Technical Standard

Motif Toolkit API

THE Open GROUP

[This page intentionally left blank]

X/Open CAE Specificaton

Motif Toolkit API

X/Open Company Ltd.

X/Open CAE Specificaton

Motif Toolkit API

ISBN: 1-85912-024-5
X/Open Document Number: C320

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

List of Tables

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

```
request corrigenda; topic index
```

This will return the index of publications for which Corrigenda exist.

This Document

This document is a CAE Specification (see above). It defines the interfaces offered to X Windows application programs by the X/Open Motif Toolkit application programming interface (API). It defines these interfaces and their run-time behaviour without imposing any particular restrictions on the way in which the interfaces are implemented.

Structure

This document is organised as follows:

- Chapter 1 is an introduction covering conformance, terminology, related standards and the layout of manual pages for services.
- Chapter 2 provides the general definition and the rationale for inclusion and specification of services.
- Chapter 3 contains the reference information for X/Open Motif commands.
- Chapter 4 defines the data types used in X/Open Motif.
- Chapter 5 contains the reference information for X/Open Motif widgets and functions.
- Appendix A contains the header files information for X/Open Motif.
- A glossary and index are provided.

Intended Audience

This document is intended for:

- software engineers developing applications that are compliant with X/Open Motif to run on implementations that are compliant with X/Open Motif
- software engineers developing implementations that are compliant with X/Open Motif on which applications tht are compliant with X/Open Motif can run
- organisations (for example, standards-setting bodies) for whom X/Open Motif (or some part of it) is an appropriate part of the formal, *de jure* process.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures, resources and class pointers.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — utility names
  - — functions; these are shown as follows: *name*( )
  - — widget class names.
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- The notation [RABCD] is used to identify a return value RABCD.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font.
- Variables within syntax statements are shown in *`italic fixed width font`*.
- Shading is used as defined in Section 1.2 on page 2.

In contrast with normal X/Open style, this document uses American English spelling for consistency with the software.

Trade Marks

Referenced Documents

The following documents are referenced in this specification:

ISO C

ISO / IEC 9899: 1990, Programming Languages — C (technically identical to ANSI standard X3.159-1989).

ISO 8859-1

ISO 8859-1: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets — Part 1: Latin Alphabet No. 1.

Programmers' Guide

OSF / Motif Programmers' Guide, Revision 1.2, Open Software Foundation, ISBN 0-13-643107-0, published by Prentice Hall.

X11 Release 5 documentation as follows:

X Protocol

X / Open CAE Specification, Window Management (X11R5): X Window System Protocol, April 1995 (ISBN: 1-85912-087-3, C507).

Xlib

X / Open CAE Specification, Window Management (X11R5): Xlib — C Language Binding, April 1995 (ISBN: 1-85912-088-1, C508).

X Toolkit Intrinsics

X / Open CAE Specification, Window Management (X11R5): X Toolkit Intrinsics, April 1995 (ISBN: 1-85912-089-X, C509).

ICCCM

X / OpenCAE Specification, Window Management (X11R5): File Formats and Application Conventions, April 1995 (ISBN: 1-85912-090-3, C510).

Chapter 1

Introduction

This document defines the interfaces offered to X Windows application programs by the X/Open Motif Toolkit. It defines these interfaces and their run-time behaviour without imposing any particular restrictions on the way in which the interfaces are implemented.

The interfaces are defined in ISO C. It is possible that some implementations may make the interfaces available to languages other than C, but this document does not currently define bindings for other languages.

This specification allows an application to be built using a set of services that are consistent across all systems that conform to this specification (see Section 1.1). Applications written in C using only these interfaces and avoiding implementation-dependent constructs are portable to all systems that conform to this specification.

This chapter specifies conformance requirements, defines terminology, explains this document's relationship to standards, and describes the format of the manual pages in Chapter 3 and Chapter 5.

1.1 Conformance

An implementation conforming to this document shall meet the following criteria:

- The system shall support all the interfaces and headers defined within this specification.
- The system may provide additional or enhanced interfaces, headers and facilities not required by this specification, provided that such additions or enhancements do not affect the behaviour of an application that requires only the facilities described in this document.

Some of the text in this document refers to filenames, pathnames, and so on that are specific to the UNIX operating system; these are examples. This specification assumes that a conforming Motif implementation uses an XSI-conformant operating system (see the **XPG4**, Version 2 specifications). For operating systems that are not XSI-conformant, operating system resources are undefined.

1.2 Terminology

The following terms are used in this specification:

can

This describes a permissible optional feature or behaviour available to the user or application; all systems support such features or behaviour as mandatory requirements.

implementation-dependent

The value or behaviour is not consistent across all implementations. The provider of an implementation normally documents the requirements for correct program construction and correct data in the use of that value or behaviour. When the value or behaviour in the implementation is designed to be variable or customisable on each instantiation of the system, the provider of the implementation normally documents the nature and permissible ranges of this variation. Applications that are intended to be portable must not rely on implementation-dependent values or behaviour.

may

With respect to implementations, the feature or behaviour is optional. Applications should not rely on the existence of the feature. To avoid ambiguity, the reverse sense of *may* is expressed as *need not*, instead of *may not*.

must

This describes a requirement on the application or user.

obsolescent

Certain features are *obsolescent*, which means that they may be considered for withdrawal in future revisions of this document. They are retained in this version because of their widespread use. Their use in new applications is discouraged. Obscolescent features are marked by OB in the left margin and the text is shaded as shown in the following example:

OB Obsolescent feature. If the whole of a synopsis is marked obsolescent, the complete interface is obsolescent.

should

With respect to implementations, the feature is recommended, but it is not mandatory. Applications should not rely on the existence of the feature.

With respect to users or applications, the word means recommended programming practice that is necessary for maximum portability.

undefined

A value or behaviour is undefined if this document imposes no portability requirements on applications for erroneous program constructs or erroneous data. Implementations may specify the result of using that value or causing that behaviour, but such specifications are not guaranteed to be consistent across all implementations. An application using such behaviour is not fully portable to all systems.

unspecified

A value or behaviour is unspecified if this document imposes no portability requirements on applications for correct program construct or correct data. Implementations may specify the result of using that value or causing that behaviour, but such specifications are not guaranteed to be consistent across all implementations. An application requiring a specific behaviour, rather than tolerating any behaviour when using that functionality, is not fully portable to all systems.

shall

This means that the behaviour described is a requirement on the implementation and applications can rely on its existence.

1.3 Relationship to Formal Standards

This specification is aligned with the IEEE standard IEEE 1295.1.

1.4 Format of Entries

The entries in Chapter 5 are based on a common format.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarises the use of the entry being described. If it is necessary to include a header to use this interface, the names of such headers are shown, for example:

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

DESCRIPTION

This section describes the functionality of the interface or header.

RETURN VALUE

This section indicates the possible return values, if any.

SEE ALSO

This section gives references to related information.

CHANGE HISTORY

This section shows the derivation of the entry and any significant changes that have been made to it.

The only sections relating to conformance are the **SYNOPSIS**, **DESCRIPTION** and **RETURN VALUE** sections.

Chapter 3 uses the same basic format with additional sections on **OPTIONS**, **APPEARANCE**, **ENVIRONMENT VARIABLES** and **FILES**, and without a **RETURN VALUE** section. All the additional sections relate to conformance.

Chapter 2

X/Open Motif Overview

This chapter gives an overview of the services in X/Open Motif.

2.1 Service Outline

The services in X/Open Motif are listed in Table 2-1:

- The left column is the name of the service. The table is organized alphabetically by this column.
- The right column is the type of the service. The type is widget, function, command or data type.

Table 2-1 Service Outline

Service Name	Service Type
ApplicationShell	widget
Composite	widget
Constraint	widget
Core	widget
MrmCloseHierarchy	function
MrmFetchBitmapLiteral	function
MrmFetchColorLiteral	function
MrmFetchIconLiteral	function
MrmFetchLiteral	function
MrmFetchSetValues	function
MrmFetchWidget	function
MrmFetchWidgetOverride	function
MrmInitialize	function
MrmOpenHierarchyPerDisplay	function
MrmRegisterClass	function
MrmRegisterNames	function
MrmRegisterNamesInHierarchy	function
Object	widget
OverrideShell	widget
RectObj	widget
Shell	widget
TopLevelShell	widget
TransientShell	widget
Uil	function
VendorShell	widget
WMShell	widget
XmActivateProtocol	function
XmActivateWMProtocol	function

Service Name	**Service Type**
XmAddProtocolCallback	function
XmAddProtocols	function
XmAddTabGroup	function
XmAddWMProtocolCallback	function
XmAddWMProtocols	function
XmArrowButton	widget
XmArrowButtonGadget	widget
XmBulletinBoard	widget
XmCascadeButton	widget
XmCascadeButtonGadget	widget
XmCascadeButtonHighlight	function
XmChangeColor	function
XmClipboardCancelCopy	function
XmClipboardCopy	function
XmClipboardCopyByName	function
XmClipboardEndCopy	function
XmClipboardEndRetrieve	function
XmClipboardInquireCount	function
XmClipboardInquireFormat	function
XmClipboardInquireLength	function
XmClipboardInquirePendingItems	function
XmClipboardLock	function
XmClipboardRegisterFormat	function
XmClipboardRetrieve	function
XmClipboardStartCopy	function
XmClipboardStartRetrieve	function
XmClipboardUndoCopy	function
XmClipboardUnlock	function
XmClipboardWithdrawFormat	function
XmCommand	widget
XmCommandAppendValue	function
XmCommandError	function
XmCommandGetChild	function
XmCommandSetValue	function
XmConvertUnits	function
XmCreateArrowButton	function
XmCreateArrowButtonGadget	function
XmCreateBulletinBoard	function
XmCreateBulletinBoardDialog	function
XmCreateCascadeButton	function
XmCreateCascadeButtonGadget	function
XmCreateCommand	function
XmCreateCommandDialog	function
XmCreateDialogShell	function
XmCreateDragIcon	function
XmCreateDrawingArea	function
XmCreateDrawnButton	function

Service Name	**Service Type**
XmCreateErrorDialog	function
XmCreateFileSelectionBox	function
XmCreateFileSelectionDialog	function
XmCreateForm	function
XmCreateFormDialog	function
XmCreateFrame	function
XmCreateInformationDialog	function
XmCreateLabel	function
XmCreateLabelGadget	function
XmCreateList	function
XmCreateMainWindow	function
XmCreateMenuBar	function
XmCreateMenuShell	function
XmCreateMessageBox	function
XmCreateMessageDialog	function
XmCreateOptionMenu	function
XmCreatePanedWindow	function
XmCreatePopupMenu	function
XmCreatePromptDialog	function
XmCreatePulldownMenu	function
XmCreatePushButton	function
XmCreatePushButtonGadget	function
XmCreateQuestionDialog	function
XmCreateRadioBox	function
XmCreateRowColumn	function
XmCreateScale	function
XmCreateScrollBar	function
XmCreateScrolledList	function
XmCreateScrolledText	function
XmCreateScrolledWindow	function
XmCreateSelectionBox	function
XmCreateSelectionDialog	function
XmCreateSeparator	function
XmCreateSeparatorGadget	function
XmCreateTemplateDialog	function
XmCreateText	function
XmCreateTextField	function
XmCreateToggleButton	function
XmCreateToggleButtonGadget	function
XmCreateWarningDialog	function
XmCreateWorkArea	function
XmCreateWorkingDialog	function
XmCvtCTToXmString	function
XmCvtXmStringToCT	function
XmDeactivateProtocol	function
XmDeactivateWMProtocol	function
XmDestroyPixmap	function

Service Name	**Service Type**
XmDialogShell	widget
XmDisplay	widget
XmDragCancel	function
XmDragContext	widget
XmDragIcon	widget
XmDragStart	function
XmDrawingArea	widget
XmDrawnButton	widget
XmDropSite	widget
XmDropSiteConfigureStackingOrder	function
XmDropSiteEndUpdate	function
XmDropSiteQueryStackingOrder	function
XmDropSiteRegister	function
XmDropSiteRegistered	function
XmDropSiteRetrieve	function
XmDropSiteStartUpdate	function
XmDropSiteUnregister	function
XmDropSiteUpdate	function
XmDropTransfer	function
XmDropTransferAdd	function
XmDropTransferStart	function
XmFileSelectionBox	widget
XmFileSelectionBoxGetChild	function
XmFileSelectionDoSearch	function
XmFontList	data type
XmFontListAppendEntry	function
XmFontListCopy	function
XmFontListEntryCreate	function
XmFontListEntryFree	function
XmFontListEntryGetFont	function
XmFontListEntryGetTag	function
XmFontListEntryLoad	function
XmFontListFree	function
XmFontListFreeFontContext	function
XmFontListInitFontContext	function
XmFontListNextEntry	function
XmFontListRemoveEntry	function
XmForm	widget
XmFrame	widget
XmGadget	widget
XmGetAtomName	function
XmGetColors	function
XmGetDestination	function
XmGetFocusWidget	function
XmGetMenuCursor	function
XmGetPixmap	function
XmGetPixmapByDepth	function

Service Name	**Service Type**
XmGetPostedFromWidget	function
XmGetTabGroup	function
XmGetVisibility	function
XmGetXmDisplay	function
XmGetXmScreen	function
XmInstallImage	function
XmInternAtom	function
XmIsMotifWMRunning	function
XmIsTraversable	function
XmLabel	widget
XmLabelGadget	widget
XmList	widget
XmListAddItem	function
XmListAddItemUnselected	function
XmListAddItems	function
XmListAddItemsUnselected	function
XmListDeleteAllItems	function
XmListDeleteItem	function
XmListDeleteItems	function
XmListDeleteItemsPos	function
XmListDeletePos	function
XmListDeletePositions	function
XmListDeselectAllItems	function
XmListDeselectItem	function
XmListDeselectPos	function
XmListGetKbdItemPos	function
XmListGetMatchPos	function
XmListGetSelectedPos	function
XmListItemExists	function
XmListItemPos	function
XmListPosSelected	function
XmListPosToBounds	function
XmListReplaceItems	function
XmListReplaceItemsPos	function
XmListReplaceItemsPosUnselected	function
XmListReplaceItemsUnselected	function
XmListReplacePositions	function
XmListSelectItem	function
XmListSelectPos	function
XmListSetAddMode	function
XmListSetBottomItem	function
XmListSetBottomPos	function
XmListSetHorizPos	function
XmListSetItem	function
XmListSetKbdItemPos	function
XmListSetPos	function
XmListUpdateSelectedList	function

Service Name	Service Type
XmListYToPos	function
XmMainWindow	widget
XmMainWindowSep1	function
XmMainWindowSep2	function
XmMainWindowSep3	function
XmMainWindowSetAreas	function
XmManager	widget
XmMenuPosition	function
XmMenuShell	widget
XmMessageBox	widget
XmMessageBoxGetChild	function
XmOptionButtonGadget	function
XmOptionLabelGadget	function
XmPanedWindow	widget
XmPrimitive	widget
XmProcessTraversal	function
XmPushButton	widget
XmPushButtonGadget	widget
XmRegisterSegmentEncoding	function
XmRemoveProtocolCallback	function
XmRemoveProtocols	function
XmRemoveTabGroup	function
XmRemoveWMProtocolCallback	function
XmRemoveWMProtocols	function
XmRepTypeAddReverse	function
XmRepTypeGetId	function
XmRepTypeGetNameList	function
XmRepTypeGetRecord	function
XmRepTypeGetRegistered	function
XmRepTypeRegister	function
XmRepTypeValidValue	function
XmResolvePartOffsets	function
XmRowColumn	widget
XmScale	widget
XmScaleGetValue	function
XmScaleSetValue	function
XmScreen	widget
XmScrollBar	widget
XmScrollBarGetValues	function
XmScrollBarSetValues	function
XmScrollVisible	function
XmScrolledWindow	widget
XmScrolledWindowSetAreas	function
XmSecondaryResourceData	function
XmSelectionBox	widget
XmSelectionBoxGetChild	function
XmSeparator	widget

Service Name	Service Type
XmSeparatorGadget	widget
XmSetMenuCursor	function
XmSetProtocolHooks	function
XmSetWMProtocolHooks	function
XmString	data type
XmStringBaseline	function
XmStringByteCompare	function
XmStringCompare	function
XmStringConcat	function
XmStringCopy	function
XmStringCreate	function
XmStringCreateLocalized	function
XmStringCreateSimple	function
XmStringDirection	data type
XmStringDraw	function
XmStringDrawImage	function
XmStringDrawUnderline	function
XmStringEmpty	function
XmStringExtent	function
XmStringFree	function
XmStringFreeContext	function
XmStringGetNextSegment	function
XmStringHasSubstring	function
XmStringHeight	function
XmStringInitContext	function
XmStringLength	function
XmStringLineCount	function
XmStringSegmentCreate	function
XmStringSeparatorCreate	function
XmStringTable	data type
XmStringWidth	function
XmText	widget
XmTextClearSelection	function
XmTextCopy	function
XmTextCut	function
XmTextDisableRedisplay	function
XmTextEnableRedisplay	function
XmTextField	widget
XmTextFieldClearSelection	function
XmTextFieldCopy	function
XmTextFieldCut	function
XmTextFieldGetBaseline	function
XmTextFieldGetEditable	function
XmTextFieldGetInsertionPosition	function
XmTextFieldGetLastPosition	function
XmTextFieldGetMaxLength	function
XmTextFieldGetSelection	function

Service Name	**Service Type**
XmTextFieldGetSelectionPosition	function
XmTextFieldGetString	function
XmTextFieldGetSubstring	function
XmTextFieldInsert	function
XmTextFieldPaste	function
XmTextFieldPosToXY	function
XmTextFieldRemove	function
XmTextFieldReplace	function
XmTextFieldSetAddMode	function
XmTextFieldSetEditable	function
XmTextFieldSetHighlight	function
XmTextFieldSetInsertionPosition	function
XmTextFieldSetMaxLength	function
XmTextFieldSetSelection	function
XmTextFieldSetString	function
XmTextFieldShowPosition	function
XmTextFieldXYToPos	function
XmTextFindString	function
XmTextGetBaseline	function
XmTextGetEditable	function
XmTextGetInsertionPosition	function
XmTextGetLastPosition	function
XmTextGetMaxLength	function
XmTextGetSelection	function
XmTextGetSelectionPosition	function
XmTextGetSource	function
XmTextGetString	function
XmTextGetSubstring	function
XmTextGetTopCharacter	function
XmTextInsert	function
XmTextPaste	function
XmTextPosToXY	function
XmTextPosition	data type
XmTextRemove	function
XmTextReplace	function
XmTextScroll	function
XmTextSetAddMode	function
XmTextSetEditable	function
XmTextSetHighlight	function
XmTextSetInsertionPosition	function
XmTextSetMaxLength	function
XmTextSetSelection	function
XmTextSetSource	function
XmTextSetString	function
XmTextSetTopCharacter	function
XmTextShowPosition	function
XmTextXYToPos	function

Service Name	Service Type
XmToggleButton	widget
XmToggleButtonGadget	widget
XmToggleButtonGadgetGetState	function
XmToggleButtonGadgetSetState	function
XmToggleButtonGetState	function
XmToggleButtonSetState	function
XmTrackingEvent	function
XmTrackingLocate	function
XmUninstallImage	function
XmUpdateDisplay	function
XmWidgetGetBaselines	function
XmWidgetGetDisplayRect	function
mwm	command

2.2 Overview of Services by Type and Function

All the services are broken into four types in the tables below.

