondary selection and for taking any related actions.

move-destination():
Moves the insertion cursor to the pointer position without changing any existing current selection. If there is a current selection, sets the widget as the destination widget. This also moves the widget focus to match the insertion cursor.

The move-destination() action produces calls to the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

move-to(): If a secondary selection exists, this action moves the secondary selection to the insertion position of the destination component. If the secondary selection is in the destination widget, and the secondary selection and the primary selection overlap, the result is undefined. This action calls the destination's **XmNdestinationCallback** procedures for the *SECONDARY* selection and the **XmCOPY** operation. The destination's **XmNdestinationCallback** procedures or the destination component itself invokes the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *SECONDARY* selection. If the transfer is successful, this action then calls the selection owner's **XmNconvertCallback** procedures for the *SECONDARY* selection and the *DELETE* target.

If no secondary selection exists, this action moves the primary selection to the pointer position. This action calls the **XmNdestinationCallback** procedures for the *PRIMARY* selection and the **XmMOVE** operation. It calls the selection owner's **XmNconvertCallback** procedures, possibly multiple times, for the *PRIMARY* selection. If the transfer is successful, this action then calls the selection owner's **XmNconvertCallback** procedures for the *PRIMARY* selection and the *DELETE* target.

In addition, the move-to() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**. This action may also produce calls to the **XmNgainPrimaryCallback** procedures.

next-tab-group():
Traverses to the next tab group.

The next-tab-group() action produces no callbacks, unless it results in the widget losing focus, in which case, the **XmNlosingFocusCallback** procedures are called with reason value **XmCR_LOSING_FOCUS**.

page-left(): Scrolls the viewing window left one page of text.

The page-left() action produces no callbacks.

page-right(): Scrolls the viewing window right one page of text.

The page-right() action produces no callbacks.

paste-clipboard():
Pastes the contents of the clipboard before the insertion cursor. This action calls the **XmNdestinationCallback** procedures for the *CLIPBOARD* selection and the **XmCOPY** operation.

The paste-clipboard() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

prev-tab-group():
Traverses to the previous tab group.

The prev-tab-group() action produces no callbacks, unless it results in the widget losing focus, in which case, the **XmNlosingFocusCallback** procedures are called with reason value **XmCR_LOSING_FOCUS**.

process-bdrag()
If the pointer is within the selection, this action starts a drag operation for the selection. This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures, possibly multiple times, for the _MOTIF_DROP selection.

If no selection exists or the pointer is outside the selection, this action prepares to start a secondary selection at the pointer position.

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, it passes the event to the parent.

secondary-adjust():
Extends the secondary selection to the pointer position.

The secondary-adjust() action produces no callbacks.

secondary-start():
Marks the beginning of a secondary selection.

The secondary-start() action produces no callbacks.

select-all():
Selects all text.

The select-all() action can produce no callbacks, however, it may produce calls to the **XmNgainPrimaryCallback** and **XmNmotionVerifyCallback** procedures. See callback description for more information.

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.

The self-insert() action produces calls to the **XmNmodifyVerifyCallback** procedures with reason value **XmCR_MODIFYING_TEXT_VALUE**, the **XmNvalueChangedCallback** procedures with reason value **XmCR_VALUE_CHANGED**, and the **XmNmotionVerifyCallback** procedures with reason value **XmCR_MOVING_INSERT_CURSOR**.

set-anchor(): Resets the anchor point for extended selections. Resets the destination of secondary selection actions.

The set-anchor() action produces no callbacks.

toggle-add-mode():
Toggles the state of Add Mode.

The toggle-add-mode() action produces no callbacks.

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.

The following traversal actions generate no callbacks unless they result in the loss of focus by the widget in question, as when **XmNnavigationType** is **XmNONE**. In this case, they produce calls to the **XmNlosingFocusCallback** procedures, with reason value **XmCR_FOCUS_MOVED**.

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. For information about bindings for virtual buttons and keys, see **VirtualBindings**(3).

Related Information

Core(3), **XmCreateTextField**(3), **XmFontList**(3), **XmFontListAppendEntry**(3), **XmPrimitive**(3), **XmTextFieldClearSelection**(3), **XmTextFieldCopy**(3), **XmTextFieldCopyLink**(3), **XmTextFieldCut**(3), **XmTextFieldGetBaseline**(3), **XmTextFieldGetEditable**(3), **XmTextFieldGetInsertionPosition**(3), **XmTextFieldGetLastPosition**(3), **XmTextFieldGetMaxLength**(3), **XmTextFieldGetSelection**(3), **XmTextFieldGetSelectionPosition**(3), **XmTextFieldGetSelectionWcs**(3), **XmTextFieldGetString**(3), **XmTextFieldGetStringWcs**(3), **XmTextFieldGetSubstring**(3), **XmTextFieldGetSubstringWcs**(3), **XmTextFieldInsert**(3), **XmTextFieldInsertWcs**(3), **XmTextFieldPaste**(3), **XmTextFieldPasteLink**(3), **XmTextFieldPosToXY**(3), **XmTextFieldRemove**(3), **XmTextFieldReplace**(3), **XmTextFieldReplaceWcs**(3), **XmTextFieldSetAddMode**(3), **XmTextFieldSetEditable**(3), **XmTextFieldSetHighlight**(3), **XmTextFieldSetInsertionPosition**(3), **XmTextFieldSetMaxLength**(3), **XmTextFieldSetSelection**(3), **XmTextFieldSetString**(3), **XmTextFieldSetStringWcs**(3), **XmTextFieldShowPosition**(3), and **XmTextFieldXYToPos**(3).

XmToggleButton

Purpose

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, diamond, or round) 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**, **N-of-many**, or **1-of-many-round** selection state. When a toggle indicator is displayed, a square indicator shows an **N-of-many** selection state, a diamond-shaped indicator shows a **1-of-many** selection state, and a circle-shaped indicator shows a **1-of-many-round** selection state.

ToggleButton implies a set or unset state. In the case of a label and an indicator, an empty indicator (square, diamond, or round) indicates that ToggleButton is unset, and a filled indicator shows that it is set. The indicator may be filled with a check mark, a cross, or the select color. In the case of a pixmap toggle, different pixmaps are used to display the set/unset states. ToggleButton can also indicate an indeterminate state. In the case of a label and an indicator, an indeterminate state is indicated by a stippled flat box. In the case of a pixmap toggle, a different pixmap is used to display the indeterminate state.

The default behavior associated with a ToggleButton in a menu depends on the type of menu system in which it resides. By default, Btn1 controls the behavior of the ToggleButton. In addition, Btn3 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.

ToggleButton uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits.

Classes

ToggleButton inherits behavior, resources, and traits 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	Type	Default	Access
XmNarmCallback	XmCArmCallback	XtCallbackList	NULL	C
XmNdetailShadow-Thickness	XmCDetailShadow-Thickness	Dimension	2	CSG
XmNdisarmCallback	XmCDisarmCallback	XtCallbackList	NULL	C
XmNfillOnSelect	XmCFillOnSelect	Boolean	dynamic	CSG
XmNindeterminate-Pixmap	XmCIndeterminate-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNindicatorOn	XmCIndicatorOn	unsigned char	XmINDICATOR_- FILL	CSG
XmNindicatorSize	XmCIndicatorSize	Dimension	dynamic	CSG
XmNindicatorType	XmCIndicatorType	unsigned char	dynamic	CSG
XmNselectColor	XmCSelectColor	Pixel	dynamic	CSG
XmNselectInsensitive-Pixmap	XmCSelect-InsensitivePixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNselectPixmap	XmCSelectPixmap	Pixmap	XmUNSPECIFIED_P-IXMAP	CSG
XmNset	XmCSet	unsigned char	XmUNSET	CSG

XmNspacing	XmCSpacing	Dimension	4	CSG
XmNtoggleMode	XmCToggleMode	unsigned char	XmTOGGLE_-BOOLEAN	CSG
XmNunselectColor	XmCUnselectColor	Pixel	dynamic	CSG
XmNvalueChanged-Callback	XmCValueChanged-Callback	XtCallbackList	NULL	C
XmNvisibleWhenOff	XmCVisibleWhenOff	Boolean	dynamic	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**.