- window manager (see Table 2-2)
- widgets and widget functions (see Table 2-3)
- toolkit functions (see Table 2-4 on page 20)
- user interface language (see Table 2-5 on page 22).

Within each table, components are organized by function.

2.2.1 Window Manager

Table 2-2 Window Manager Services

Service Name	**Service Type**
mwm	command

2.2.2 Widgets and Widget Functions

The following table organizes widgets by hierarchy. Position in the hierarchy is shown by the indentation of the service name. The functions for each widget immediately follow the widget.

Table 2-3 Widget Services

Service Name	**Service Type**
XmDropSite	widget
XmDropSiteConfigureStackingOrder	function
XmDropSiteEndUpdate	function
XmDropSiteQueryStackingOrder	function
XmDropSiteRegister	function
XmDropSiteRegistered	function
XmDropSiteRetrieve	function
XmDropSiteStartUpdate	function
XmDropSiteUnregister	function
XmDropSiteUpdate	function
Object	widget
XmDropSiteTransfer	widget
XmDropSiteTransferAdd	function
XmDropSiteTransferStart	function
RectObj	widget
Core	widget
DragIcon	widget
XmCreateDragIcon	function
Composite	widget
DragContext	widget

Service Name	**Service Type**
XmDragCancel	function
XmDragStart	function
Constraint	widget
XmManager	widget
XmBulletinBoard	widget
XmCreateBulletinBoard	function
XmCreateBulletinBoardDialog	function
XmForm	widget
XmCreateForm	function
XmCreateFormDialog	function
XmMessageBox	widget
XmCreateMessageBox	function
XmCreateErrorDialog	function
XmCreateInformationDialog	function
XmCreateMessageDialog	function
XmCreateQuestionDialog	function
XmCreateTemplateDialog	function
XmCreateWarningDialog	function
XmCreateWorkingDialog	function
XmMessageBoxGetChild	function
XmSelectionBox	widget
XmCreateSelectionBox	function
XmCreatePromptDialog	function
XmCreateSelectionDialog	function
XmSelectionBoxGetChild	function
XmCommand	widget
XmCreateCommand	function
XmCreateCommandDialog	function
XmCommandAppendValue	function
XmCommandError	function
XmCommandGetChild	function
XmCommandSetValue	function
XmFileSelectionBox	widget
XmCreateFileSelectionBox	function
XmCreateFileSelectionDialog	function
XmFileSelectionBoxGetChild	function
XmFileSelectionDoSearch	function
XmDrawingArea	widget
XmCreateDrawingArea	function
XmFrame	widget
XmCreateFrame	function
XmPanedWindow	widget
XmCreatePanedWindow	function
XmRowColumn	widget
XmCreateRowColumn	function
XmCreateMenuBar	function
XmCreateOptionMenu	function

Service Name	**Service Type**
XmCreatePopupMenu	function
XmCreatePulldownMenu	function
XmCreateRadioBox	function
XmCreateWorkArea	function
XmGetMenuCursor	function
XmGetPostedFromWidget	function
XmGetTearOffControl	function
XmMenuPosition	function
XmOptionButtonGadget	function
XmOptionLabelGadget	function
XmSetMenuCursor	function
XmScale	widget
XmCreateScale	function
XmScaleGetValue	function
XmScaleSetValue	function
XmScrolledWindow	widget
XmCreateScrolledWindow	function
XmCreateScrolledList	function
XmCreateScrolledText	function
XmScrollVisible	function
XmScrolledWindowSetAreas	function
XmMainWindow	widget
XmCreateMainWindow	function
XmMainWindowSep1	function
XmMainWindowSep2	function
XmMainWindowSep3	function
XmMainWindowSetAreas	function
Shell	widget
OverrideShell	widget
XmMenuShell	widget
XmCreateMenuShell	function
WMShell	widget
VendorShell	widget
XmGetAtomName	function
XmInternAtom	function
TopLevelShell	widget
ApplicationShell	widget
XmDisplay	widget
XmGetXmDisplay	function
TransientShell	widget
XmDialogShell	widget
XmCreateDialogShell	function
XmPrimitive	widget
XmArrowButton	widget
XmCreateArrowButton	function
XmLabel	widget
XmCreateLabel	function

Service Name	**Service Type**
XmCascadeButton	widget
XmCreateCascadeButton	function
XmCascadeButtonHighlight	function
XmDrawnButton	widget
XmCreateDrawnButton	function
XmPushButton	widget
XmCreatePushButton	function
XmToggleButton	widget
XmCreateToggleButton	function
XmToggleButtonGetState	function
XmToggleButtonSetState	function
XmList	widget
XmCreateList	function
XmListAddItem	function
XmListAddItemUnselected	function
XmListAddItems	function
XmListAddItemsUnselected	function
XmListDeleteAllItems	function
XmListDeleteItem	function
XmListDeleteItems	function
XmListDeleteItemsPos	function
XmListDeletePos	function
XmListDeletePositions	function
XmListDeselectAllItems	function
XmListDeselectItem	function
XmListDeselectPos	function
XmListGetKbdItemPos	function
XmListGetMatchPos	function
XmListGetSelectedPos	function
XmListItemExists	function
XmListItemPos	function
XmListPosSelected	function
XmListPosToBounds	function
XmListReplaceItems	function
XmListReplaceItemsPos	function
XmListReplaceItemsPosUnselected	function
XmListReplaceItemsUnselected	function
XmListReplacePositions	function
XmListSelectItem	function
XmListSelectPos	function
XmListSetAddMode	function
XmListSetBottomItem	function
XmListSetBottomPos	function
XmListSetHorizPos	function
XmListSetItem	function
XmListSetKbdItemPos	function
XmListSetPos	function

Service Name	Service Type
XmListUpdateSelectedList	function
XmListYToPos	function
XmScrollBar	widget
XmCreateScrollBar	function
XmScrollBarGetValues	function
XmScrollBarSetValues	function
XmSeparator	widget
XmCreateSeparator	function
XmText	widget
XmCreateText	function
XmTextClearSelection	function
XmTextCopy	function
XmTextCut	function
XmTextDisableRedisplay	function
XmTextEnableRedisplay	function
XmTextFindString	function
XmTextGetBaseline	function
XmTextGetEditable	function
XmTextGetInsertionPosition	function
XmTextGetLastPosition	function
XmTextGetMaxLength	function
XmTextGetSelection	function
XmTextGetSelectionPosition	function
XmTextGetSource	function
XmTextGetString	function
XmTextGetSubstring	function
XmTextGetTopCharacter	function
XmTextInsert	function
XmTextPaste	function
XmTextPosToXY	function
XmTextRemove	function
XmTextReplace	function
XmTextScroll	function
XmTextSetAddMode	function
XmTextSetEditable	function
XmTextSetHighlight	function
XmTextSetInsertionPosition	function
XmTextSetMaxLength	function
XmTextSetSelection	function
XmTextSetSource	function
XmTextSetString	function
XmTextSetTopCharacter	function
XmTextShowPosition	function
XmTextXYToPos	function
XmTextField	widget
XmCreateTextField	function
XmTextFieldClearSelection	function

Service Name	**Service Type**
XmTextFieldCopy	function
XmTextFieldCut	function
XmTextFieldGetBaseline	function
XmTextFieldGetEditable	function
XmTextFieldGetInsertionPosition	function
XmTextFieldGetLastPosition	function
XmTextFieldGetMaxLength	function
XmTextFieldGetSelection	function
XmTextFieldGetSelectionPosition	function
XmTextFieldGetString	function
XmTextFieldGetSubstring	function
XmTextFieldInsert	function
XmTextFieldPaste	function
XmTextFieldPosToXY	function
XmTextFieldRemove	function
XmTextFieldReplace	function
XmTextFieldSetAddMode	function
XmTextFieldSetEditable	function
XmTextFieldSetHighlight	function
XmTextFieldSetInsertionPosition	function
XmTextFieldSetMaxLength	function
XmTextFieldSetSelection	function
XmTextFieldSetString	function
XmTextFieldShowPosition	function
XmTextFieldXYToPos	function
XmScreen	widget
XmGetXmScreen	function
XmGadget	widget
XmArrowButtonGadget	widget
XmCreateArrowButtonGadget	function
XmLabelGadget	widget
XmCreateLabelGadget	function
XmCascadeButtonGadget	widget
XmCreateCascadeButtonGadget	function
XmPushButtonGadget	widget
XmCreatePushButtonGadget	function
XmToggleButtonGadget	widget
XmCreateToggleButtonGadget	function
XmToggleButtonGadgetGetState	function
XmToggleButtonGadgetSetState	function
XmSeparatorGadget	widget
XmCreateSeparatorGadget	function

2.2.3 Toolkit Functions and Data Types

Table 2-4 Toolkit Services

Service Name	Service Type
XmActivateProtocol	function
XmActivateWMProtocol	function
XmAddProtocolCallback	function
XmAddProtocols	function
XmAddTabGroup	function
XmAddWMProtocolCallback	function
XmAddWMProtocols	function
XmChangeColor	function
XmClipboardCancelCopy	function
XmClipboardCopy	function
XmClipboardCopyByName	function
XmClipboardEndCopy	function
XmClipboardEndRetrieve	function
XmClipboardInquireCount	function
XmClipboardInquireFormat	function
XmClipboardInquireLength	function
XmClipboardInquirePendingItems	function
XmClipboardLock	function
XmClipboardRegisterFormat	function
XmClipboardRetrieve	function
XmClipboardStartCopy	function
XmClipboardStartRetrieve	function
XmClipboardUndoCopy	function
XmClipboardUnlock	function
XmClipboardWithdrawFormat	function
XmConvertUnits	function
XmCvtCTToXmString	function
XmCvtXmStringToCT	function
XmDeactivateProtocol	function
XmDeactivateWMProtocol	function
XmDestroyPixmap	function
XmFontList	data type
XmFontListAppendEntry	function
XmFontListCopy	function
XmFontListEntryCreate	function
XmFontListEntryFree	function
XmFontListEntryGetFont	function
XmFontListEntryGetTag	function
XmFontListEntryLoad	function
XmFontListFree	function
XmFontListFreeFontContext	function
XmFontListInitFontContext	function
XmFontListNextEntry	function

Service Name	Service Type
XmFontListRemoveEntry	function
XmGetColors	function
XmGetDestination	function
XmGetFocusWidget	function
XmGetPixmap	function
XmGetPixmapByDepth	function
XmGetTabGroup	function
XmGetVisibility	function
XmInstallImage	function
XmIsMotifWMRunning	function
XmIsTraversable	function
XmMapSegmentEncoding	function
XmProcessTraversal	function
XmRegisterSegmentEncoding	function
XmRemoveProtocolCallback	function
XmRemoveProtocols	function
XmRemoveTabGroup	function
XmRemoveWMProtocolCallback	function
XmRemoveWMProtocols	function
XmResolvePartOffsets	function
XmSecondaryResourceData	function
XmSetProtocolHooks	function
XmSetWMProtocolHooks	function
XmString	data type
XmStringBaseline	function
XmStringByteCompare	function
XmStringCompare	function
XmStringConcat	function
XmStringCopy	function
XmStringCreate	function
XmStringCreateLocalized	function
XmStringCreateSimple	function
XmStringDirection	data type
XmStringDraw	function
XmStringDrawImage	function
XmStringDrawUnderline	function
XmStringEmpty	function
XmStringExtent	function
XmStringFree	function
XmStringFreeContext	function
XmStringGetNextSegment	function
XmStringHasSubstring	function
XmStringHeight	function
XmStringInitContext	function
XmStringLength	function
XmStringLineCount	function
XmStringSegmentCreate	function

Service Name	Service Type
XmStringSeparatorCreate	function
XmStringTable	data type
XmStringWidth	function
XmTextPosition	data type
XmTrackingEvent	function
XmTrackingLocate	function
XmUninstallImage	function
XmUpdateDisplay	function
XmWidgetGetBaselines	function
XmWidgetGetDisplayRect	function

2.2.4 User Interface Language

Table 2-5 User Interface Language Services

Service Name	Service Type
MrmCloseHierarchy	function
MrmFetchBitmapLiteral	function
MrmFetchColorLiteral	function
MrmFetchIconLiteral	function
MrmFetchLiteral	function
MrmFetchSetValues	function
MrmFetchWidget	function
MrmFetchWidgetOverride	function
MrmInitialize	function
MrmOpenHierarchyPerDisplay	function
MrmRegisterClass	function
MrmRegisterNames	function
MrmRegisterNamesInHierarchy	function
Uil	function

2.3 X/Open Motif Name Space

This specification reserves symbolic names prefixed with **Xm** for symbols available to the application and **_Xm** for symbols used internally by an implementation. The name space **Xme** is reserved for future use. The header files listed in Section 2.3.1 define one or more of these symbols. Any application that uses one of these header files must respect the reserved name space.

2.3.1 Header Files

The following header files and any files on which they depend are required for an X/Open Motif implementation:

<ArrowB.h> **<ArrowBG.h>** **<AtomMgr.h>** **<BulletinB.h>**
<CascadeB.h> **<CascadeBG.h>** **<Command.h>** **<CutPaste.h>**
<DialogS.h> **<Display.h>** **<DragDrop.h>** **<DragIcon.h>**
<DragOverS.h> **<DrawingA.h>** **<DrawnB.h>** **<DropSMgr.h>**
<DropTrans.h> **<FileSB.h>** **<Form.h>** **<Frame.h>**
<Label.h> **<LabelG.h>** **<List.h>** **<MainW.h>**
<MenuShell.h> **<MessageB.h>** **<MrmPublic.h>** **<MwmUtil.h>**
<PanedW.h> **<Protocols.h>** **<PushB.h>** **<PushBG.h>**
<RepType.h> **<RowColumn.h>** **<Scale.h>** **<Screen.h>**
<ScrollBar.h> **<ScrolledW.h>** **<SelectioB.h>** **<SeparatoG.h>**
<Separator.h> **<Text.h>** **<TextF.h>** **<ToggleB.h>**
<ToggleBG.h> **<VendorS.h>** **<VirtKeys.h>** **<Xm.h>**

These header files declare all function prototypes, constant values, symbols, data structures, class pointers and type definitions required for an X/Open Motif implementation. The individual reference pages for each function specify the header files that must be included for that function to compile and link correctly. The individual reference pages for each widget specify the name of the class pointer variable and its own resource class.

2.3.2 Resource Name Format

Widget resources are specified by a name, of the form **XmN***resourceName*. Every **XmN***resourceName* defined in X/Open Motif shall be declared in a header file. The declaration must be such that each symbol **XmN***resourceName* resolves to the string *resourceName* through the compile and link process.

Each widget resource is associated with a resource class. The name of the resource class is usually, but not always, **XmC***ResourceName*. Any **XmC***ResourceName* symbol shall be declared in the header files such that it resolves to the string *ResourceName*.

There are a number of exceptions to the rule that the class of **XmN***resourceName* is **XmC***ResourceName*. These are listed in the following table:

Resource Name	**Resource Class Name**
XmNallowResize	**XmCBoolean**
XmNbottomAttachment	**XmCAttachment**
XmNbottomOffset	**XmCOffset**
XmNbottomPosition	**XmCPosition**

Resource Name	Resource Class Name
XmNbottomWidget	XmCWidget
XmNcancelButton	XmCWidget
XmNcascadePixmap	XmCPixmap
XmNcommand	XmCTextString
XmNdefaultButton	XmCWidget
XmNentryAlignment	XmCAlignment
XmNentryVerticalAlignment	XmCVerticalAlignment
XmNfractionBase	XmCMaxValue
XmNhistoryItemCount	XmCItemCount
XmNhistoryItems	XmCItems
XmNhistoryMaxItems	XmCMaxItems
XmNhistoryVisibleItemCount	XmCVisibleItemCount
XmNhorizontalSpacing	XmCSpacing
XmNleftAttachment	XmCAttachment
XmNleftOffset	XmCOffset
XmNleftPosition	XmCPosition
XmNleftWidget	XmCWidget
XmNlistItemCount	XmCItemCount
XmNlistItems	XmCItems
XmNlistVisibleItemCount	XmCVisibleItemCount
XmNmenuAccelerator	XmCAccelerators
XmNmenuCursor	XmCCursor
XmNmenuHelpWidget	XmCMenuWidget
XmNmenuHistory	XmCMenuWidget
XmNmessageAlignment	XmCAlignment
XmNoffsetX	XmCOffset
XmNoffsetY	XmCOffset
XmNrefigureMode	XmCBoolean
XmNresizable	XmCBoolean
XmNrightAttachment	XmCAttachment
XmNrightOffset	XmCOffset
XmNrightPosition	XmCPosition
XmNrightWidget	XmCWidget
XmNsashShadowThickness	XmCShadowThickness
XmNscrollHorizontal	XmCScroll
XmNscrollLeftSide	XmCScrollSide
XmNscrollTopSide	XmCScrollSide
XmNscrollVertical	XmCScroll
XmNskipAdjust	XmCBoolean
XmNsubMenuId	XmCMenuWidget
XmNsymbolPixmap	XmCPixmap
XmNtextColumns	XmCColumns
XmNtextTranslations	XmCTranslations
XmNtopAttachment	XmCAttachment
XmNtopOffset	XmCOffset
XmNtopPosition	XmCPosition
XmNtopWidget	XmCWidget

Resource Name	**Resource Class Name**
XmNverticalSpacing	**XmCSpacing**

Chapter 3

X/Open Motif Commands

This chapter contains the reference information for X/Open Motif commands.

NAME

mwm — the Motif Window Manager

SYNOPSIS

```
mwm [options]
```

DESCRIPTION

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

OPTIONS

–**display** *display*

This option specifies the display to use; see the X server manual page in the X Window System User's Guide.

–**xrm** *resourcestring*

This option specifies a resource string to use.

–**multiscreen**

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

–**name** *name*

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

–**screens** *name* [*name* [...]]

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

APPEARANCE

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

Screens

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

When *mwm* is managing multiple screens, the –**screens** option can be used to give each screen a unique resource name. The names are separated by blanks, for example:

```
-screens mwm0 mwm1
```

If there are more screens than names, resources for the remaining screens are retrieved using the first name.

Windows

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

Title Area

In addition to displaying the client's title, the title area is used to move the window. To move the window, place the pointer over the title area, pressing button 1 and dragging the window to a new location. 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.

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.

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

Default Window Menu

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

Resize Border Handles

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

Matte

An optional matte decoration can be added between the client area and the window frame.

A matte is not actually part of the window frame. There is no functionality associated with a matte.

Icons

Icons are small graphic representations of windows. A window can be minimized (iconified) with 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 (pressing and releasing a button without moving the mouse equals a click) causes the menu to stay posted. The menu contains the selections described in the following table.

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

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

Double-clicking button 1 on an icon invokes the *f.restore* function and restores the icon's associated window to its previous state. Double-clicking button 1 on the icon box's icon opens the icon box and allows access to the contained icons.

Icon Box

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

Icons in the icon box can be manipulated with the mouse. The following table summarizes the behavior of this interface. Button actions apply whenever the pointer is on any part of the icon. Note that invoking the *f.raise* function on an icon in the icon box raises an already open window to the top of the stack.

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

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

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

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

Input Focus

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

The following tables summarize the keyboard input focus selection behavior.

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

Key Action	Function Description
<Alt> <Tab>	Move input focus to next window in window stack
<Alt> <Shift> <Tab>	Move input focus to previous window in window stack

Window Stacking

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

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

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

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

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

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

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

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

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

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

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

X Defaults

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

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

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

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

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

mwm uses the following types of resources:

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

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

Client-specific Resources
> These *mwm* resources can be set for a particular client window or class of client window. They specify client-specific icon and client window frame appearance and behavior.
>
> Resource identifiers can be either a resource name (for example, **foreground**) or a resource class (for example, **Foreground**). If the value of a resource is a filename and if the filename is prefixed by ˜/ (tilde, slash), then it is relative to the path contained in the *HOME* environment variable (generally the user's home directory).

Component Appearance Resources

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

```
Mwm*resource_id
```

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

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

```
Mwm*[menu|icon|client|feedback]*resource_id
```

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

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

```
Mwm*client*title*resource_id
```

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

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

```
Mwm*menu*menu_name*resource_id
```

For example, **Mwm*menu*my_menu*foreground** specifies the foreground color for the menu named **my_menu**.

The 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	**BottomShadowPixmap**	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*( ).

‡ X11 R5 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 resource 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* sets the save under attribute on the window manager frame of any client that has it set. If **saveUnder** is False, save unders not 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*( ).

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

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

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

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

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

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

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

General Appearance and Behavior Resources

The syntax for specifying general appearance and behavior resources is:

```
Mwm*resource_id
```

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

The following general appearance and behavior resources can be specified.

General Appearance and Behavior Resources			
Name	**Class**	**Value Type**	**Default**
autoKeyFocus	**AutoKeyFocus**	T/F	T
autoRaiseDelay	**AutoRaiseDelay**	millisec	500
bitmapDirectory	**BitmapDirectory**	directory	**/usr/include/**\ **X11/bitmaps**
buttonBindings	**ButtonBindings**	string	DefaultBut\ tonBindings
cleanText	**CleanText**	T/F	T
clientAutoPlace	**ClientAutoPlace**	T/F	T
colormapFocusPolicy	**ColormapFocusPolicy**	string	keyboard
configFile	**ConfigFile**	file	**.mwmrc**
deiconifyKeyFocus	**DeiconifyKeyFocus**	T/F	T
doubleClickTime	**DoubleClickTime**	millisec.	multiclick time
enableWarp	**enableWarp**	T/F	T
enforceKeyFocus	**EnforceKeyFocus**	T/F	T
fadeNormalIcon	**FadeNormalIcon**	T/F	F
frameBorderWidth	**FrameBorderWidth**	pixels	5
iconAutoPlace	**IconAutoPlace**	T/F	T
iconBoxGeometry	**IconBoxGeometry**	string	6x1+0-0
iconBoxName	**IconBoxName**	string	iconbox
iconBoxSBDisplayPolicy	**IconBoxSBDisplayPolicy**	string	all
iconBoxTitle	**IconBoxTitle**	**XmString**	Icons
iconClick	**IconClick**	T/F	T
iconDecoration	**IconDecoration**	string	varies
iconImageMaximum	**IconImageMaximum**	wxh	50x50
iconImageMinimum	**IconImageMinimum**	wxh	16x16
iconPlacement	**IconPlacement**	string	left bottom
iconPlacementMargin	**IconPlacementMargin**	pixels	varies
interactivePlacement	**InteractivePlacement**	T/F	F
keyBindings	**KeyBindings**	string	DefaultKey\ Bindings
keyboardFocusPolicy	**KeyboardFocusPolicy**	string	explicit
limitResize	**LimitResize**	T/F	T
lowerOnIconify	**LowerOnIconify**	T/F	T
maximumMaximumSize	**MaximumMaximumSize**	wxh (pixels)	2X screen w and h

General Appearance and Behavior Resources			
Name	**Class**	**Value Type**	**Default**
moveThreshold	**MoveThreshold**	pixels	4
multiScreen	**MultiScreen**	T/F	F
passButtons	**PassButtons**	T/F	F
passSelectButton	**PassSelectButton**	T/F	T
positionIsFrame	**PositionIsFrame**	T/F	T
positionOnScreen	**PositionOnScreen**	T/F	T
quitTimeout	**QuitTimeout**	millisec.	1000
raiseKeyFocus	**RaiseKeyFocus**	T/F	F
resizeBorderWidth	**ResizeBorderWidth**	pixels	10
resizeCursors	**ResizeCursors**	T/F	T
screens	**Screens**	string	varies
showFeedback	**ShowFeedback**	string	all
startupKeyFocus	**StartupKeyFocus**	T/F	T
transientDecoration	**TransientDecoration**	string	menu title
transientFunctions	**TransientFunctions**	string	−minimize −maximize
useIconBox	**UseIconBox**	T/F	F
wMenuButtonClick	**WMenuButtonClick**	T/F	T
wMenuButtonClick2	**WMenuButtonClick2**	T/F	T

autoKeyFocus (class **AutoKeyFocus**)

This resource is available only when the keyboard input focus policy is explicit. If **autoKeyFocus** is given a value of True, then when a window with the keyboard input focus is withdrawn from window management or is iconified, the focus is set to the previous window that had the focus. If the value given is False, there is no automatic setting of the keyboard input focus. 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* waits before raising a window after it gets the keyboard focus. The default value of this resource is 500 (ms).

bitmapDirectory (class **BitmapDirectory**)

This resource identifies a directory to be searched for bitmaps referenced by *mwm* resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is **/usr/include/X11/bitmaps**. The directory **/usr/include/X11/bitmaps** represents the customary locations for this directory. The actual location of this directory may vary on some systems.

buttonBindings (class **ButtonBindings**)

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

cleanText (class **CleanText**)

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

clientAutoPlace (class **ClientAutoPlace**)

This resource determines the position of a window when the window has not been given a 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* attempts to place the windows totally on-screen. The default value is True.

colormapFocusPolicy (class **ColormapFocusPolicy**)

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

configFile (class **ConfigFile**)

The resource value is the pathname for an *mwm* resource description file.

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

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

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

deiconifyKeyFocus (class **DeiconifyKeyFocus**)

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

doubleClickTime (class **DoubleClickTime**)

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

enableWarp (class **EnableWarp**)

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

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

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

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 5 pixels.

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

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

```
[=][widthxheight][{+-}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 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 used in the title area of the icon box frame. The default value is **Icons**.

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

iconDecoration (class **IconDecoration**)
This resource specifies the general icon decoration. The resource value is label (only the

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

iconImageMaximum (class **IconImageMaximum**)
This resource specifies the maximum size of the icon *image*. The resource value is *widthxheight* (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 *widthxheight* (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
```

The layout values are described in the following table.

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

A horizontal (vertical) layout value should not be used for both the *primary_layout* and the *secondary_layout* (for example, do not use top for the *primary_layout* and bottom for the *secondary_layout*). The *primary_layout* indicates whether, when an icon placement is performed, 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).

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 (zero). A default value is used if the value specified is invalid. The default value for this resource is equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column).

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

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

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

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

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

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

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 or 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 this resource is False, *mwm* manages only a single screen. The default value is False.

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

passSelectButton (class **PassSelectButton**)
This resource indicates whether or not to pass the select button press events to clients after they are used to do a window manager function in the client context. If the resource value is False, then the button press 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 performed 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* waits for a client to update the WM_COMMAND property after *mwm* has sent the WM_SAVE_YOURSELF message. The default value of this resource is 1000 (ms). (Refer to the *f.kill* function description for additional information.)

raiseKeyFocus (class **RaiseKeyFocus**)

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

resizeBorderWidth (class **ResizeBorderWidth**)

This resource specifies the width (in pixels) of a client window frame border with resize handles. The specified border width includes the 3-D shadows. The default is 10 (pixels).

resizeCursors (class **ResizeCursors**)

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

screens (class **Screens**)

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

showFeedback (class **ShowFeedback**)

This resource controls when feedback information is displayed. It controls both window position and size feedback during move or resize operations and initial client placement. It also controls window manager message and dialog boxes.

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

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

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

The following sample command line illustrates the syntax for **showFeedback**:

```
Mwm*showFeedback: placement resize behavior restart
```

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

startupKeyFocus (class **StartupKeyFocus**)

This resource is available only when the keyboard input focus policy is explicit. When given the default value of True, a window gets the keyboard input focus when the window is mapped (that is, initially managed by the window manager).

transientDecoration (class **TransientDecoration**)

This resource 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* 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 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* reference page.

useIconBox (class **UseIconBox**)

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

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

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

Client-specific Resources

The syntax for specifying client-specific resources is:

```
Mwm*client_name_or_class*resource_id
```

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

```
Mwm*resource_id
```

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

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

```
Mwm*defaults*resource_id
```

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

The client-specific resources in the following table can be specified.

Client-specific Resources

Name	Class	Value Type	Default
clientDecoration	**ClientDecoration**	string	all
clientFunctions	**ClientFunctions**	string	all
focusAutoRaise	**FocusAutoRaise**	T/F	varies
iconImage	**IconImage**	pathname	(image)
iconImageBackground	**Background**	color	icon background
iconImageBottomShadowColor	**Foreground**	color	icon bottom shadow
iconImageBottomShadowPixmap	**BottomShadow-Pixmap**	pixmap	icon bottom shadow pixmap
iconImageForeground	**Foreground**	color	varies
iconImageTopShadowColor	**Background**	color	icon top shadow color
iconImageTopShadowPixmap	**TopShadow-Pixmap**	pixmap	icon top shadow pixmap

Client-specific Resources			
Name	**Class**	**Value Type**	**Default**
matteBackground	**Background**	color	background
matteBottomShadowColor	**Foreground**	color	bottom shadow color
matteBottomShadowPixmap	**BottomShadow-Pixmap**	pixmap	bottom shadow pixmap
matteForeground	**Foreground**	color	foreground
matteTopShadowColor	**Background**	color	top shadow color
matteTopShadowPixmap	**TopShadow-Pixmap**	pixmap	top shadow pixmap
matteWidth	**MatteWidth**	pixels	0
maximumClientSize	**MaximumClientSize**	wxh	fill the screen
useClientIcon	**UseClientIcon**	T/F	F
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* 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)

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 a function is preceded by a minus sign, that function is not applicable to the client window; otherwise, the function is applicable to the client window. The sign of the first item in the list determines the initial set of functions. If the sign of the first function is minus, *mwm* assumes all functions are applicable and starts subtracting from that set. If the sign of the first function is plus (or not specified), *mwm* starts with no functions and builds up a list from the resource. The items in the list are delimited by spaces.

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* reference page.

The following table lists the functions available for this resource.

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

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

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

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

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

iconImageForeground (class **Foreground**)
This resource specifies the foreground color of the icon image 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 displayed in the image part of an icon. The default value of this resource is the icon top shadow color (which is specified by **Mwm*icon*topShadowColor**).

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

matteBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow pixmap of the matte, when **matteWidth** is positive. The default value is the client bottom shadow pixmap (which 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 (which 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 (which 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 (which 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 (zero), which effectively disables the matte.

maximumClientSize (class **MaximumClientSize**)
This is a size specification that indicates the client size to be used when an application is maximized. The resource value is specified as *widthxheight*. The width and height are interpreted in the units the client uses (for example, for terminal emulators this is generally characters). If this resource is not specified, the maximum size from the WM_NORMAL_HINTS property is used if set. Otherwise the default value is the size where the client window with window management borders fills the screen. When the maximum client size is not determined by the **maximumClientSize** resource, the **maximumMaximumSize** resource value is used as a constraint on the maximum size.

useClientIcon (class **UseClientIcon**)

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

windowMenu (class **WindowMenu**)

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

Resource Description File

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

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

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

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

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

mwm Resource Description File Syntax

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

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

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

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

f.beep

This function causes a beep.

f.circle_down (icon | window)

This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it no longer obscures any other window or icon).

This function affects only those windows and icons that obscure other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an icon function argument is specified, the function applies only to icons. If a window function argument is specified, the function applies only to windows.

f.circle_up (icon | window)

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

f.exec or **!**

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

f.focus_color

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

f.focus_key

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

f.kill

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

f.lower (–*client* | within | freeFamily)

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

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

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

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

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

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

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

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

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

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

f.nop
This function does nothing.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Window Manager Event Specification

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

Button events have the following syntax:

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

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

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

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

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

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

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

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

Button Bindings

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

The button binding syntax is:

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

The syntax for the *context* specification is:

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

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

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

Key Bindings

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

The key binding syntax is:

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

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

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

Menu Panes

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

The menu pane specification syntax is:

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

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

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

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

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

A **mnemonic** specification has the following syntax:

```
mnemonic = _character
```

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

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

ENVIRONMENT VARIABLES

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

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

mwm uses the environment variables *XFILESEARCHPATH*, *XUSERFILESEARCHPATH*, *XAPPLRESDIR*, *XENVIRONMENT*, *LANG*, and *HOME* in determining search paths for resource defaults files.

mwm reads the **$HOME/.motifbind** file if it exists to install a virtual key bindings property on the root window.

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

Target for Validating mwm

The target **MWM_FRAME_ICON_INFO** is provided for automatic testing support. It allows a client to obtain *mwm* parameters such as window and icon sizes and positions, window menu contents, and window manager decorations.

FILES

$HOME/Mwm
$HOME/.Xdefaults
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
$HOME/.motifbind

SEE ALSO

VendorShell, *XmInstallImage*( ), and X server manual page in the X Window System User's Guide.

Chapter 4

Data Types

This chapter defines the data types used by Motif functions.

4.1 X11 Data Types

The following data types are defined in X11 header files (see the **X Toolkit Intrinsics** specification and the **Xlib** specification):

ArgList
Atom
Boolean
Cardinal
Colormap
Cursor
Dimension
Display
Pixel
Pixmap
Position
String
Time
Widget
WidgetClass
XButtonPressedEvent
XEvent
XtCallbackList
XtPointer

The X11 header files may be included in **<Xm/Xm.h>**.

4.2 XmFontList

XmFontList is the data type for a font list. It is defined in the header file **<Xm/Xm.h>**.

A font list consists of font list entries. Each entry contains a font or a font set (a group of fonts) and is identified with a tag, which is optional. If this tag is NULL, the tag is set to XmFONTLIST_DEFAULT_TAG.

When a compound string is displayed, the font list element tag of the compound string segment is matched with a font list entry tag in the font list and the matching font list entry is used to display the compound string. A font list entry is chosen as follows:

- The first font list entry whose tag matches the tag of the compound string segment is used.
- If no match has been found, the first entry in the font list is used.

The font list interface consists of the following routines:

XmFontListAppendEntry( )
XmFontListCopy( )
XmFontListEntryCreate( )
XmFontListEntryFree( )
XmFontListEntryGetFont( )
XmFontListEntryGetTag( )
XmFontListEntryLoad( )
XmFontListFree( )
XmFontListFreeFontContext( )
XmFontListInitFontContext( )
XmFontListNextEntry( )
XmFontListRemoveEntry( ).

Font lists are specified in resource files with the following syntax:

```
resource_spec: font_entry [ , font_entry ]+
```

The resource value string consists of one or more font list entries separated by commas. Each *font_entry* identifies a font or font set and an optional font list entry tag. A tag specified for a single font follows the font name and is separated by = (equals sign); otherwise, in a font set the tag is separated by a colon. The colon is required whether a tag is specified or not. A font entry uses the following syntax to specify a single font:

```
font_name [ '=' tag ]
```

For example, the following entry specifies a 10 point Times Italic font without a font list entry tag:

```
*fontList:  -Adobe-Times-Medium-I-Normal--10*
```

A font entry containing a font set is similar, except a semicolon separates multiple font names and the specification ends with a colon followed by an optional tag:

```
font_name [ ';' font_name ]+ ':' [ tag ]
```

A *font_name* is an X Logical Font Description (XLFD) string and *tag* is any set of characters from the ISO 646 IRV standard except space, comma, colon, equal sign and semicolon. Following is an example of a font set entry. It consists of three fonts (except for charsets) and an explicit font list entry tag.

```
*fontList : -Adobe-Courier-Bold-R-Normal--25-180-100-100-M-150;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-240;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-120:MY_TAG
```

4.3 XmString

XmString is the data type for a compound string. It is defined in the header files **<Xm/Command.h>**, **<Xm/FileSB.h>**, **<Xm/List.h>** and **<Xm/Xm.h>**. Compound strings include one or more segments, each of which may contain a font list element tag, string direction and text component. When a compound string is displayed, the font list element tag and the direction are used to determine how to display the text.

Calling *XtGetValues*( ) for a resource whose type is **XmString** yields a copy of the compound string resource value. The application is responsible for using *XmStringFree*( ) to free the memory allocated for the copy.

Refer to Section 4.2 on page 60 for a description of the algorithm that associates the font list element tag of a compound string segment with a font list entry in a font list.

The compound string interface consists of the following routines:

XmStringBaseline( )
XmStringByteCompare( )
XmStringCompare( )
XmStringConcat( )
XmStringCopy( )
XmStringCreate( )
XmStringCreateLocalized( )
XmStringCreateSimple( )
XmStringDraw( )
XmStringDrawImage( )
XmStringDrawUnderline( )
XmStringEmpty( )
XmStringExtent( )
XmStringFree( )
XmStringFreeContext( )
XmStringGetNextSegment( )
XmStringHasSubstring( )
XmStringHeight( )
XmStringInitContext( )
XmStringLength( )
XmStringLineCount( )
XmStringSegmentCreate( )
XmStringSeparatorCreate( )
XmStringWidth( ).