XmNdetailShadowThickness

Specifies the thickness of the indicator shadow. The default thickness is 2 pixels.

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. If unset, fills the indicator with the unselect color. If indeterminate, fills the indicator with half select color and half unselect color. Otherwise, it switches only the top and bottom shadow colors. The default is True only if a box type of indicator (such as a check box) is specified, or if the **XmNindicatorType** is a **1-of** type and a toggle indicator is drawn.

If **XmNfillOnSelect** is True, **XmNset** is **XmSET**, and **XmNindicatorOn** is **XmINDICATOR_NONE** the ToggleButton's background is set to **XmNselectColor**. For the other **XmNindicatorOn** values, only the indicator is filled with **XmNselectColor**.

XmNindeterminateInsensitivePixmap

Specifies the pixmap to be displayed as the button face when the Label *XmNlableType* resource is **XmPIXMAP**, the ToggleButton **XmNset** resource is **XmINDETERMINATE**, and the **Core XmNsensitive** resource is False.

XmNindeterminatePixmap

Specifies the pixmap to be displayed as the button face when the Label *XmNlableType* resource is **XmPIXMAP**, the ToggleButton **XmNset** resource is **XmINDETERMINATE**, and the **Core XmNsensitive** resource is True.

XmNindicatorOn

Specifies that if a toggle indicator is to be drawn, it will be drawn to one side of the toggle text or pixmap, depending on the **XmNlayoutDirection** resource of the widget. The default value is **XmINDICATOR_FILL**. Toggles accept the following values:

XmINDICATOR_NONE

No space is allocated for the indicator, and it is not displayed. Any shadows around the entire widget are switched when the toggle is selected or unselected.

XmINDICATOR_BOX

The toggle indicator is in the shape of a shadowed box.

XmINDICATOR_FILL

If the value of the **XmDisplay XmNenableToggleVisual** resource is **True** , the visuals are those of *XmINDICATOR_CHECK_BOX* ; if **False** , the indicator visuals are those of *XmINDICATOR_BOX*.

XmINDICATOR_CHECK

The toggle indicator is in the shape of a checkmark in the foreground color.

XmINDICATOR_CHECK_BOX

The toggle indicator is in the shape of a checkmark enclosed in a box. This is the default if the **XmDisplay XmNenableToggleVisual** resource is set.

XmINDICATOR_CROSS_BOX

The toggle indicator is in the shape of a cross enclosed in a box.

XmINDICATOR_CROSS
The toggle indicator is in the shape of a cross.

All ToggleButton checks and crosses should be drawn in the foreground color.

If this resource is not **XmINDICATOR_NONE**, it will control the appearance of the toggle visual. If **XmNset** is **XmINDETERMINATE** and **XmNindicatorOn** is not **XmINDICATOR_NONE**, this resource shows a stippled flat box. If **XmNset** is **XmINDETERMINATE**, **XmNindicatorOn** is **XmINDICATOR_NONE**, and **XmNtoggleMode** is **XmTOGGLE_INDETERMINATE**, the label and the ToggleButton are stippled with a combination of the **XmNselectColor** and the **XmNunselectColor** color, and the border is flat.

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. The size of indicators inside menus may differ from those outside of menus. Note that a change in this resource may also cause a change in the values of the inherited resources **XmNmarginTop**, **XmNmarginBottom**, and **XmNmarginLeft**.

XmNindicatorType
Specifies if the indicator is a **1-of** or **N-of** indicator. For the **1-of** indicator, the value can be **XmONE_OF_MANY**, **XmONE_OF_MANY_ROUND**, or **XmONE_OF_MANY_DIAMOND**. For the **N-of** indicator, the value is **XmN_OF_MANY**. This value specifies only the visuals and does not enforce the behavior. When the ToggleButton is in a radio box, the default is **XmONE_OF_MANY**; otherwise, the default is **XmN_OF_MANY**. Legal values are:

XmONE_OF_MANY
When the Display **XmNenableToggleVisual** resource is set, indicators are drawn with the same appearance as **XmONE_OF_MANY_ROUND**; otherwise, they appear the same as **XmONE_OF_MANY_DIAMOND**.