4.4 XmStringDirection

XmStringDirection is the data type for specifying the direction in which the system displays characters of a string, or characters of a segment of a compound string. It is defined in the header file **<Xm/Xm.h>**. This is an enumeration type with two possible values:

XmSTRING_DIRECTION_L_TO_R
Specifies left to right display.

XmSTRING_DIRECTION_R_TO_L
Specifies right to left display.

Refer to Section 4.3 on page 61 for further information.

4.5 XmStringTable

XmStringTable is the data type for an array of compound strings (objects of type **XmString**). It is defined in the header file **<Xm/Xm.h>**.

Refer to Section 4.3 on page 61 for further information.

4.6 XmTextPosition

XmTextPosition is an integer data type that holds a character's position within a text string. It is defined in the header files **<Xm/Text.h>**, **<Xm/TextF.h>** and **<Xm/Xm.h>**.

An **XmTextPosition** value conceptually points to the gap between two characters. For example, consider a text string consisting of *N* characters. A value of 0 refers to the position immediately prior to the first character. A value of 1 refers to the position between the first and second characters. A value of *N* refers to the position immediately following the last character. Therefore, the text string of *N* characters actually contains *N* + 1 positions.

Refer to the widget *XmText* on page 577 for further information.

Chapter 5

X/Open Motif Interfaces

This chapter contains the reference information for X/Open Motif interfaces.

Some of the function definitions refer to ‘‘interning WM_PROTOCOLS’’. This means take the string WM_PROTOCOLS and issue an *XInternAtom*( ) on it to turn that string into an **Atom**.

NAME

ApplicationShell — the *ApplicationShell* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

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

The class pointer is **applicationShellWidgetClass**.

The class name is *ApplicationShell*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
XmNdefaultFontList	**XmCDefaultFontList**	**XmFontList**	dynamic	CG
XmNdeleteResponse	**XmCDeleteResponse**	**unsigned char**	XmDESTROY	CSG
XmNkeyboardFocusPolicy	**XmCKeyboardFocusPolicy**	**unsigned char**	XmEXPLICIT	CSG
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG
XmNmwmDecorations	**XmCMwmDecorations**	**int**	−1	CG
XmNmwmFunctions	**XmCMwmFunctions**	**int**	−1	CG
XmNmwmInputMode	**XmCMwmInputMode**	**int**	−1	CG
XmNmwmMenu	**XmCMwmMenu**	**String**	NULL	CG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	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**	−1	CSG
XmNiconY	**XmCIconY**	**int**	−1	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
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

SEE ALSO

Composite, *Core*, *Shell*, *WMShell*, *VendorShell* and *TopLevelShell*.

NAME

Composite — the *Composite* widget class

SYNOPSIS

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

DESCRIPTION

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

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

Classes

Composite inherits behavior and resources from *Core*.

The class pointer is **compositeWidgetClass**.

The class name is *Composite*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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 an *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 *insert_position* procedure from the class's *insert_child* procedure to determine where a new child should go in its children array. Thus, a client of a composite class can apply different sorting criteria to widget instances of the class, passing in a different *insert_position* procedure when it creates each composite widget instance.

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

Inherited Resources

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

Core **Resource Set**				
Name	**Class**	**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

SEE ALSO

Core.

NAME

Constraint — the *Constraint* widget class

SYNOPSIS

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

DESCRIPTION

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

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

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

Classes

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

The class pointer is **constraintWidgetClass**.

The class name is *Constraint*.

New Resources

Constraint defines no new resources.

Inherited Resources

Constraint inherits behavior and resources from *Composite* and *Core*. The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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

SEE ALSO

Composite and *Core*.

NAME

Core — the *Core* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

All widgets are built from *Core*.

The class pointer is **widgetClass**.

The class name is *Core*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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

XmNaccelerators

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

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

XmNbackground
Specifies the background color for the widget.

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

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

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

XmNborderWidth
Specifies the width of the border that surrounds the widget's window on all four sides. The width is specified in pixels. A width of 0 (zero) means that no border shows.

XmNcolormap
Specifies the colormap 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 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 is not 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 or leave, and focus events. Insensitive widgets do not receive these events. Use the function *XtSetSensitive*( ) to change the sensitivity argument. Using *XtSetSensitive*( ) ensures that if a parent widget has **XmNsensitive** set to False, the ancestor-sensitive flag of all its children is appropriately set.

XmNtranslations

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

XmNwidth

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

XmNx

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

XmNy

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

SEE ALSO

Object and *RectObj*.

NAME

MrmCloseHierarchy — closes a UID hierarchy

SYNOPSIS

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

Cardinal MrmCloseHierarchy(
      MrmHierarchy              hierarchy_id);
```

DESCRIPTION

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

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

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

[MrmFAILURE]
The function failed.

SEE ALSO

MrmOpenHierarchyPerDisplay( ).

NAME

MrmFetchBitmapLiteral — fetches a bitmap literal from a hierarchy

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

pixmap_return
Returns the resulting X pixmap value.

width Specifies a pointer to the width of the pixmap.

height Specifies a pointer to the height of the pixmap.

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

[MrmNOT_FOUND]
The bitmap literal was not found in the hierarchy.

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

[MrmFAILURE]
The function failed.

SEE ALSO

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

NAME

MrmFetchColorLiteral — fetches a named color literal from a UID file

SYNOPSIS

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

int MrmFetchColorLiteral(
      MrmHierarchy              hierarchy_id,
      String                    index,
      Display                   display,
      Colormap                  colormap,
      Pixmap                    pixel);
```

DESCRIPTION

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

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

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

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

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

pixel Returns the ID of the color literal.

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

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

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

[MrmFAILURE]
The function failed.

SEE ALSO

MrmOpenHierarchyPerDisplay( ), *MrmFetchIconLiteral*( ), *MrmFetchLiteral*( ) and *XOpenDisplay*( ).

NAME

MrmFetchIconLiteral — fetches an icon literal from a hierarchy

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

fgpix Specifies the foreground color for the pixmap.

bgpix Specifies the background color for the pixmap.

pixmap Returns the resulting X pixmap value.

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

[MrmNOT_FOUND]
The icon literal was not found in the hierarchy.

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

[MrmFAILURE]
The function failed.

SEE ALSO

MrmOpenHierarchyPerDisplay( ), *MrmFetchLiteral*( ), *MrmFetchColorLiteral*( ), and *XOpenDisplay*( ).

NAME

MrmFetchLiteral — fetches a literal from a UID file

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

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

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

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

[MrmNOT_FOUND]
The literal was not found in the UIL file.

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

[MrmFAILURE]
The function failed.

SEE ALSO

MrmOpenHierarchyPerDisplay( ), *MrmFetchIconLiteral*( ), *MrmFetchColorLiteral*( ) and *XOpenDisplay*( ).

NAME

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

SYNOPSIS

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

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

DESCRIPTION

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

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

widget Specifies the widget that is modified.

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

num_args Specifies the number of entries in *args*.

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

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

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

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

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmPARTIAL_SUCCESS]
At least one literal was successfully fetched.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

[MrmFAILURE]
The function failed.

SEE ALSO

MrmOpenHierarchyPerDisplay( ) and *XtSetValues*( ).

NAME

MrmFetchWidget — fetches and creates an indexed (UIL named) application widget and its children

SYNOPSIS

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

Cardinal MrmFetchWidget(
      MrmHierarchy              hierarchy_id,
      String                    index,
      Display                   display,
      Pixmap                    pixmap_return,
      Widget                    parent_widget,
      Widget                    widget,
      MrmCode                  *class);
```

DESCRIPTION

The *MrmFetchWidget*( ) function fetches and creates an indexed application widget and its children. The indexed application widget is any widget that is named in UIL. In fetch operations, the fetched widget's subtree is also fetched and created. This widget must not appear as the child of a widget within its own subtree. *MrmFetchWidget*( ) does not execute *XtManageChild* for the newly created widget. All widgets fetched by a call to *MrmFetchWidget*( ) are not managed at the time of their creation callbacks.

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

index Specifies the UIL name of the widget to fetch.

parent_widget Specifies the parent widget ID.

widget Returns the widget ID of the created widget.

class This argument must be set to an actual pointer; it cannot be a NULL pointer. *MrmFetchWidget*( ) sets this argument to an implementation dependent value.

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

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

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

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

[MrmNOT_FOUND]
The widget was not found in UID hierarchy.

[MrmFAILURE]
The function failed.

SEE ALSO

MrmOpenHierarchyPerDisplay( ) and *MrmFetchWidgetOverride*( ).

NAME

MrmFetchWidgetOverride — fetches any indexed (UIL named) application widget and overrides the arguments specified for this application widget in UIL

SYNOPSIS

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

Cardinal MrmFetchWidgetOverride(
      MrmHierarchy              hierarchy_id,
      String                    index,
      Widget                    parent_widget,
      String                    override_name,
      ArgList                   override_args,
      Cardinal                  override_num_args,
      Widget                    widget,
      MrmType                  *class);
```

DESCRIPTION

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

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

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

index Specifies the UIL name of the widget to fetch.

parent_widget Specifies the parent widget ID.

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

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

override_num_args
Specifies the number of arguments in *override_args*.

widget Returns the widget ID of the created widget.

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

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmBAD_HIERARCHY]
The hierarchy ID was invalid.

[MrmNOT_FOUND]
The widget was not found in UID hierarchy.

[MrmFAILURE]
The function failed.

SEE ALSO

MrmOpenHierarchyPerDisplay( ) and *MrmFetchWidget*( ).

NAME

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

SYNOPSIS

```
void MrmInitialize()
```

DESCRIPTION

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

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

NAME

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

SYNOPSIS

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

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

DESCRIPTION

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

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

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

num_files Specifies the number of files in the name list.

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

ancillary_structures_list
A list of ancillary structures dependent on the operating system corresponding to items such as filenames, clobber flags, and so forth. This argument should be NULL for most operations. If you need to reference this structure, see the definition of **MrmOsOpenParamPtr** in Section A.8 on page 698 for more information.

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

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

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

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

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

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

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

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

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

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

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

The following substitutions are used in these paths:

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

%N The class name of the application.

%L The display's language string.

%l The language component of the display's language string.

%S The suffix to the filename. The entire path is first searched with a suffix of **.uid**. If no file is found, it is searched again with a NULL suffix.

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmNOT_FOUND]
File not found.

[MrmFAILURE]
The function failed.

SEE ALSO

MrmCloseHierarchy( ).

NAME

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

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

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

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

class_record Specifies a pointer to the class record.

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmFAILURE]
The function failed.

NAME

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

SYNOPSIS

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

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

DESCRIPTION

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

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

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

register_count Specifies the number of entries in *register_list*.

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

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

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

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

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

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

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

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmFAILURE]
The function failed.

NAME

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

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

register_count Specifies the number of entries in *register_list*.

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

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

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

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

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

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

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

RETURN VALUE

This function returns one of the following status return constants:

[MrmSUCCESS]
The function executed successfully.

[MrmFAILURE]
The function failed.

NAME

Object — the *Object* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

The class pointer is **objectClass**.

The class name is *Object*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

Object **Resource Set**

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

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

NAME

OverrideShell — the *OverrideShell* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

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

The class pointer is **overrideShellWidgetClass**.

The class name is *OverrideShell*.

New Resources

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

Inherited Resources

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

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
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

SEE ALSO

Composite, *Core*, and *Shell*.

NAME

RectObj — the *RectObj* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

RectObj inherits behavior and a resource from *Object*.

The class pointer is **rectObjClass**.

The class name is *RectObj*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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

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* widget's rectangular display area.

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

XmNx
Specifies the x coordinate of the upper-left outside corner of the *RectObj* widget'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* widget'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

SEE ALSO

Object.

NAME

Shell — the *Shell* widget class

SYNOPSIS

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

DESCRIPTION

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

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

Classes

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

The class pointer is **shellWidgetClass**.

The class name is *Shell*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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

XmNallowShellResize

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

XmNcreatePopupChildProc

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

XmNgeometry

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

XmNpopupCallback

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

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

SEE ALSO

Composite and *Core*.

NAME

TopLevelShell — the *TopLevelShell* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

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

The class pointer is **topLevelShellWidgetClass**.

The class name is *TopLevelShell*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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

XmNiconName

Specifies the short form of the application name to be displayed by the window manager when the application is iconified. 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*( )[1] with an encoding

[1] An X Windows function.

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
XmNdefaultFontList	**XmCDefaultFontList**	**XmFontList**	dynamic	CG
XmNdeleteResponse	**XmCDeleteResponse**	**unsigned char**	XmDESTROY	CSG
XmNkeyboardFocusPolicy	**XmCKeyboardFocusPolicy**	**unsigned char**	XmEXPLICIT	CSG
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG
XmNmwmDecorations	**XmCMwmDecorations**	**int**	−1	CG
XmNmwmFunctions	**XmCMwmFunctions**	**int**	−1	CG
XmNmwmInputMode	**XmCMwmInputMode**	**int**	−1	CG
XmNmwmMenu	**XmCMwmMenu**	**String**	NULL	CG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	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**	−1	CSG
XmNiconY	**XmCIconY**	**int**	−1	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 ***	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
XmNinitialResourcesPersistent	**XmCInitialResourcesPersistent**	**Boolean**	True	C
XmNmappedWhenManaged	**XmCMappedWhenManaged**	**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

SEE ALSO

Composite, *Core*, *Shell*, *WMShell*, and *VendorShell*.

NAME

TransientShell — the *TransientShell* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

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

The class pointer is **transientShellWidgetClass**.

The class name is *TransientShell*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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 for which the shell acts as a pop-up window. 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
XmNdefaultFontList	**XmCDefaultFontList**	**XmFontList**	dynamic	CG
XmNdeleteResponse	**XmCDeleteResponse**	**unsigned char**	XmDESTROY	CSG
XmNkeyboardFocusPolicy	**XmCKeyboardFocusPolicy**	**unsigned char**	XmEXPLICIT	CSG
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG
XmNmwmDecorations	**XmCMwmDecorations**	**int**	−1	CG
XmNmwmFunctions	**XmCMwmFunctions**	**int**	−1	CG
XmNmwmInputMode	**XmCMwmInputMode**	**int**	−1	CG
XmNmwmMenu	**XmCMwmMenu**	**String**	NULL	CG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	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**	−1	CSG
XmNiconY	**XmCIconY**	**int**	−1	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 ***	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
XmNinitialResourcesPersistent	**XmCInitialResourcesPersistent**	**Boolean**	True	C
XmNmappedWhenManaged	**XmCMappedWhenManaged**	**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

SEE ALSO

Composite, *Core*, *Shell*, *VendorShell* and *WMShell*.

NAME

Uil — invokes the UIL compiler from within an application

SYNOPSIS

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

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

DESCRIPTION

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

command_desc
Specifies the **uil** command line.

compile_desc Returns the results of the compilation.

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

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

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

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

The data structures **Uil_command_type** and **Uil_compile_desc_type** are as follows:

```
typedef struct Uil_command_type {
  char         *source_file;            /* single source to compile */
  char         *resource_file;          /* name of output file */
  char         *listing_file;           /* name of listing file */
  unsigned int *include_dir_count;      /* no. dirs. in include_dir */
  char         *((*include_dir) []);    /* dir. to search for */
                                        /* include files */
  unsigned      listing_file_flag: 1;   /* produce a listing */
  unsigned      resource_file_flag: 1;  /* generate UID output */
  unsigned      machine_code_flag: 1;   /* generate machine code */
  unsigned      report_info_msg_flag: 1; /* report info messages */
  unsigned      report_warn_msg_flag: 1; /* report warnings */
  unsigned      parse_tree_flag: 1;     /* generate parse tree */
  unsigned int  status_update_delay;     /* number of times a status */
                                         /* point is passed before */
                                         /* calling status_cb */
                                         /* function 0 means called */
                                         /* every time */
};

typedef struct Uil_compile_desc_type {
  unsigned int  compiler_version;        /* version no. of compiler */
  unsigned int  data_version;            /* version no. of structures*/
  char         *parse_tree_root;         /* parse tree output */
  unsigned int  message_count [Uil_k_max_status+1];
                                         /* array of severity counts */
};
```

The following is a description of the message callback function specified by *message_cb*( ):

```
  Uil_continue_type (*message_cb)(
        char                     *message_data,
        int                       message_number,
        int                       severity,
        char                     *msg_buffer,
        char                     *src_buffer,
        char                     *ptr_buffer,
        chr                      *loc_buffer,
        int                       message_count[]);
```

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

[Uil_k_terminate]
Terminate processing of the source file.

[Uil_k_continue]
Continue processing the source file.

The arguments are:

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

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

severity An integer that indicates the severity of the error. The possible values are the status constants returned by the *Uil*( ) function.

msg_buffer A string that describes the error.

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

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

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

message_count

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

The following is a description of the status callback function specified by *status_cb*( ):

```
Uil_continue_type (*status_cb)(
      char                     *status_data,
      int                       percent_complete,
      int                       lines_processed,
      char                     *current_file,
      int                       message_count[]);
```

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

[Uil_k_terminate]

Terminate processing of the source file.

[Uil_k_continue]

Continue processing the source file.

The arguments are:

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

percent_complete

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

lines_processed

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

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

message_count

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

RETURN VALUE

This function returns one of the following status return constants:

[Uil_k_success_status]
The operation succeeded.

[Uil_k_info_status]
The operation succeeded. An informational message is returned.

[Uil_k_warning_status]
The operation succeeded. A warning message is returned.

[Uil_k_error_status]
The operation failed due to an error.

[Uil_k_severe_status]
The operation failed due to an error.

NAME

VendorShell — the *VendorShell* widget class

SYNOPSIS

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

DESCRIPTION

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

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

Classes

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

The class pointer is **vendorShellWidgetClass**.

The class name is *VendorShell*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a 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).

VendorShell **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNaudibleWarning	**XmCAudibleWarning**	**unsigned char**	XmBELL	CSG
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	CSG
XmNdefaultFontList	**XmCDefaultFontList**	**XmFontList**	dynamic	CG
XmNdeleteResponse	**XmCDeleteResponse**	**unsigned char**	XmDESTROY	CSG
XmNkeyboardFocusPolicy	**XmCKeyboardFocusPolicy**	**unsigned char**	XmEXPLICIT	CSG
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG
XmNmwmDecorations	**XmCMwmDecorations**	**int**	−1	CG
XmNmwmFunctions	**XmCMwmFunctions**	**int**	−1	CG
XmNmwmInputMode	**XmCMwmInputMode**	**int**	−1	CG
XmNmwmMenu	**XmCMwmMenu**	**String**	NULL	CG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	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 *VendorShell* widget's button descendants. If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNbuttonFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the *XmBulletinBoard*, *VendorShell* or *XmMenuShell* widget class. If such an ancestor is found, **XmNbuttonFontList** is initialized to the **XmNbuttonFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

XmNdefaultFontList

Specifies a default font list for *VendorShell* widget'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.

XmNkeyboardFocusPolicy

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

XmNlabelFontList

Specifies the font list used for *VendorShell* widget's label descendants (*XmLabels* and *XmLabelGadgets*). If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNlabelFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the *XmBulletinBoard*, *VendorShell* or *XmMenuShell* widget class. If such an ancestor is found, **XmNlabelFontList** is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

XmNmwmDecorations

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

XmNmwmFunctions

Specifies the function flags (specific window manager functions to apply or not to apply to the client window) for the _MOTIF_WM_HINTS property. If any function flags are specified by the _MOTIF_WM_HINTS property, only functions indicated by both that property and the MWM **clientFunctions** and **transientFunctions** resources are applied. If no function flags are specified by the _MOTIF_WM_HINTS property, functions indicated by the MWM **clientFunctions** and **transientFunctions** resources are applied. The default for the **XmNmwmFunctions** resource is not to specify any function flags for the _MOTIF_WM_HINTS property.

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 *XmBulletinBoard* widget's **XmNdialogStyle** resource rather than the parent *XmDialogShell* widget's **XmNmwmInputMode** resource.

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.

XmNtextFontList

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

XmNuseAsyncGeometry

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

Inherited Resources

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

WMShell Resource Set				
Name	**Class**	**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**	–1	CSG
XmNiconY	**XmCIconY**	**int**	–1	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 ***	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
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

SEE ALSO

Composite, *Core*, *mwm*, *Shell*, *WMShell*, *XmActivateProtocol*( ), *XmActivateWMProtocol*( ), *XmAddProtocolCallback*( ), *XmAddWMProtocolCallback*( ), *XmAddProtocols*( ), *XmAddWMProtocols*( ), *XmDeCommon/activate.incotocol*( ), *XmDeactivateWMProtocol*( ), *XmGetAtomName*( ), *XmInternAtom*( ), *Headers/Xm.incMotifWMRunning*( ), *XmRemoveProtocolCallback*( ), *XmRemoveWMProtocolCallback*( ), *XmRemoveProtocols*( ), *XmRemoveWMProtocols*( ), *XmSetProtocolHooks*( ), and *XmSetWMProtocolHooks*( ).

NAME

WMShell — the *WMShell* widget class

SYNOPSIS

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

DESCRIPTION

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

Classes

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

The class pointer is **wmShellWidgetClass**.

The class name is *WMShell*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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**	−1	CSG
XmNiconY	**XmCIconY**	**int**	−1	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

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 non-rectangular.

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 in root window coordinates; this is a hint to the window manager. Because the window manager controls icon placement policy, this resource may be ignored. If no initial value is specified, the value is set to −1 when the widget is realized.

XmNiconY

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

XmNinitialState

Specifies the state in which the application wants the widget instance to start. 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 **ICCCM** specification.

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

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

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

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

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

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

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

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

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

XmNtitleEncoding
Specifies a property type that represents the encoding of the **XmNtitle** string. If a language procedure has been set, the default is None; otherwise, the default is XA_STRING. When the widget is realized, if the value is None, the corresponding name is assumed to be in the current locale. The name is passed to **XmbTextListToTextProperty** with an encoding style of **XStdICCTextStyle**. The resulting encoding is STRING if the name is fully convertible to STRING; otherwise 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 window 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 wait 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**
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
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

SEE ALSO

Composite, *Core* and *Shell*.

NAME

XmActivateProtocol — a *VendorShell* function that activates a protocol

SYNOPSIS

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

void XmActivateProtocol(
      Widget                    shell,
      Atom                      property,
      Atom                      protocol);
```

DESCRIPTION

XmActivateProtocol( ) activates a protocol. It updates the handlers and the *property* if the *shell* is realized. It is sometimes useful to allow a protocol's state information (callback lists, and so on) to persist, even though the client may choose temporarily to resign from the interaction. This is supported by allowing a protocol to be in one of two states: active or inactive. If the *protocol* is active and the *shell* is realized, the *property* contains the *protocol*, which is of type **Atom**. If the *protocol* is inactive, the **Atom** is not present in the *property*.

XmActivateWMProtocol( ) is a convenience interface. It calls *XmActivateProtocol*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

property Specifies the protocol property.

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**).

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmActivateWMProtocol*( ), *XmInternAtom*( ) and *XmRemoveProtocols*( ).

NAME

XmActivateWMProtocol — a *VendorShell* convenience interface that activates a protocol

SYNOPSIS

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

void XmActivateWMProtocol(
      Widget                     shell,
      Atom                       protocol);
```

DESCRIPTION

XmActivateWMProtocol( ) is a convenience interface. It calls *XmActivateProtocol*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**).

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmActivateProtocol*( ) *XmInternAtom*( ) and *XmRemoveWMProtocols*( ).

NAME

XmAddProtocolCallback — a *VendorShell* function that adds client callbacks for a protocol

SYNOPSIS

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

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

DESCRIPTION

XmAddProtocolCallback( ) adds client callbacks for a protocol. It checks if the protocol is registered, and if it is not, calls *XmAddProtocols*( ). It then adds the callback to the internal list. These callbacks are called when the corresponding client message is received.

XmAddWMProtocolCallback( ) is a convenience interface. It calls *XmAddProtocolCallback*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

property Specifies the protocol property.

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**).

callback Specifies the procedure to call when a protocol message is received.

closure Specifies the client data to be passed to the callback when it is invoked.

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddProtocols*( ), *XmAddWMProtocolCallback*( ), *XmInternAtom*( ) and *XmRmovePprotocolCallback*( ).

NAME

XmAddProtocols — a *VendorShell* function that adds the protocols to the protocol manager and allocates the internal tables

SYNOPSIS

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

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

DESCRIPTION

XmAddProtocols( ) adds the protocols to the protocol manager and allocates the internal tables.

XmAddWMProtocols( ) is a convenience interface. It calls *XmAddProtocols*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

property Specifies the protocol property.

protocols Specifies the protocol **Atoms** (or **int** types cast to **Atom**).

num_protocols
Specifies the number of elements in *protocols*

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddWMProtocols*( ) *XmInternAtom*( ) and *XmRemoveProtocols*( ).

NAME

XmAddTabGroup — a function that adds a manager or a primitive widget to the list of tab groups

SYNOPSIS

OB

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

void XmAddTabGroup(
      Widget                    tab_group);
```

DESCRIPTION

This function is obsolete and its behavior is replaced by setting **XmNnavigationType** to XmEXCLUSIVE_TAB_GROUP. When the keyboard is used to traverse through a widget hierarchy, primitive or manager widgets are grouped together into what are known as *tab groups*. Any manager or primitive widget can be a tab group. Within a tab group, move the focus to the next widget in the tab group by using the arrow keys. To move to another tab group, use **KNextField** or **KPrevField**.

Tab groups are ordinarily specified by the **XmNnavigationType** resource. *XmAddTabGroup*( ) is called to control the order of traversal of tab groups. The widget specified by *tab_group* is appended to the list of tab groups to be traversed, and the widget's **XmNnavigationType** is set to XmEXCLUSIVE_TAB_GROUP.

tab_group Specifies the manager or primitive widget ID.

SEE ALSO

XmManager, *XmPrimitive* and *XmRemoveTabGroup*( ).

NAME

XmAddWMProtocolCallback — a *VendorShell* convenience interface that adds client callbacks for a protocol

SYNOPSIS

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

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

DESCRIPTION

XmAddWMProtocolCallback( ) is a convenience interface. It calls *XmAddProtocolCallback*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

protocol Specifies the protocol **Atom** (or an **int** type cast to **Atom**).

callback Specifies the procedure to call when a protocol message is received.

closure Specifies the client data to be passed to the callback when it is invoked.

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddProtocolCallback*( ) *XmInternAtom*( ), and *XmRemoveWMProtocolCallback*( ).

NAME

XmAddWMProtocols — a *VendorShell* convenience interface that adds the protocols to the protocol manager and allocates the internal tables

SYNOPSIS

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

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

DESCRIPTION

XmAddWMProtocols( ) is a convenience interface. It calls *XmAddProtocols*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

protocols Specifies the protocol **Atoms** (or **int** types cast to **Atom**).

num_protocols
Specifies the number of elements in *protocols*.

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddProtocols*( ), *XmInternAtom*( ) and *XmRemoveWMProtocols*( ).

NAME

XmArrowButton — the ArrowButton widget class

SYNOPSIS

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

DESCRIPTION

ArrowButton consists of a directional arrow surrounded by a border shadow. When it is selected, the shadow changes to give the appearance that the ArrowButton has been pressed in. When the ArrowButton is unselected, the shadow reverts to give the appearance that the ArrowButton is released, or out.

Classes

ArrowButton inherits behavior and resources from the *Core* and *XmPrimitive* classes.

The class pointer is **xmArrowButtonWidgetClass**.

The class name is *XmArrowButton*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
XmNdisarmCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmultiClick	**XmCMultiClick**	**unsigned char**	dynamic	CSG

XmNactivateCallback

Specifies a list of callbacks called when the ArrowButton is activated. To activate the button, press and release **BSelect** while the pointer is inside the ArrowButton widget. Activating the ArrowButton also disarms it. The reason sent by this callback is XmCR_ACTIVATE.

XmNarmCallback

Specifies a list of callbacks called when the ArrowButton is armed. To arm this widget, press **BSelect** while the pointer is inside the ArrowButton. The reason sent by this callback is XmCR_ARM.

XmNarrowDirection

Sets the arrow direction. The values for this resource are:

XmARROW_UP
XmARROW_DOWN
XmARROW_LEFT
XmARROW_RIGHT.

XmNdisarmCallback

Specifies a list of callbacks 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	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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                        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.

Action Routines

The *XmArrowButton* action routines are:

Activate( )
Draws the shadow in the unselected state. If the pointer is within the ArrowButton, calls the callbacks for **XmNactivateCallback**.

Arm( )
Draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**.

ArmAndActivate( )

Draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**. Arranges for the shadow to be drawn in the unselected state and the callbacks for **XmNactivateCallback** and **XmNdisarmCallback** to be called, either immediately or at a later time.

Disarm( )

Draws the shadow in the unselected state and calls the callbacks for **XmNdisarmCallback**.

Help( )

Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MultiActivate( )

If **XmNmultiClick** is XmMULTICLICK_DISCARD, this action does nothing.

If **XmNmultiClick** is XmMULTICLICK_KEEP, this action increments **click_count** in the callback structure and draws the shadow in the unselected state. If the pointer is within the ArrowButton, this action calls the callbacks for **XmNactivateCallback** and **XmNdisarmCallback**.

MultiArm( )

If **XmNmultiClick** is XmMULTICLICK_DISCARD, this action does nothing. If **XmNmultiClick** is XmMULTICLICK_KEEP, this action draws the shadow in the selected state and calls the callbacks for **XmNarmCallback**.

SEE ALSO

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

NAME

XmArrowButtonGadget — the ArrowButtonGadget widget class

SYNOPSIS

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

DESCRIPTION

ArrowButtonGadget consists of a directional arrow surrounded by a border shadow. When it is selected, the shadow changes to give the appearance that the ArrowButtonGadget has been pressed in. When it is unselected, the shadow reverts to give the appearance that the button is released, or out.

Classes

ArrowButtonGadget inherits behavior and resources from the *Object*, *RectObj*, and *XmGadget* classes.

The class pointer is **xmArrowButtonGadgetClass**.

The class name is *XmArrowButtonGadget*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
XmNdisarmCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmultiClick	**XmCMultiClick**	**unsigned char**	dynamic	CSG

XmNactivateCallback

Specifies a list of callbacks called when the ArrowButtonGadget is activated. To activate the button, press and release **BSelect** while the pointer is inside the ArrowButtonGadget. Activating the ArrowButtonGadget also disarms it. The reason sent by this callback is XmCR_ACTIVATE.

XmNarmCallback

Specifies a list of callbacks called when the ArrowButtonGadget is armed. To arm this widget, press **BSelect** while the pointer is inside the ArrowButtonGadget. The reason sent by this callback is XmCR_ARM.

XmNarrowDirection

Sets the arrow direction. The values for this resource are:

XmARROW_UP
XmARROW_DOWN
XmARROW_LEFT
XmARROW_RIGHT.

XmNdisarmCallback

Specifies a list of callbacks 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
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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.

SEE ALSO

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

NAME

XmBulletinBoard — 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.

Classes

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

The class pointer is **xmBulletinBoardWidgetClass**.

The class name is *XmBulletinBoard*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
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
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
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 BulletinBoard's button descendants. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the BulletinBoard, *VendorShell*, or MenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNbuttonFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList** for more information on the creation and structure of a font list.

XmNcancelButton

Specifies the widget ID of the **Cancel** button. BulletinBoard's subclasses, which define a **Cancel** button, set this resource. BulletinBoard does not directly provide any behavior for that button.

XmNdefaultButton

Specifies the widget ID of the default button. Some BulletinBoard subclasses, which define a default button, set this resource. BulletinBoard defines translations and installs accelerators that activate that button when **KActivate** is pressed and the keyboard focus is not in another button.

XmNdefaultPosition

Controls whether or not the BulletinBoard is automatically positioned by its parent. If this resource is True, and the parent of the BulletinBoard is a DialogShell, the BulletinBoard is centered within or around the parent of the DialogShell when the BulletinBoard is mapped and managed. If this resource is False, the BulletinBoard is not automatically positioned.

XmNdialogStyle

Indicates the dialog style associated with the BulletinBoard. If the parent of the BulletinBoard is a DialogShell, the parent's **XmNmwmInputMode** is set according to the value of this resource. This resource can be set only if the BulletinBoard is unmanaged. Possible values for this resource include the following:

XmDIALOG_SYSTEM_MODAL

Used for dialogs that must be responded to before any other interaction in any application.

XmDIALOG_PRIMARY_APPLICATION_MODAL

Used for dialogs that must be responded to before some other interactions in ancestors of the widget.

XmDIALOG_APPLICATION_MODAL

Used for dialogs that must be responded to before some other interactions in ancestors of the widget. This value is the same as XmDIALOG_PRIMARY_APPLICATION_MODAL, and remains for compatibility.

XmDIALOG_FULL_APPLICATION_MODAL

Used for dialogs that must be responded to before some other interactions in the same application.

XmDIALOG_MODELESS

Used for dialogs that do not interrupt interaction of any application. This is the default when the parent of the BulletinBoard is a DialogShell.

XmDIALOG_WORK_AREA

Used for BulletinBoard widgets whose parents are not DialogShells. **XmNdialogStyle** is forced to have this value when the parent of the BulletinBoard is not a DialogShell.

XmNdialogTitle

Specifies the dialog title. If this resource is not NULL, and the parent of the BulletinBoard is a subclass of WMShell, BulletinBoard sets the **XmNtitle** and **XmNtitleEncoding** of its parent. If the only character set in **XmNdialogTitle** is that defined in the ISO 8859-1 standard, **XmNtitle** is set to the string of the title, and **XmNtitleEncoding** is set to STRING. If **XmNdialogTitle** contains character sets other than that defined in the ISO 8859-1 standard, **XmNtitle** is set to the string of the title converted to a compound text string, and **XmNtitleEncoding** is set to COMPOUND_TEXT.

XmNfocusCallback

Specifies the list of callbacks called when the BulletinBoard widget or one of its descendants accepts the input focus. The callback reason is XmCR_FOCUS.

XmNlabelFontList

Specifies the font list used for BulletinBoard's label descendants (Labels and LabelGadgets). If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the BulletinBoard, *VendorShell*, or MenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList** for more information on the creation and structure of a font list.

XmNmapCallback

Specifies the list of callbacks 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 visual symbols and top shadow visual symbols are reversed.

XmSHADOW_OUT
Draws the BulletinBoard shadow so that it appears outset.

XmSHADOW_ETCHED_IN
Draws the BulletinBoard shadow using a double line giving the effect of a line etched into the window, similar to the Separator widget.

XmSHADOW_ETCHED_OUT
Draws the BulletinBoard shadow using a double line giving the effect of a line coming out of the window, similar to the Separator widget.

XmNtextFontList
Specifies the font list used for BulletinBoard's Text and List descendants. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard or *VendorShell* widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList** for more information on the creation and structure of a font list.

XmNunmapCallback

Specifies the list of callbacks 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
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 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

SEE ALSO

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

NAME

XmCascadeButton — the CascadeButton widget class

SYNOPSIS

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

DESCRIPTION

CascadeButton links two MenuPanes or a MenuBar to a MenuPane.

It is used in menu systems and must have a RowColumn parent with its **XmNrowColumnType** resource set to XmMENU_BAR, XmMENU_POPUP or XmMENU_PULLDOWN.

It is the only widget that can have a Pulldown MenuPane attached to it as a submenu. The submenu is displayed when this widget is activated within a MenuBar, a PopupMenu, or a PulldownMenu. Its visual symbols can include a label or pixmap and a cascading indicator when it is in a Popup or Pulldown MenuPane; or it can include only a label or a pixmap when it is in a MenuBar.

The default behavior associated with a CascadeButton depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the CascadeButton. In addition, **BMenu** controls the behavior of the CascadeButton if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

A CascadeButton's visual symbols differ from most other button gadgets. When the button becomes armed, its visual symbols 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 visual symbols.

When a CascadeButton within a Pulldown or Popup MenuPane is armed as the result of the user moving the mouse pointer into the widget, it does not immediately display its submenu. Instead, it waits a short amount of time to see if the arming was temporary (that is, the user was simply passing through the widget), or whether the user really wanted the submenu posted. This time delay is configurable using **XmNmappingDelay**.

CascadeButton provides a single mechanism for activating the widget from the keyboard. This mechanism is referred to as a keyboard mnemonic. If a mnemonic has been specified for the widget, the user may activate the CascadeButton by simply typing the mnemonic while the CascadeButton is visible. If the CascadeButton is in a MenuBar and the MenuBar does not have the focus, the **MAlt** modifier must be pressed with the mnemonic. Mnemonics are typically used to interact with a menu using the keyboard interface.

If the Cascadebutton is in a Pulldown or Popup MenuPane and there is a submenu attached, the **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight** and **XmNmarginTop** resources may enlarge to accommodate **XmNcascadePixmap**. **XmNmarginWidth** defaults to 6 if this resource is in a MenuBar; otherwise, it takes Label's default, which is 2.

Classes

CascadeButton inherits behavior and resources from *Core*, *XmPrimitive* and *XmLabel* classes.

The class pointer is **xmCascadeButtonWidgetClass**.

The class name is *XmCascadeButton*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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 called when the user activates the CascadeButton widget and there is no submenu attached to pop up. The activation occurs when a mouse button is released or when the mnemonic associated with the widget is typed. The specific mouse button depends on information in the RowColumn parent. The reason sent by the callback is XmCR_ACTIVATE.

XmNcascadePixmap

Specifies the cascade pixmap displayed on one end of the widget when a CascadeButton is used within a Popup or Pulldown MenuPane and a submenu is attached. The Label class resources **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight** and **XmNmarginTop** may be modified to ensure that room is left for the cascade pixmap. The default cascade pixmap is an arrow pointing to the side of the menu where the submenu appears.

XmNcascadingCallback

Specifies the list of callbacks called just prior to the mapping of the submenu associated with CascadeButton. The reason sent by the callback is XmCR_CASCADING.

XmNmappingDelay

Specifies the amount of time, in milliseconds, between when a CascadeButton becomes armed and when it maps its submenu. This delay is used only when the widget is within a Popup or Pulldown MenuPane. The value must not be negative.

XmNsubMenuId

Specifies the widget ID for the Pulldown MenuPane to be associated with this CascadeButton. The specified MenuPane is displayed when the CascadeButton becomes armed. The MenuPane must have been created with the appropriate parentage depending on the type of menu used. See *XmCreateMenuBar*( ), *XmCreatePulldownMenu*( ) and *XmCreatePopupMenu*( ) 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
XmNacceleratorText	**XmCAcceleratorText**	**XmString**	NULL	N/A
XmNalignment	**XmCAlignment**	**unsigned char**	dynamic	CSG
XmNfontList	**XmCFontList**	**XmFontList**	dynamic	CSG
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	XmFONTLIST_ DEFAULT_TAG	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CSG

XmPrimitive **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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**

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.

SEE ALSO

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

NAME

XmCascadeButtonGadget — the CascadeButtonGadget widget class

SYNOPSIS

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

DESCRIPTION

CascadeButtonGadget links two MenuPanes, a MenuBar to a MenuPane, or an OptionMenu to a MenuPane.

It is used in menu systems and must have a RowColumn parent with its **XmNrowColumnType** resource set to XmMENU_BAR, XmMENU_POPUP, XmMENU_PULLDOWN, or XmMENU_OPTION.

It is the only gadget that can have a Pulldown MenuPane attached to it as a submenu. The submenu is displayed when this gadget is activated within a PopupMenu, a PulldownMenu, or an OptionMenu. Its visual symbols can include a label or pixmap and a cascading indicator when it is in a Popup or Pulldown MenuPane; or it can include only a label or a pixmap when it is in an OptionMenu.

The default behavior associated with a CascadeButtonGadget depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the CascadeButtonGadget. In addition, **BMenu** controls the behavior of the CascadeButtonGadget if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

A CascadeButtonGadget's visual symbols differ from most other button gadgets. When the button becomes armed, its visual symbols 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 visual symbols.

When a CascadeButtonGadget within a Pulldown or Popup MenuPane is armed as the result of the user moving the mouse pointer into the gadget, it does not immediately display its submenu. Instead, it waits a short time to see if the arming is temporary (that is, the user was simply passing through the gadget), or the user really wanted the submenu posted. This delay is configurable using **XmNmappingDelay**.

CascadeButtonGadget provides a single mechanism for activating the gadget from the keyboard. This mechanism is referred to as a keyboard mnemonic. If a mnemonic has been specified for the gadget, the user may activate it by simply typing the mnemonic while the CascadeButtonGadget is visible. If the CascadeButtonGadget is in a MenuBar and the MenuBar does not have focus, the **MAlt** modifier must be pressed with the mnemonic. Mnemonics are typically used to interact with a menu using the keyboard.

If a CascadeButtonGadget is in a Pulldown or Popup MenuPane and there is a submenu attached, the **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight** and **XmNmarginTop** resources may enlarge to accommodate **XmNcascadePixmap**. **XmNmarginWidth** defaults to 6 if this resource is in a MenuBar; otherwise, it takes LabelGadget's default, which is 2.

Classes

CascadeButtonGadget inherits behavior and resources from the *Object*, *RectObj*, *XmGadget*, and *XmLabelGadget* classes.

The class pointer is **xmCascadeButtonGadgetClass**.

The class name is *XmCascadeButtonGadget*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmCascadeButtonGadget **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 CascadeButtonGadget, and there is no submenu attached to pop up. The activation occurs when a mouse button is released or when the mnemonic associated with the gadget is typed. The specific mouse button depends on information in the RowColumn parent. The reason sent by the callback is XmCR_ACTIVATE.

XmNcascadePixmap

Specifies the cascade pixmap displayed on one end of the gadget when a CascadeButtonGadget is used within a Popup or Pulldown MenuPane and a submenu is attached. The LabelGadget class resources **XmNmarginBottom**, **XmNmarginLeft**, **XmNmarginRight** and **XmNmarginTop** may be modified to ensure that room is left for the cascade pixmap. The default cascade pixmap in menus other than option menus is an arrow pointing to the side of the menu where the submenu will appear. The default for the CascadeButtonGadget in an option menu is XmUNSPECIFIED_PIXMAP.

XmNcascadingCallback

Specifies the list of callbacks that is called just prior to the mapping of the submenu associated with the CascadeButtonGadget. The reason sent by the callback is XmCR_CASCADING.

XmNmappingDelay

Specifies the amount of time, in milliseconds, between when a CascadeButtonGadget becomes armed and when it maps its submenu. This delay is used only when the gadget is within a Popup or Pulldown MenuPane. The value must not be negative.

XmNsubMenuId

Specifies the widget ID for the Pulldown MenuPane to be associated with this CascadeButtonGadget. The specified MenuPane is displayed when the

CascadeButtonGadget becomes armed. The MenuPane must have been created with the appropriate parentage depending on the type of menu used. See *XmCreatePulldownMenu*( ), *XmCreatePopupMenu*( ) and *XmCreateOptionMenu*( ) 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
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	dynamic	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	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**

SEE ALSO

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

NAME

XmCascadeButtonHighlight — a CascadeButton and CascadeButtonGadget function that sets the highlight state

SYNOPSIS