XmN_OF_MANY
The indicators are drawn as specified by the **XmNindicatorOn** resource.

XmONE_OF_MANY_ROUND
A shadowed circle.

XmONE_OF_MANY_DIAMOND
A shadowed diamond.

XmNselectColor
Allows the application to specify what color fills the center of the square, diamond-shaped, or round 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. The results of this resource depend on the value of the Display resource **XmNenableToggleColor**. A value of True causes the fill color to use the **XmHIGHLIGHT_COLOR** color by default. A value of False causes the fill color to use the background color. 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 **XmINDICATOR_NONE**, the value of **XmNfillOnSelect** must be explicitly set to True.

This resource is also used as the background color when all of the following conditions are met: the button is armed in a menu, the **XmNenableEtchedInMenu** resource is **True**, the **XmNindicatorOn** resource is **False**, and the **XmNfillOnSelect** resource is **True**.

This resource can take the following values:

XmDEFAULT_SELECT_COLOR
Is the same as the current dynamic default, which is a color between the background and the bottom shadow color.

XmREVERSED_GROUND_COLORS
Forces the select color to the foreground color and causes the default color of any text rendered over the select color to be in the background color.

XmHIGHLIGHT_COLOR
Forces the fill color to use the highlight color.

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 **XmUNSET** indicates that the ToggleButton is not set. A value of **XmSET** indicates that the ToggleButton is set. A value of **XmINDETERMINATE** indicates that the ToggleButton is in an indeterminate state (neither set nor unset). The ToggleButton states cycle through in the order of **XmSET**, **XmINDETERMINATE** (if **XmNtoggleMode** is set to **XmTOGGLE_INDETERMINATE**), and **XmUNSET**, and then back around to **XmSET**. If **XmNtoggleMode** is set to **XmTOGGLE_BOOLEAN**, then the ToggleButton states cycle through in the order of **XmSET**, then **XmUNSET**, and then back around to **XmSET**. 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).

XmNtoggleMode
Specifies the mode of the ToggleButton as either **XmTOGGLE_BOOLEAN** or **XmTOGGLE_INDETERMINATE**. The **XmTOGGLE_INDETERMINATE** value allows the **XmNset** resource to be able to accept the values **XmINDETERMINATE**, **XmSET**, and **XmUNSET**. The **XmNtoggleMode** resource is forced to **XmTOGGLE_BOOLEAN** if the toggle is in an **XmRowColumn** widget whose radio behavior is **XmONE_OF_MANY**. In

XmTOGGLE_BOOLEAN mode, the **XmNset** resource can only accept **XmSET** and **XmUNSET**.

XmNunselectColor
Allows the application to specify what color fills the center of the square, diamond-shaped, or round indicator when it is not 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 **XmNbackground**. For a monochrome display, the default is set to the background color. To set the background of the button to **XmNunselectColor** when **XmNindicatorOn** is **XmINDICATOR_NONE**, the value of **XmNfillOnSelect** must be explicitly set to True. This resource acts like the **XmNselectColor** resource, but for the case when **XmNset** is **XmUNSET**.

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 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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	CSG
XmNacceleratorText	XmCAcceleratorText	XmString	NULL	CSG
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG

XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelPixmap	XmCLabelPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	dynamic	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	dynamic	CSG
XmNmarginRight	XmCMarginRight	Dimension	0	CSG
XmNmarginTop	XmCMarginTop	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonicCharSet	XmCMnemonic- CharSet	String	XmFONTLIST_-DEFAULT_TAG	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmString-Direction	dynamic	CSG

XmPrimitive Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadow- Color	XmCBottom-ShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottom-ShadowPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNconvertCallback	XmCCallback	XtCallbackList	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallbackList	NULL	C
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlight-OnEnter	Boolean	False	CSG