```
#include <Xm/CascadeB.h>
#include <Xm/CascadeBG.h>

void XmCascadeButtonHighlight(
      Widget                    cascadeButton,
      Boolean                   highlight);
```

DESCRIPTION

XmCascadeButtonHighlight( ) either draws or erases the shadow highlight around the CascadeButton or the CascadeButtonGadget.

cascadeButton Specifies the CascadeButton or CascadeButtonGadget to be highlighted or unhighlighted

highlight Specifies whether to highlight (True) or to unhighlight (False)

For a complete definition of CascadeButton or CascadeButtonGadget and their associated resources, see *XmCascadeButton* or *XmCascadeButtonGadget*.

SEE ALSO

XmCascadeButton and *XmCascadeButtonGadget*.

NAME

XmChangeColor — recalculates all associated colors of a widget

SYNOPSIS

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

void XmChangeColor(
      Widget                    widget,
      Pixel                     background);
```

DESCRIPTION

XmChangeColor( ) handles all color modifications for the specified widget when a new background pixel value is specified. This function recalculates the foreground, select and shadow colors based on the new background color and sets the corresponding resources for the widget. If a color calculation procedure has been set, *XmChangeColor*( ) uses that procedure to calculate the new colors. Otherwise, the routine uses a default procedure.

widget Specifies the widget ID whose colors are updated.

background Specifies the background color pixel value.

SEE ALSO

XmGetColors( ).

NAME

XmClipboardCancelCopy — a clipboard function that cancels a copy to the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardCancelCopy( ) cancels the copy to clipboard that is in progress and frees up temporary storage. When a copy is to be performed, *XmClipboardStartCopy*( ) allocates temporary storage for the clipboard data. *XmClipboardCopy*( ) copies the appropriate data into the temporary storage. *XmClipboardEndCopy*( ) copies the data to the clipboard structure and frees up the temporary storage structures. If *XmClipboardCancelCopy*( ) is called, the *XmClipboardEndCopy*( ) function does not have to be called. A call to *XmClipboardCancelCopy*( ) is valid only after a call to *XmClipboardStartCopy*( ) and before a call to *XmClipboardEndCopy*( ).

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies a widget's window ID that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to *XmClipboardStartCopy*( ).

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardFail]
The function failed because *XmClipboardStartCopy*( ) was not called or because the data item contains too many formats.

SEE ALSO

XmClipboardCopy( ), *XmClipboardEndCopy*( ) and *XmClipboardStartCopy*( ).

NAME

XmClipboardCopy — a clipboard function that copies a data item to temporary storage for later copying to clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardCopy( ) copies a data item to temporary storage. The data item is moved from temporary storage to the clipboard data structure when a call to *XmClipboardEndCopy*( ) is made. Additional calls to *XmClipboardCopy*( ) before a call to *XmClipboardEndCopy*( ) add additional data item formats to the same data item or append data to an existing format. Formats are described in the **ICCCM** specification as targets.

Note: Do not call *XmClipboardCopy*( ) before a call to *XmClipboardStartCopy*( ) has been made. The latter function allocates temporary storage required by *XmClipboardCopy*( ).

If the *buffer* argument is NULL, the data is considered to be passed by name. When data that has been passed by name is later requested by another application, the application that owns the data receives a callback with a request for the data. The application that owns the data must then transfer the data to the clipboard with the *XmClipboardCopyByName*( ) function. When a data item that was passed by name is deleted from the clipboard, the application that owns the data receives a callback stating that the data is no longer needed.

For information on the callback function, see the callback argument description for *XmClipboardStartCopy*( ).

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to *XmClipboardStartCopy*( ).

format_name Specifies the name of the format in which the data item is stored on the clipboard. The format was known as target in the **ICCCM** specification.

buffer Specifies the buffer from which the clipboard copies the data.

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

private_id Specifies the private data that the application wants to store with the data item.

data_id Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This argument is required only for data that is passed by name.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardFail]
The function failed because *XmClipboardStartCopy*( ) was not called or because the data item contains too many formats.

SEE ALSO

XmClipboardCopyByName( ), *XmClipboardEndCopy*( ) and *XmClipboardStartCopy*( ).

NAME

XmClipboardCopyByName — a clipboard function that copies a data item passed by name

SYNOPSIS

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

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

DESCRIPTION

XmClipboardCopyByName( ) copies the actual data for a data item that was previously passed by name to the clipboard. Data is considered to be passed by name when a call to *XmClipboardCopy*( ) is made with a NULL buffer argument. Additional calls to this function append new data to the existing data.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each clipboard function it calls.

data_id Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This number was assigned by *XmClipboardCopy*( ) to the data item.

buffer Specifies the buffer from which the clipboard copies the data.

length Specifies the number of bytes in the data item.

private_id Specifies the private data that the application wants to store with the data item.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardCopy( ), *XmClipboardLock*( ), *XmClipboardStartCopy*( ) and *XmClipboardUnlock*( ).

NAME

XmClipboardEndCopy — a clipboard function that ends a copy to the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardEndCopy( ) locks the clipboard from access by other applications, places data in the clipboard data structure, and unlocks the clipboard. Data items copied to the clipboard by *XmClipboardCopy*( ) are not actually entered in the clipboard data structure until the call to *XmClipboardEndCopy*( ).

This function also frees up temporary storage that was allocated by *XmClipboardStartCopy*( ), which must be called before *XmClipboardEndCopy*( ). The latter function should not be called if *XmClipboardCancelCopy*( ) has been called.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each clipboard function it calls.

item_id Specifies the number assigned to this data item, which was returned by a previous call to *XmClipboardStartCopy*( ).

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardFail]
The function failed because *XmClipboardStartCopy*( ) was not called.

SEE ALSO

XmClipboardCancelCopy( ), *XmClipboardCopy*( ) and *XmClipboardStartCopy*( ).

NAME

XmClipboardEndRetrieve — a clipboard function that ends a copy from the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardEndRetrieve( ) suspends copying data incrementally from the clipboard. It tells the clipboard routines that the application is through copying an item from the clipboard. Until this function is called, data items can be retrieved incrementally from the clipboard with *XmClipboardRetrieve*( ).

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained with *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardRetrieve( ), *XmClipboardStartCopy*( ) and *XmClipboardStartRetrieve*( ).

NAME

XmClipboardInquireCount — a clipboard function that returns the number of data item formats

SYNOPSIS

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

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

DESCRIPTION

XmClipboardInquireCount( ) returns the number of data item formats available for the data item in the clipboard. This function also returns the maximum name length for all formats in which the data item is stored.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

count Returns the number of data item formats available for the data item in the clipboard. If no formats are available, this argument equals 0 (zero). The count includes the formats that were passed by name.

max_format_name_length
Specifies the maximum length of all format names for the data item in the clipboard.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardNoData]
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

SEE ALSO

XmClipboardStartCopy( ).

NAME

XmClipboardInquireFormat — a clipboard function that returns a specified format name

SYNOPSIS

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

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

DESCRIPTION

XmClipboardInquireFormat( ) returns a specified format name for the data item in the clipboard. If the name must be truncated, the function returns a warning status.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

index Specifies which of the ordered format names to obtain. If this index is greater than the number of formats for the data item, this function returns a 0 (zero) in the *copied_len* argument.

format_name_buf
Specifies the buffer that receives the format name.

buffer_len Specifies the number of bytes in the format name buffer.

copied_len Specifies the number of bytes in the string copied to the buffer. If this argument equals 0 (zero), there is no *n*th format for the data item.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardTruncate]
The data returned is truncated because the user did not provide a buffer large enough to hold the data.

[ClipboardNoData]

The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

SEE ALSO

XmClipboardStartCopy( ).

NAME

XmClipboardInquireLength — a clipboard function that returns the length of the stored data

SYNOPSIS

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

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

DESCRIPTION

XmClipboardInquireLength( ) returns the length of the data stored under a specified format name for the clipboard data item. If no data is found for the specified format, or if there is no item on the clipboard, this function returns a value of 0 (zero).

Any format passed by name is assumed to have *length* passed in a call to *XmClipboardCopy*( ), even though the data has not yet been transferred to the clipboard in that format.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name Specifies the name of the format for the data item.

length Specifies the length of the next data item in the specified format. This argument equals 0 (zero) if no data is found for the specified format, or if there is no item on the clipboard.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardNoData]
The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard, but not in the requested format; or the data in the requested format was passed by name and is no longer available.

SEE ALSO

XmClipboardCopy( ) and *XmClipboardStartCopy*( ).

NAME

XmClipboardInquirePendingItems — a clipboard function that returns a list of data ID and private ID pairs

SYNOPSIS

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

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

DESCRIPTION

XmClipboardInquirePendingItems( ) returns a list of data ID and private ID pairs for the specified format name. A data item is considered pending if the application originally passed it by name, the application has not yet copied the data and the item has not been deleted from the clipboard. The application is responsible for freeing the memory provided by this function to store the list. To free the memory, call *XtFree*( ).

This function is used by an application when exiting, to determine if the data that is passed by name should be sent to the clipboard.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name Specifies a string that contains the name of the format for which the list of data ID and private ID pairs is to be obtained.

item_list Specifies the address of the array of data ID and private ID pairs for the specified format name. This argument is a type **XmClipboardPendingList**. The application is responsible for freeing the memory provided by this function for storing the list.

count Specifies the number of items returned in the list. If there is no data for the specified format name, or if there is no item on the clipboard, this argument equals 0 (zero).

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardStartCopy( ).

NAME

XmClipboardLock — a clipboard function that locks the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardLock( ) locks the clipboard from access by another application until *XmClipboardUnlock*( ) is called. All clipboard functions lock and unlock the clipboard to prevent simultaneous access. This function allows the application to keep the clipboard data from changing between calls to *Inquire*() and other clipboard functions. The application does not need to lock the clipboard between calls to *XmClipboardStartCopy*( ) and *XmClipboardEndCopy*( ) or to *XmClipboardStartRetrieve*( ) and *XmClipboardEndRetrieve*( ).

If the clipboard is already locked by another application, *XmClipboardLock*( ) returns an error status. Multiple calls to this function by the same application increase the lock level.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardEndCopy( ), *XmClipboardEndRetrieve*( ), *XmClipboardStartCopy*( ), *XmClipboardStartRetrieve*( ) and *XmClipboardUnlock*( ).

NAME

XmClipboardRegisterFormat — a clipboard function that registers a new format

SYNOPSIS

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

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

DESCRIPTION

XmClipboardRegisterFormat( ) registers a new format. Each format stored on the clipboard should have a length associated with it; this length must be known to the clipboard routines. Formats are known as targets in the **ICCCM** specification. All of the formats specified by the **ICCCM** specification conventions are pre-registered. Any other format that the application wants to use must either be 8-bit data or be registered through this routine. Failure to register the length of the data results in incompatible applications across platforms having different byte-swapping orders.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

format_name Specifies the string name for the new format (target).

format_length Specifies the format length in bits (8, 16 or 32).

RETURN VALUE

[ClipboardBadFormat]
The *format_name* must not be NULL and the *format_length* must be 8, 16 or 32.

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardFail]
The function failed because the format was already registered with this length.

SEE ALSO

XmClipboardStartCopy( ).

NAME

XmClipboardRetrieve — a clipboard function that retrieves a data item from the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardRetrieve( ) retrieves the current data item from clipboard storage. It returns a warning if the clipboard is locked, if there is no data on the clipboard or if the data needs to be truncated because the buffer length is too short.

Between a call to *XmClipboardStartRetrieve*( ) and a call to *XmClipboardEndRetrieve*( ), multiple calls to *XmClipboardRetrieve*( ) with the same format name result in data being incrementally copied from the clipboard until the data in that format has all been copied.

The return value [ClipboardTruncate] from calls to *XmClipboardRetrieve*( ) indicates that more data remains to be copied in the given format. It is recommended that any calls to the *Inquire*( ) functions that the application needs to make to effect the copy from the clipboard be made between the call to *XmClipboardStartRetrieve*( ) and the first call to *XmClipboardRetrieve*( ). This way, the application does not need to call *XmClipboardLock*( ) and *XmClipboardUnlock*( ).

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

format_name Specifies the name of a format in which the data is stored on the clipboard.

buffer Specifies the buffer to which the application wants the clipboard to copy the data.

length Specifies the length of the application buffer.

num_bytes Specifies the number of bytes of data copied into the application buffer.

private_id Specifies the private data stored with the data item by the application that placed the data item on the clipboard. If the application did not store private data with the data item, this argument returns 0 (zero).