XmNhighlightPixmap	XmCHighlight-Pixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	2	CSG
XmNlayoutDirection	XmCLayout-Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigation-Type	XmNavigationType	XmNONE	CSG
XmNpopupHandler-Callback	XmCCallback	XtCallbackList	NULL	C
XmNshadowThickness	XmCShadow-Thickness	Dimension	dynamic	CSG
XmNtopShadowColor	XmCTopShadow-Color	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

Core Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	0	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CG
XmNdepth	XmCDepth	int	dynamic	CG
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C

XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNinitialResources-Persistent	XmCInitialResources-Persistent	Boolean	True	C
XmNmappedWhen-Managed	XmCMappedWhen-Managed	Boolean	True	CSG
XmNscreen	XmCScreen	Screen *	dynamic	CG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNtranslations	XmCTranslations	XtTranslations	dynamic	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	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 state, either **XmSET** (selected), **XmUNSET** (unselected), or **XmINDETERMINATE** (neither). Note that the reported state is the state that the ToggleButton is in after the *event* has been processed. For example, suppose that a user clicks on a ToggleButton to change it from the unselected state to the selected state. In this case, ToggleButton changes the value of *set* from **XmUNSET** to **XmSET** prior to calling the callback.

Translations

XmToggleButton includes translations from *Primitive*. Additional **XmToggleButton** translations for buttons not in a menu system are described in the following list.

Note that altering translations in **#override** or **#augment** mode is undefined.

The following key names are listed in the X standard key event translation table syntax. This format is the one used by Motif to specify the widget actions corresponding to a given key. A brief overview of the format is provided under **VirtualBindings**(3). For a complete description of the format, please refer to the X Toolkit Instrinsics Documentation.

c<Btn1Down>:
    ButtonTakeFocus()

≈c<Btn1Down>:
    Arm()

≈c<Btn1Up>:
    Select() Disarm()

<Btn2Down>:
    ProcessDrag()

:<Key>osfActivate:
    PrimitiveParentActivate()

:<Key>osfCancel:
    PrimitiveParentCancel()

:<Key>osfSelect:
    ArmAndActivate()

:<Key>osfHelp:
    Help()

≈s ≈m ≈a <Key>Return:
    PrimitiveParentActivate()

≈s ≈m ≈a <Key>space:
    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, Btn3 also performs the Btn1 actions.

<Btn2Down>:
    ProcessDrag()

c<Btn1Down>:
    MenuButtonTakeFocus()

c<Btn1Up>:
: **MenuButtonTakeFocusUp()**

≈c<BtnDown>:
: **BtnDown()**

≈c<BtnUp>:
: **BtnUp()**

:<Key>osfSelect:
: **ArmAndActivate()**

:<Key>osfActivate:
: **ArmAndActivate()**

:<Key>osfHelp:
: **Help()**

:<Key>osfCancel:
: **MenuEscape()**

≈s ≈m ≈a <Key>Return:
: **ArmAndActivate()**

≈s ≈m ≈a <Key>space:
: **ArmAndActivate()**

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.

ButtonTakeFocus():
: Causes the ToggleButton to take keyboard focus when **Ctrl<Btn1Down>** is pressed, without activating the widget.

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 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.

ProcessDrag():
: Drags the contents of a ToggleButton label, identified when **BTransfer** is pressed. This action sets the **XmNconvertProc** of the DragContext to a function that calls the **XmNconvertCallback** procedures, possibly multiple times, for the _MOTIF_DROP selection. 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**(3).

Related Information

Core(3), **XmCreateRadioBox**(3), **XmCreateToggleButton**(3), **XmLabel**(3), **XmPrimitive**(3), **XmRowColumn**(3), **XmToggleButtonGetState**(3), and **XmToggleButtonSetState**(3).

XmToggleButtonGadget

Purpose

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, diamond, or round) 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**, **N-of-many**, or **N-of-many-round** selection state. When a toggle indicator is displayed, a square indicator shows an **N-of-many** selection state, a diamond-shaped indicator shows a **1-of-many** selection state, and a circle-shaped indicator shows a **1-of-many-round** selection state.

ToggleButtonGadget implies a set or unset state. In the case of a label and an indicator, an empty indicator (square, diamond, or round) indicates that ToggleButtonGadget is unset, and a filled indicator shows that it is set. The indicator may be filled with a check mark or the select color. In the case of a pixmap toggle, different pixmaps are used to display the set/unset states. ToggleButtonGadget can also indicate an indeterminate state. In the case of a label and an indicator, an indeterminate state is indicated by a stippled flat box. In the case of a pixmap toggle, a different pixmap is used to display the indeterminate state.

The default behavior associated with a ToggleButtonGadget in a menu depends on the type of menu system in which it resides. By default, Btn1 controls the behavior of the ToggleButtonGadget. In addition, Btn3 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.

ToggleButtonGadget uses the *XmQTmenuSystem* and *XmQTspecifyRenderTable* traits.

Classes

ToggleButtonGadget inherits behavior, resources, and traits 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	Type	Default	Access
XmNarmCallback	XmCArmCallback	XtCallbackList	NULL	C
XmNdetailShadow-Thickness	XmCDetailShadow-Thickness	Dimension	2	CSG
XmNdisarmCallback	XmCDisarmCallback	XtCallbackList	NULL	C
XmNfillOnSelect	XmCFillOnSelect	Boolean	dynamic	CSG
XmNindeterminate-Pixmap	XmCIndeterminate- Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNindicatorOn	XmCIndicatorOn	unsigned char	XmINDICATOR_-FILL	CSG
XmNindicatorSize	XmCIndicatorSize	Dimension	dynamic	CSG
XmNindicatorType	XmCIndicatorType	unsigned char	dynamic	CSG
XmNselectColor	XmCSelectColor	Pixel	dynamic	CSG
XmNselectInsensitive-Pixmap	XmCSelectInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNselectPixmap	XmCSelectPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG

XmNset	XmCSet	unsigned char	XmUNSET	CSG
XmNspacing	XmCSpacing	Dimension	4	CSG
XmNtoggleMode	XmCToggleMode	unsigned char	XmTOGGLE_-BOOLEAN	CSG
XmNunselectColor	XmCUnselectColor	Pixel	dynamic	CSG
XmNvalueChanged-Callback	XmCValueChanged-Callback	XtCallbackList	NULL	C
XmNvisibleWhenOff	XmCVisibleWhenOff	Boolean	dynamic	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**.

XmNdetailShadowThickness
Specifies the thickness of the indicator shadow. The default thickness is 2 pixels.

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. If unset, fills the indicator with the unselect color. If indeterminate, fills the indicator with half select color and half unselect color. Otherwise, it switches only the top and bottom shadow colors. The default is set True only if a box type of indicator (such as a check box) is specified, or if the **XmNindicatorType** is a **1-of** type and a toggle indicator is drawn. If **XmNfillOnSelect** is True, **XmNset** is **XmSET**, and **XmNindicatorOn** is **XmINDICATOR_NONE** the ToggleButtonGadget's background is set to **XmNselectColor**. For the other **XmNindicatorOn** values, only the indicator is filled with **XmNselectColor**.

XmNindeterminateInsensitivePixmap

Specifies the pixmap to be displayed as the button face when the Label *XmNlableType* resource is **XmPIXMAP**, the ToggleButtonGadget **XmNset** resource is **XmINDETERMINATE**, and the **Core XmNsensitive** resource is False.

XmNindeterminatePixmap

Specifies the pixmap to be displayed as the button face when the Label *XmNlableType* resource is **XmPIXMAP**, the ToggleButtonGadget **XmNset** resource is **XmINDETERMINATE**, and the **Core XmNsensitive** resource is True.

XmNindicatorOn

Specifies that if a toggle indicator is to be drawn, it will be drawn to one side of the toggle text or pixmap, depending on the **XmNlayoutDirection** resource of the widget. The default value is **XmINDICATOR_FILL**. Toggles accept the following values:

XmINDICATOR_NONE

No space is allocated for the indicator, and it is not displayed. Any shadows around the entire widget are switched when the toggle is selected or unselected.

XmINDICATOR_BOX

The toggle indicator is in the shape of a shadowed box.