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock

goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

[ClipboardTruncate]

The data returned is truncated because the user did not provide a buffer large enough to hold the data.

[ClipboardNoData]

The function could not find data on the clipboard corresponding to the format requested. This could occur because the clipboard is empty; there is data on the clipboard but not in the requested format; or the data in the requested format was passed by name and is no longer available.

SEE ALSO

XmClipboardEndRetrieve( ), *XmClipboardLock*( ), *XmClipboardStartCopy* ( ), *XmClipboardStartRetrieve*( ), and *XmClipboardUnlock*( ).

NAME

XmClipboardStartCopy — a clipboard function that sets up a storage and data structure

SYNOPSIS

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

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

DESCRIPTION

XmClipboardStartCopy( ) sets up storage and data structures to receive clipboard data. An application calls this function during a cut or copy operation. The data item that these structures receive then becomes the next data item in the clipboard.

Copying a large piece of data to the clipboard can take a long time. It is possible that, once the data is copied, no application ever requests that data. The Motif Toolkit provides a mechanism so that an application does not need actually to pass data to the clipboard until the data has been requested by some application.

Instead, the application passes format and length information in *XmClipboardCopy*( ) to the clipboard functions, along with a widget ID and a callback function address that is passed in *XmClipboardStartCopy*( ). The widget ID is necessary for communication between the clipboard functions in the application that owns the data and the clipboard functions in the application that requests the data.

The callback functions are responsible for copying the actual data to the clipboard through *XmClipboardCopyByName*( ). The callback function is also called if the data item is removed from the clipboard and the actual data is no longer needed.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

clip_label Specifies the label to be associated with the data item. This argument is used to identify the data item, for example, in a clipboard viewer. An example of a label is the name of the application that places the data in the clipboard.

timestamp Specifies the time of the event that triggered the copy. A valid timestamp must be supplied; it is not sufficient to use **CurrentTime**.

widget Specifies the ID of the widget that receives messages requesting data previously passed by name. This argument must be present in order to pass data by name. Any valid widget ID in your application can be used for this purpose and all the message handling is taken care of by the cut and paste functions.

callback Specifies the address of the callback function that is called when the clipboard needs data that was originally passed by name. This is also the callback to receive the delete message for items that were originally passed by name. This argument must be present in order to pass data by name.

item_id Specifies the number assigned to this data item. The application uses this number in calls to *XmClipboardCancelCopy*( ), *XmClipboardEndCopy*( ) and *XmClipboardCopy*( ).

For more information on passing data by name, see *XmClipboardCopy*( ) and *XmClipboardCopyByName*( ).

The *widget* and *callback* arguments must be present in order to pass data by name. The callback format is as follows:

```
void (*callback)(
   Widget       widget,
   int         *data_id,
   int         *private,
   int         *reason);
```

widget Specifies the ID of the widget passed to this function.

data_id Specifies the identifying number returned by *XmClipboardCopy*( ), which identifies the pass-by-name data.

private Specifies the private information passed to *XmClipboardCopy*( ).

reason Specifies the reason; the possible values are:

XmCR_CLIPBOARD_DATA_DELETE
XmCR_CLIPBOARD_DATA_REQUEST.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardCancelCopy( ), *XmClipboardCopy*( ), *XmClipboardCopyByName*( ), *XmClipboardEndCopy*( ), *XmClipboardEndRetrieve*( ), *XmClipboardInquireCount*( ), *XmClipboardInquireFormat*( ), *XmClipboardInquireLength*( ), *XmClipboardInquirePendingItems*( ), *XmClipboardLock*( ), *XmClipboardRegisterFormat*( ), *XmClipboardRetrieve*( ), *XmClipboardStartRetrieve*( ), *XmClipboardUndoCopy*( ), *XmClipboardUnlock*( ) and *XmClipboardWithdrawFormat*( ).

NAME

XmClipboardStartRetrieve — a clipboard function that starts a copy from the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardStartRetrieve( ) tells the clipboard routines that the application is ready to start copying an item from the clipboard. The clipboard is locked by this routine and stays locked until *XmClipboardEndRetrieve*( ) is called. Between a call to *XmClipboardStartRetrieve*( ) and a call to *XmClipboardEndRetrieve*( ), multiple calls to *XmClipboardRetrieve*( ) with the same format name result in data being incrementally copied from the clipboard until the data in that format has all been copied.

A return value of [ClipboardTruncate] from calls to *XmClipboardRetrieve*( ) indicates that more data remains to be copied in the given format. It is recommended that any calls to the *Inquire*( ) functions that the application needs to make to complete the copy from the clipboard be made between the call to *XmClipboardStartRetrieve*( ) and the first call to *XmClipboardRetrieve*( ). This way, the application does not need to call *XmClipboardLock*( ) and *XmClipboardUnlock*( ).

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

timestamp Specifies the time of the event that triggered the copy. A valid timestamp must be supplied; it is not sufficient to use **CurrentTime**.

RETURN VALUE

[ClipboardSuccess]
The function is successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardEndRetrieve( ), *XmClipboardInquireCount*( ), *XmClipboardInquireFormat*( ), *XmClipboardInquireLength*( ), *XmClipboardInquirePendingItems*( ), *XmClipboardLock*( ), *XmClipboardRetrieve*( ), *XmClipboardStartCopy* ( ) and *XmClipboardUnlock*( ).

NAME

XmClipboardUndoCopy — a clipboard function that deletes the last item placed on the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardUndoCopy( ) deletes the last item placed on the clipboard if the item was placed there by an application with the passed *display* and *window* arguments. Any data item deleted from the clipboard by the original call to *XmClipboardCopy*( ) is restored. If the *display* or *window* IDs do not match the last copied item, no action is taken; this function has no effect.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each clipboard function it calls.

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardLock( ) and *XmClipboardStartCopy*( ).

NAME

XmClipboardUnlock — a clipboard function that unlocks the clipboard

SYNOPSIS

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

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

DESCRIPTION

XmClipboardUnlock( ) unlocks the clipboard, enabling it to be accessed by other applications.

If multiple calls to *XmClipboardLock*() have occurred, the same number of calls to *XmClipboardUnlock*( ) is necessary to unlock the clipboard, unless *remove_all_locks* is set to True.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*(). The same application instance should pass the same window ID to each of the clipboard functions that it calls.

remove_all_locks

When True, indicates that all nested locks should be removed. When False, indicates that only one level of lock should be removed.

RETURN VALUE

[ClipboardSuccess]

The function was successful.

[ClipboardFail]

The function failed because the clipboard was not locked or was locked by another application.

SEE ALSO

XmClipboardCancelCopy ( ), *XmClipboardCopy* ( ), *XmClipboardEndCopy* ( ), *XmClipboardEndRetrieve*( ), *XmClipboardInquireCount*( ), *XmClipboardInquireFormat*( ), *XmClipboardInquireLength*( ), *XmClipboardInquirePendingItems*( ), *XmClipboardLock*( ), *XmClipboardRegisterFormat*( ), *XmClipboardRetrieve*( ), *XmClipboardStartCopy* ( ), *XmClipboardStartRetrieve*( ), *XmClipboardUndoCopy* ( ) and *XmClipboardWithdrawFormat*( ).

NAME

XmClipboardWithdrawFormat — a clipboard function that indicates that the application no longer wants to supply a data item

SYNOPSIS

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

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

DESCRIPTION

XmClipboardWithdrawFormat( ) indicates that the application no longer supplies a data item to the clipboard that the application had previously passed by name.

display Specifies a pointer to the **Display** structure that was returned in a previous call to *XOpenDisplay*( ) or *XtDisplay*( ).

window Specifies the window ID of a widget that relates the application window to the clipboard. The widget's window ID can be obtained through *XtWindow*( ). The same application instance should pass the same window ID to each clipboard function it calls.

data_id Specifies an identifying number assigned to the data item, which uniquely identifies the data item and the format. This was assigned to the item when it was originally passed by *XmClipboardCopy*( ).

RETURN VALUE

[ClipboardSuccess]
The function was successful.

[ClipboardLocked]
The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same arguments until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

SEE ALSO

XmClipboardCopy( ) and *XmClipboardStartCopy*( ).

NAME

XmCommand — the Command widget class

SYNOPSIS

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

DESCRIPTION

Command is a special-purpose composite widget for command entry that provides a built-in command-history mechanism. Command includes a command-line text-input field, a command-line prompt, and a command-history list region.

One additional **WorkArea** child may be added to the Command after creation.

Whenever a command is entered, it is automatically added to the end of the command-history list and made visible. This does not change the selected item in the list, if there is one.

Many of the new resources specified for Command are actually SelectionBox resources that have been renamed for clarity and ease of use.

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
Text	*XmTextField*	command-line text-input field
Selection	*XmLabel*	command-line prompt
ItemsList	*XmList*	command-history list region
WorkArea	Dynamic	optional work area

Classes

Command inherits behavior and resources 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 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).

XmCommand **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNcommand	**XmCTextString**	**XmString**	""	CSG
XmNcommandChangedCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNcommandEnteredCallback	**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 called when the value of the command changes. The callback reason is XmCR_COMMAND_CHANGED. This is equivalent to the **XmNvalueChangedCallback** of the Text widget, except that a pointer to an **XmCommandCallbackStructure** is passed, and the structure's **value** member contains the **XmString**.

XmNcommandEnteredCallback

Specifies the list of callbacks 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.

XmNhistoryItemCount

Specifies the number of **XmStrings** in **XmNhistoryItems**. This is the **XmNlistItemCount** resource in SelectionBox, renamed for Command. The value must not be negative.

XmNhistoryMaxItems

Specifies the maximum number of items allowed in the history list. Once this number is reached, an existing list item must be removed before a new item can be added to the list. For each command entered, the first list item is removed from the list, so the new command can be added to the list. The value must be greater than 0 (zero).

XmNhistoryVisibleItemCount

Specifies the number of items in the history list that should be visible at one time. In effect, it sets the height (in lines) of the history list window. This is the **XmNlistVisibleItemCount** resource in SelectionBox, renamed for Command. The value must be greater than 0 (zero). The default is dynamic based on the height of the list.

XmNpromptString

Specifies a prompt for the command line. This is the **XmNselectionLabelString** resource in SelectionBox, renamed for Command. The default may vary depending on the value of the **XmNstringDirection** resource and the locale. In the C locale the default is > (right angle bracket).

Inherited Resources

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

XmSelectionBox **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNapplyCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNapplyLabelString	**XmCApplyLabelString**	**XmString**	dynamic	N/A
XmNcancelCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNcancelLabelString	**XmCCancelLabelString**	**XmString**	dynamic	N/A
XmNchildPlacement	**XmCChildPlacement**	**unsigned char**	XmPLACE _ABOVE_ SELECTION	CSG
XmNdialogType	**XmCDialogType**	**unsigned char**	XmDIALOG _COMMAND	G
XmNhelpLabelString	**XmCHelpLabelString**	**XmString**	dynamic	N/A
XmNlistItemCount	**XmCItemCount**	**int**	0	CSG
XmNlistItems	**XmCItems**	**XmStringTable**	NULL	CSG
XmNlistLabelString	**XmCListLabelString**	**XmString**	NULL	N/A
XmNlistVisibleItemCount	**XmCVisibleItemCount**	**int**	dynamic	CSG
XmNminimizeButtons	**XmCMinimizeButtons**	**Boolean**	False	N/A
XmNmustMatch	**XmCMustMatch**	**Boolean**	False	N/A
XmNnoMatchCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNokCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNokLabelString	**XmCOkLabelString**	**XmString**	dynamic	N/A
XmNselectionLabelString	**XmCSelectionLabelString**	**XmString**	dynamic	CSG
XmNtextAccelerators	**XmCTextAccelerators**	**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
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
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
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 each callback:

```
typedef struct
{
      int                       reason;
      XEvent                   *event;
      XmString                  value;
      int                       length;
} XmCommandCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

value Specifies the **XmString** in the CommandArea.

length Specifies the size of the command in **XmString**.

Action Routines

The *XmCommand* action routines are:

SelectionBoxUpOrDown(Previous | Next | First | Last)

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

When called with an argument of 1 (or optionally Next), selects the next item in the history list and replaces the text with that item.

When called with an argument of 2 (or optionally First), selects the first item in the history list and replaces the text with that item.

When called with an argument of 3 (or optionally Last), selects the last item in the history list and replaces the text with that item.

An implementation may optionally implement the values Previous, Next, First, Last in addition to the numeric values.

Calls the callbacks for **XmNcommandChangedCallback**.

SEE ALSO

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

NAME

XmCommandAppendValue — a Command function that appends the passed XmString to the end of the string displayed in the command area of the widget

SYNOPSIS

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

void XmCommandAppendValue(
      Widget                      widget,
      XmString                    command);
```

DESCRIPTION

XmCommandAppendValue( ) appends the passed **XmString** to the end of the string displayed in the command area of the Command widget.

widget Specifies the Command widget ID.

command Specifies the passed **XmString**.

For a complete definition of Command and its associated resources, see *XmCommand*.

SEE ALSO

XmCommand.

NAME

XmCommandError — a Command function that displays an error message

SYNOPSIS

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

void XmCommandError(
      Widget                      widget,
      XmString                    error);
```

DESCRIPTION

XmCommandError( ) displays an error message in the history area of the Command widget. The **XmString** error is displayed until the next command entered occurs.

widget Specifies the Command widget ID.

error Specifies the passed **XmString**.

For a complete definition of Command and its associated resources, see *XmCommand*.

SEE ALSO

XmCommand.

NAME

XmCommandGetChild — a Command function that is used to access a component

SYNOPSIS

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

Widget XmCommandGetChild(
      Widget                    widget,
      unsigned char             child);
```

DESCRIPTION

XmCommandGetChild( ) is used to access a component within a Command. The arguments given to the function are the Command widget and a value indicating which component to access.

widget Specifies the Command widget ID.

child Specifies a component within the Command. The following values are legal for this argument:

XmDIALOG_COMMAND_TEXT
XmDIALOG_PROMPT_LABEL
XmDIALOG_HISTORY_LIST
XmDIALOG_WORK_AREA.

For a complete definition of Command and its associated resources, see *XmCommand*.

RETURN VALUE

Returns the widget ID of the specified Command component. An application should not assume that the returned widget is of any particular class.

SEE ALSO

XmCommand.