XmINDICATOR_FILL

If the value of the **XmDisplay XmNenableToggleVisual** resource is **True** , the visuals are those of *XmINDICATOR_CHECK_BOX* ; if **False** , the indicator visuals are those of *XmINDICATOR_BOX*.

XmINDICATOR_CHECK

The toggle indicator is in the shape of a checkmark in the foreground color.

XmINDICATOR_CHECK_BOX

The toggle indicator is in the shape of a checkmark enclosed in a box. This is the default if the **XmDisplay XmNenableToggleVisual** resource is set.

XmINDICATOR_CROSS_BOX

The toggle indicator is in the shape of a cross enclosed in a box.

XmINDICATOR_CROSS
The toggle indicator is in the shape of a cross.

All ToggleButton checks and crosses should be drawn in the foreground color.

If this resource is not **XmINDICATOR_NONE**, it will control the appearance of the toggle visual. If **XmNset** is **XmINDETERMINATE** and **XmNindicatorOn** is not **XmINDICATOR_NONE**, this resource shows a stippled flat box. If **XmNset** is **XmINDETERMINATE**, **XmNindicatorOn** is **XmINDICATOR_NONE**, and **XmNtoggleMode** is **XmTOGGLE_INDETERMINATE**, the label and the ToggleButton are stippled with a combination of the **XmNselectColor** and the **XmNunselectColor** color, and the border is flat.

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. The size of indicators inside menus may differ from those outside of menus. Note that a change in this resource may also cause a change in the values of the inherited resources **XmNmarginTop**, **XmNmarginBottom**, and **XmNmarginLeft**.

XmNindicatorType
Specifies if the indicator is a **1-of** or **N-of** indicator. For the **1-of** indicator, the value can be **XmONE_OF_MANY**, **XmONE_OF_MANY_ROUND**, or **XmONE_OF_MANY_DIAMOND**. For the **N-of** indicator, the value is **XmN_OF_MANY**. This value specifies only the visuals and does not enforce the behavior. When the ToggleButton is in a radio box, the default is **XmONE_OF_MANY**; otherwise, the default is **XmN_OF_MANY**. Legal values are:

XmONE_OF_MANY
When the Display **XmNenableToggleVisual** resource is set, indicators are drawn with the same appearance as **XmONE_OF_MANY_ROUND**; otherwise, they appear the same as **XmONE_OF_MANY_DIAMOND**.

XmN_OF_MANY
The indicators are drawn as specified by the **XmNindicatorOn** resource.

XmONE_OF_MANY_ROUND
A shadowed circle.

XmONE_OF_MANY_DIAMOND
A shadowed diamond.

XmNselectColor
Allows the application to specify what color fills the center of the square, diamond-shaped, or round 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. The results of this resource depend on the value of the Display resource **XmNenableToggleColor**. A value of True causes the fill color to use the **XmHIGHLIGHT_COLOR** color by default. A value of False causes the fill color to use the background color. 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 **XmINDICATOR_NONE**, the value of **XmNfillOnSelect** must be explicitly set to True.

This resource can take the following values:

XmDEFAULT_SELECT_COLOR
Is the same as the current dynamic default, which is a color between the background and the bottom shadow color.

XmREVERSED_GROUND_COLORS
Forces the select color to the foreground color and causes the default color of any text rendered over the select color to be in the background color.

XmHIGHLIGHT_COLOR
Forces the fill color to use the highlight color.

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 **XmUNSET** indicates that the ToggleButton is not set. A value of **XmSET** indicates that the ToggleButton is set. A value of **XmINDETERMINATE** indicates that the ToggleButton is in an indeterminate state (neither set nor unset). The ToggleButton states cycle through in the order of **XmSET**, **XmINDETERMINATE** (if **XmNtoggleMode** is set to **XmTOGGLE_INDETERMINATE**), and **XmUNSET**, and then back around to **XmSET**. If **XmNtoggleMode** is set to **XmTOGGLE_BOOLEAN**, then the ToggleButton states cycle through in the order of **XmSET**, then **XmUNSET**, and then back around to **XmSET**. 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).

XmNtoggleMode
Specifies the mode of the ToggleButtonGadget as either **XmTOGGLE_BOOLEAN** or **XmTOGGLE_INDETERMINATE**. The **XmTOGGLE_INDETERMINATE** value allows the **XmNset** resource to be able to accept the values **XmINDETERMINATE**, **XmSET**, and **XmUNSET**. The **XmNtoggleMode** resource is forced to **XmTOGGLE_BOOLEAN** if the toggle is in an **XmRowColumn** widget whose radio behavior is **XmONE_OF_MANY**. In **XmTOGGLE_BOOLEAN** mode, the **XmNset** resource can only accept **XmSET** and **XmUNSET**.

XmNunselectColor

Allows the application to specify what color fills the center of the square, diamond-shaped, or round indicator when it is not 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 **XmNbackground**. For a monochrome display, the default is set to the background color. To set the background of the button to **XmNunselectColor** when **XmNindicatorOn** is **XmINDICATOR_NONE**, the value of **XmNfillOnSelect** must be explicitly set to True. This resource acts like the **XmNselectColor** resource, but for the case when **XmNset** is **XmUNSET**.

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**	**Type**	**Default**	**Access**
XmNaccelerator	XmCAccelerator	String	NULL	CSG
XmNacceleratorText	XmCAccelerator- Text	XmString	NULL	CSG
XmNalignment	XmCAlignment	unsigned char	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlabelInsensitive-Pixmap	XmCLabelInsensitive-Pixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG

XmNlabelPixmap	XmCLabelPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNlabelString	XmCXmString	XmString	dynamic	CSG
XmNlabelType	XmCLabelType	unsigned char	XmSTRING	CSG
XmNmarginBottom	XmCMarginBottom	Dimension	dynamic	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginLeft	XmCMarginLeft	Dimension	dynamic	CSG
XmNmarginRight	XmCMarginRight	Dimension	0	CSG
XmNmarginTop	XmCMarginTop	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonic- CharSet	XmCMnemonic-CharSet	String	dynamic	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CSG

XmGadget Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackground- Pixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_-PIXMAP	CSG
XmNbottomShadow-Color	XmCBottomShadowColor	Pixel	dynamic	CSG
XmNbottomShadow-Pixmap	XmCBottomShadow-Pixmap	Pixmap	dynamic	CSG
XmNhelpCallback	XmCCallback	XtCallback- List	NULL	C
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOn- Enter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlight-Thickness	Dimension	2	CSG

XmNlayoutDirection	XmNCLayout- Direction	XmDirection	dynamic	CG
XmNnavigationType	XmCNavigationType	XmNavigation-Type	XmNONE	CSG
XmNshadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadow-Pixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

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

Object Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	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 state, either **XmSET** (selected), **XmUNSET** (unselected), or **XmINDETERMINATE** (neither). Note that the reported state is the state that the ToggleButtonGadget is in after the *event* has been processed. For example, suppose that a user clicks on a ToggleButtonGadget to change it from the unselected state to the selected state. In this case, ToggleButtonGadget changes the value of *set* from **XmUNSET** to **XmSET** prior to calling the callback.

Behavior

XmToggleButtonGadget includes behavior from **XmGadget**. **XmToggleButtonGadget** includes menu traversal behavior from **XmLabelGadget**. Additional **XmToggleButtonGadget** behavior is described in the following list:

Btn2Down: Drags the contents of a ToggleButtonGadget label, identified when Btn2 is pressed. This action is undefined for ToggleButtonGadgets used in a menu system.

Btn1Down: 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.

Btn1Up: 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.

KeyosfHelp: In a Pulldown or Popup 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.

KeyosfSelect:

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.

KeyosfCancel:

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, 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**(3).

Related Information

Object(3), **RectObj**(3), **XmCreateRadioBox**(3), **XmCreateToggleButtonGadget**(3), **XmGadget**(3), **XmLabelGadget**(3), **XmRowColumn**(3), **XmToggleButtonGadgetGetState**(3), and **XmToggleButtonGadgetSetState**(3).