NAME

XmCommandSetValue — a Command function that replaces a displayed string

SYNOPSIS

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

void XmCommandSetValue(
      Widget                      widget,
      XmString                    command);
```

DESCRIPTION

XmCommandSetValue( ) replaces the string displayed in the command area of the Command widget with the passed **XmString**.

widget Specifies the Command widget ID.

command Specifies the passed **XmString**.

For a complete definition of Command and its associated resources, see *XmCommand*.

SEE ALSO

XmCommand.

NAME

XmConvertUnits — a function that converts a value in one unit type to another unit type

SYNOPSIS

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

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

DESCRIPTION

XmConvertUnits( ) converts the value and returns it as the return value from the function.

widget Specifies the widget for which the data is to be converted.

orientation Specifies whether the converter uses the horizontal or vertical screen resolution when performing the conversions. The *orientation* argument can have values of XmHORIZONTAL or XmVERTICAL.

from_unit_type
Specifies the current unit type of the supplied value.

from_value Specifies the value to be converted.

to_unit_type Converts the value to the unit type specified.

The arguments *from_unit_type* and *to_unit_type* can have the following values:

XmPIXELS
All values provided to the widget are treated as normal pixel values. This is the default for the resource.

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

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

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

RETURN VALUE

Returns the converted value. If a NULL widget, incorrect *orientation*, or incorrect *unit_type* is supplied as argument data, 0 (zero) is returned.

NAME

XmCreateArrowButton — the ArrowButton widget creation function

SYNOPSIS

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

Widget XmCreateArrowButton(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateArrowButton( ) creates an instance of an ArrowButton widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of ArrowButton and its associated resources, see *XmArrowButton*.

RETURN VALUE

Returns the ArrowButton widget ID.

SEE ALSO

XmArrowButton.

NAME

XmCreateArrowButtonGadget — the ArrowButtonGadget creation function

SYNOPSIS

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

Widget XmCreateArrowButtonGadget(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateArrowButtonGadget( ) creates an instance of an ArrowButtonGadget widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of ArrowButtonGadget and its associated resources, see *XmArrowButtonGadget*.

RETURN VALUE

Returns the ArrowButtonGadget widget ID.

SEE ALSO

XmArrowButtonGadget.

NAME

XmCreateBulletinBoard — the BulletinBoard widget creation function

SYNOPSIS

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

Widget XmCreateBulletinBoard(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateBulletinBoard( ) creates an instance of a BulletinBoard widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of BulletinBoard and its associated resources, see *XmBulletinBoard*.

RETURN VALUE

Returns the BulletinBoard widget ID.

SEE ALSO

XmBulletinBoard.

NAME

XmCreateBulletinBoardDialog — the BulletinBoard BulletinBoardDialog convenience creation function

SYNOPSIS

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

Widget XmCreateBulletinBoardDialog(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateBulletinBoardDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged BulletinBoard child of the DialogShell. A BulletinBoardDialog is used for interactions not supported by the standard dialog set. This function does not automatically create any labels, buttons, or other dialog components. Such components should be added by the application after the BulletinBoardDialog is created.

Use *XtManageChild*( ) to pop up the BulletinBoardDialog (passing the BulletinBoard as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of BulletinBoard and its associated resources, see *XmBulletinBoard*.

RETURN VALUE

Returns the BulletinBoard widget ID.

SEE ALSO

XmBulletinBoard.

NAME

XmCreateCascadeButton — the CascadeButton widget creation function

SYNOPSIS

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

Widget XmCreateCascadeButton(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateCascadeButton( ) creates an instance of a CascadeButton widget and returns the associated widget ID.

parent Specifies the parent widget ID. The parent must be a RowColumn widget.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of CascadeButton and its associated resources, see *XmCascadeButton*.

RETURN VALUE

Returns the CascadeButton widget ID.

SEE ALSO

XmCascadeButton.

NAME

XmCreateCascadeButtonGadget — the CascadeButtonGadget creation function

SYNOPSIS

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

Widget XmCreateCascadeButtonGadget(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateCascadeButtonGadget( ) creates an instance of a CascadeButtonGadget and returns the associated widget ID.

parent Specifies the parent widget ID. The parent must be a RowColumn widget.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of CascadeButtonGadget and its associated resources, see *XmCascadeButtonGadget*.

RETURN VALUE

Returns the CascadeButtonGadget widget ID.

SEE ALSO

XmCascadeButtonGadget.

NAME

XmCreateCommand — the Command widget creation function

SYNOPSIS

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

Widget XmCreateCommand(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateCommand( ) creates an instance of a Command widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Command and its associated resources, see *XmCommand*.

RETURN VALUE

Returns the Command widget ID.

SEE ALSO

XmCommand.

NAME

XmCreateCommandDialog — a function to create a DialogShell and a Command child of the shell.

SYNOPSIS

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

Widget XmCreateCommandDialog(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateCommandDialog( ) creates an instance of a DialogShell widget and a Command child of the shell. This function returns the associated Command widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Command and its associated resources, see *XmCommand*.

RETURN VALUE

Returns the Command widget ID.

SEE ALSO

XmCommand.

NAME

XmCreateDialogShell — the DialogShell widget creation function

SYNOPSIS

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

Widget XmCreateDialogShell(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateDialogShell( ) creates an instance of a DialogShell widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of DialogShell and its associated resources, see *XmDialogShell*.

RETURN VALUE

Returns the DialogShell widget ID.

SEE ALSO

XmDialogShell.

NAME

XmCreateDragIcon — a Drag and Drop function that creates a DragIcon widget

SYNOPSIS

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

Widget XmCreateDragIcon(
      Widget                    widget,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateDragIcon( ) creates a DragIcon and returns the associated widget ID.

widget Specifies the ID of the widget that the function uses to access default values for visual attributes of the DragIcon. This widget may be different from the actual parent of the DragIcon.

name Specifies the name of the DragIcon widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of DragIcon and its associated resources, see *XmDragIcon*.

RETURN VALUE

The function creates a DragIcon and returns the associated widget ID.

SEE ALSO

XmDragContext, *XmDragIcon* and *XmScreen*.

NAME

XmCreateDrawingArea — the DrawingArea widget creation function

SYNOPSIS

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

Widget XmCreateDrawingArea(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateDrawingArea( ) creates an instance of a DrawingArea widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of DrawingArea and its associated resources, see *XmDrawingArea*.

RETURN VALUE

Returns the DrawingArea widget ID.

SEE ALSO

XmDrawingArea.

NAME

XmCreateDrawnButton — the DrawnButton widget creation function

SYNOPSIS

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

Widget XmCreateDrawnButton(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateDrawnButton( ) creates an instance of a DrawnButton widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of DrawnButton and its associated resources, see *XmDrawnButton*.

RETURN VALUE

Returns the DrawnButton widget ID.

SEE ALSO

XmDrawnButton.

NAME

XmCreateErrorDialog — the MessageBox ErrorDialog convenience creation function

SYNOPSIS

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

Widget XmCreateErrorDialog(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateErrorDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. An ErrorDialog warns the user of an invalid or potentially dangerous condition. It includes a symbol, a message and three buttons. The default symbol is an octagon with a diagonal slash. The default button labels are **OK**, **Cancel** and **Help**.

Use *XtManageChild*( ) to pop up the ErrorDialog (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCreateFileSelectionBox — the FileSelectionBox widget creation function

SYNOPSIS

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

Widget XmCreateFileSelectionBox(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateFileSelectionBox( ) creates an unmanaged FileSelectionBox. A FileSelectionBox is used to select a file and includes the following:

- an editable text field for the directory mask
- a scrolling list of filenames
- an editable text field for the selected file
- labels for the list and text fields
- four buttons

The default button labels are **OK**, **Filter**, **Cancel** and **Help**. One additional **WorkArea** child may be added to the FileSelectionBox after creation.

If the parent of the FileSelectionBox is a DialogShell, use *XtManageChild*( ) to pop up the FileSelectionDialog (passing the FileSelectionBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of FileSelectionBox and its associated resources, see *XmFileSelectionBox*.

RETURN VALUE

Returns the FileSelectionBox widget ID.

SEE ALSO

XmFileSelectionBox.

NAME

XmCreateFileSelectionDialog — the FileSelectionBox FileSelectionDialog convenience creation function

SYNOPSIS

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

Widget XmCreateFileSelectionDialog(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateFileSelectionDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged FileSelectionBox child of the DialogShell. A FileSelectionDialog selects a file. It includes the following:

- an editable text field for the directory mask
- a scrolling list of filenames
- an editable text field for the selected file
- labels for the list and text fields
- four buttons

The default button labels are **OK**, **Filter**, **Cancel** and **Help**. One additional **WorkArea** child may be added to the FileSelectionBox after creation.

Use *XtManageChild*( ) to pop up the FileSelectionDialog (passing the FileSelectionBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of FileSelectionBox and its associated resources, see *XmFileSelectionBox*.

RETURN VALUE

Returns the FileSelectionBox widget ID.

SEE ALSO

XmFileSelectionBox.

NAME

XmCreateForm — the Form widget creation function

SYNOPSIS

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

Widget XmCreateForm(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateForm( ) creates an instance of a Form widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Form and its associated resources, see *XmForm*.

RETURN VALUE

Returns the Form widget ID.

SEE ALSO

XmForm.

NAME

XmCreateFormDialog — a Form FormDialog convenience creation function

SYNOPSIS

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

Widget XmCreateFormDialog(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateFormDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged Form child of the DialogShell. A FormDialog is used for interactions not supported by the standard dialog set. This function does not automatically create any labels, buttons, or other dialog components. Such components should be added by the application after the FormDialog is created.

Use *XtManageChild*( ) to pop up the FormDialog (passing the Form as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Form and its associated resources, see *XmForm*.

RETURN VALUE

Returns the Form widget ID.

SEE ALSO

XmForm.

NAME

XmCreateFrame — the Frame widget creation function

SYNOPSIS

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

Widget XmCreateFrame(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateFrame( ) creates an instance of a Frame widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Frame and its associated resources, see *XmFrame*.

RETURN VALUE

Returns the Frame widget ID.

SEE ALSO

XmFrame.

NAME

XmCreateInformationDialog — the MessageBox InformationDialog convenience creation function

SYNOPSIS

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

Widget XmCreateInformationDialog(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateInformationDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. An InformationDialog gives the user information, such as the status of an action. It includes a symbol, a message and three buttons. The default symbol is **i**. The default button labels are **OK**, **Cancel** and **Help**.

Use *XtManageChild*( ) to pop up the InformationDialog (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCreateLabel — the Label widget creation function

SYNOPSIS

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

Widget XmCreateLabel(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateLabel( ) creates an instance of a Label widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Label and its associated resources, see *XmLabel*.

RETURN VALUE

Returns the Label widget ID.

SEE ALSO

XmLabel.

NAME

XmCreateLabelGadget**( ) — the LabelGadget creation function**

SYNOPSIS

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

Widget XmCreateLabelGadget(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateLabelGadget( ) creates an instance of a LabelGadget widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of LabelGadget and its associated resources, see *XmLabelGadget*.

RETURN VALUE

Returns the LabelGadget widget ID.

SEE ALSO

XmLabelGadget.

NAME

XmCreateList — the List widget creation function

SYNOPSIS

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

Widget XmCreateList(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateList( ) creates an instance of a List widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns the List widget ID.

SEE ALSO

XmList.

NAME

XmCreateMainWindow — the MainWindow widget creation function

SYNOPSIS

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

Widget XmCreateMainWindow(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateMainWindow( ) creates an instance of a MainWindow widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MainWindow and its associated resources, see *XmMainWindow*.

RETURN VALUE

Returns the MainWindow widget ID.

SEE ALSO

XmMainWindow.

NAME

XmCreateMenuBar — a RowColumn widget convenience creation function

SYNOPSIS

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

Widget XmCreateMenuBar(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateMenuBar( ) creates an instance of a RowColumn widget of type XmMENU_BAR and returns the associated widget ID. It is provided as a convenience function for creating RowColumn widgets configured to operate as a MenuBar and is not implemented as a separate widget class.

The MenuBar widget is generally used for building a Pulldown menu system. Typically, a MenuBar is created and placed along the top of the application window, and several CascadeButtons are inserted as the children. Each of the CascadeButtons has a Pulldown MenuPane associated with it. These Pulldown MenuPanes must have been created as children of the MenuBar. The user interacts with the MenuBar by using either the mouse or the keyboard.

The MenuBar displays a 3-D shadow along its border. The application controls the shadow attributes using the visual-related resources supported by *XmManager*.

The MenuBar widget is homogeneous in that it accepts only children that are a subclass of *XmCascadeButton* or *XmCascadeButtonGadget*. Attempting to insert a child of a different class results in a warning message.

If the MenuBar does not have enough room to fit all of its subwidgets on a single line, the MenuBar attempts to wrap the remaining entries onto additional lines if allowed by the geometry manager of the parent widget.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the RowColumn widget ID.

SEE ALSO

XmCascadeButton, *XmCascadeButtonGadget*, *XmCreatePulldownMenu*( ), *XmManager* and *XmRowColumn*.

NAME

XmCreateMenuShell — the MenuShell widget creation function

SYNOPSIS

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

Widget XmCreateMenuShell(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateMenuShell( ) creates an instance of a MenuShell widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MenuShell and its associated resources, see *XmMenuShell*.

RETURN VALUE

Returns the MenuShell widget ID.

SEE ALSO

XmMenuShell.

NAME

XmCreateMessageBox — the MessageBox widget creation function

SYNOPSIS

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

Widget XmCreateMessageBox(
      Widget                       parent,
      String                       name,
      ArgList                      arglist,
      Cardinal                     argcount);
```

DESCRIPTION

XmCreateMessageBox( ) creates an unmanaged MessageBox. A MessageBox is used for common interaction tasks, which include giving information, asking questions and reporting errors. It includes an optional symbol, a message and three buttons.

By default, there is no symbol. The default button labels are **OK**, **Cancel** and **Help**.

If the parent of the MessageBox is a DialogShell, use *XtManageChild*( ) to pop up the MessageBox (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCreateMessageDialog — the MessageBox MessageDialog convenience creation function

SYNOPSIS

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

Widget XmCreateMessageDialog(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateMessageDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A MessageDialog is used for common interaction tasks, which include giving information, asking questions and reporting errors. It includes a symbol, a message and three buttons. By default, there is no symbol. The default button labels are **OK**, **Cancel** and **Help**.

Use *XtManageChild*( ) to pop up the MessageDialog (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCreateOptionMenu — a RowColumn widget convenience creation function

SYNOPSIS

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

Widget XmCreateOptionMenu(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateOptionMenu( ) creates an instance of a RowColumn widget of type XmMENU_OPTION and returns the associated widget ID.

It is provided as a convenience function for creating a RowColumn widget configured to operate as an OptionMenu and is not implemented as a separate widget class.

The OptionMenu widget is a specialized RowColumn manager composed of a label, a selection area and a single Pulldown MenuPane. When an application creates an OptionMenu widget, it supplies the label string and the Pulldown MenuPane. In order for the operation to be successful, there must be a valid **XmNsubMenuId** resource set when this function is called. The LabelGadget and the selection area (a CascadeButtonGadget) are created by the OptionMenu.

The OptionMenu's Pulldown MenuPane must not contain any ToggleButtons or ToggleButtonGadgets. The results of including CascadeButtons or CascadeButtonGadgets in the OptionMenu's Pulldown MenuPane are undefined.

An OptionMenu is laid out with the label displayed on one side of the widget and the selection area on the other side. The selection area has a dual purpose; it displays the label of the last item selected from the associated Pulldown MenuPane, and it provides the means for posting the Pulldown MenuPane.

The OptionMenu typically does not display any 3-D visual symbols around itself or the internal LabelGadget. By default, the internal CascadeButtonGadget has a visible 3-D shadow. The application may change this by getting the CascadeButtonGadget ID using *XmOptionButtonGadget*( ), and then calling *XtSetValues* using the standard visual-related resources.

The Pulldown MenuPane is posted when the mouse pointer is moved over the selection area and a mouse button defined by OptionMenu's RowColumn parent is pressed. The Pulldown MenuPane is posted and positioned so that the last selected item is directly over the selection area. The mouse is then used to arm the desired menu item. When the mouse button is released, the armed menu item is selected and the label within the selection area is changed to match that of the selected item. By default, **BSelect** is used to interact with an OptionMenu.

The OptionMenu also operates with the keyboard interface mechanism. If the application has established a mnemonic with the OptionMenu, pressing <Alt> with the mnemonic causes the Pulldown MenuPane to be posted with traversal enabled. The standard traversal keys can then be used to move within the MenuPane. Pressing <Return> or typing a mnemonic or accelerator for one of the menu items selects that item.

An application may use the **XmNmenuHistory** resource to indicate which item in the Pulldown MenuPane should be treated as the current choice and have its label displayed in the selection area. By default, the first selectable item in the Pulldown MenuPane is used.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the RowColumn widget ID.

SEE ALSO

XmCascadeButtonGadget, *XmCreatePulldownMenu*( ), *XmLabelGadget*, *XmOptionButtonGadget*( ), *XmOptionLabelGadget*( ) and *XmRowColumn*.

NAME

XmCreatePanedWindow — the PanedWindow widget creation function

SYNOPSIS

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

Widget XmCreatePanedWindow(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreatePanedWindow( ) creates an instance of a PanedWindow widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of PanedWindow and its associated resources, see *XmPanedWindow*.

RETURN VALUE

Returns the PanedWindow widget ID.

SEE ALSO

XmPanedWindow.

NAME

XmCreatePopupMenu — a RowColumn widget convenience creation function

SYNOPSIS

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

Widget XmCreatePopupMenu(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreatePopupMenu( ) creates an instance of a RowColumn widget of type XmMENU_POPUP and returns the associated widget ID. When this function is used to create the Popup MenuPane, a MenuShell widget is automatically created as the parent of the MenuPane. The parent of the MenuShell widget is the widget indicated by the *parent* argument.

XmCreatePopupMenu( ) is provided as a convenience function for creating RowColumn widgets configured to operate as Popup MenuPanes and is not implemented as a separate widget class.

The PopupMenu is used as the first MenuPane within a PopupMenu system; all other MenuPanes are of the Pulldown type. A Popup MenuPane displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the MenuPane.

The Popup MenuPane must be created as the child of a MenuShell widget in order to function properly when it is incorporated into a menu. If the application uses this convenience function for creating a Popup MenuPane, the MenuShell is automatically created as the real parent of the MenuPane. If the application does not use this convenience function to create the RowColumn to function as a Popup MenuPane, it is the application's responsibility to create the MenuShell widget.

To access the PopupMenu, the application must first position the widget using the *XmMenuPosition*( ) function and then manage it using *XtManageChild*( ).

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

Popup MenuPanes support tear-off capabilities for tear-off menus through *XmRowColumn* resources. For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the RowColumn widget ID.

SEE ALSO

XmMenuPosition( ), *XmMenuShell* and *XmRowColumn*.

NAME

XmCreatePromptDialog — the SelectionBox PromptDialog convenience creation function

SYNOPSIS

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

Widget XmCreatePromptDialog(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreatePromptDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged SelectionBox child of the DialogShell. A PromptDialog prompts the user for text input. It includes a message, a text input region and three managed buttons. The default button labels are **OK**, **Cancel** and **Help**. An additional button, with **Apply** as the default label, is created unmanaged; it may be explicitly managed if needed. One additional **WorkArea** child may be added to the SelectionBox after creation.

XmCreatePromptDialog( ) forces the value of the SelectionBox resource **XmNdialogType** to XmDIALOG_PROMPT.

Use *XtManageChild*( ) to pop up the PromptDialog (passing the SelectionBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of SelectionBox and its associated resources, see *XmSelectionBox*.

RETURN VALUE

Returns the SelectionBox widget ID.

SEE ALSO

XmSelectionBox.

NAME

XmCreatePulldownMenu — a RowColumn widget convenience creation function

SYNOPSIS

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

Widget XmCreatePulldownMenu(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreatePulldownMenu( ) creates an instance of a RowColumn widget of type XmMENU_PULLDOWN and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

When this function is used to create the Pulldown MenuPane, a MenuShell widget is automatically created as the parent of the MenuPane. If the widget specified by the *parent* argument is a Popup or a Pulldown MenuPane, the MenuShell widget is created as a child of the *parent* MenuShell; otherwise, it is created as a child of the specified *parent* widget.

XmCreatePulldownMenu( ) is provided as a convenience function for creating RowColumn widgets configured to operate as Pulldown MenuPanes and is not implemented as a separate widget class.

A Pulldown MenuPane displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the MenuPane.

A Pulldown MenuPane is used with submenus that are to be attached to a CascadeButton or a CascadeButtonGadget. This is the case for all MenuPanes that are part of a PulldownMenu system (a MenuBar), the MenuPane associated with an OptionMenu and any MenuPanes that cascade from a Popup MenuPane. Pulldown MenuPanes that are to be associated with an OptionMenu must be created before the OptionMenu is created.

The Pulldown MenuPane must be attached to a CascadeButton or CascadeButtonGadget that resides in a MenuBar, a Popup MenuPane, a Pulldown MenuPane, or an OptionMenu. It is attached with the button resource **XmNsubMenuId**.

A MenuShell widget is required between the Pulldown MenuPane and its parent. If the application uses this convenience function for creating a Pulldown MenuPane, the MenuShell is automatically created as the real parent of the MenuPane; otherwise, it is the application's responsibility to create the MenuShell widget.

To function correctly when incorporated into a menu, the Pulldown MenuPane's hierarchy must be considered. This hierarchy depends on the type of menu system being built, as follows:

- If the Pulldown MenuPane is to be pulled down from a MenuBar, its parent must be the MenuBar.
- If the Pulldown MenuPane is to be pulled down from a Popup or another Pulldown MenuPane, its parent must be that Popup or Pulldown MenuPane.

- If the Pulldown MenuPane is to be pulled down from an OptionMenu, its parent must be the same as the OptionMenu parent.

PullDown MenuPanes support tear-off capabilities for tear-off menus through *XmRowColumn* resources. For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the RowColumn widget ID.

SEE ALSO

XmCascadeButton, *XmCascadeButtonGadget*, *XmCreateOptionMenu*( ), *XmCreatePopupMenu*( ), *XmMenuShell* and *XmRowColumn*.

NAME

XmCreatePushButton — the PushButton widget creation function

SYNOPSIS

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

Widget XmCreatePushButton(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreatePushButton( ) creates an instance of a PushButton widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of PushButton and its associated resources, see *XmPushButton*.

RETURN VALUE

Returns the PushButton widget ID.

SEE ALSO

XmPushButton.

NAME

XmCreatePushButtonGadget — the PushButtonGadget creation function

SYNOPSIS

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

Widget XmCreatePushButtonGadget(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreatePushButtonGadget( ) creates an instance of a PushButtonGadget widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of PushButtonGadget and its associated resources, see *XmPushButtonGadget*.

RETURN VALUE

Returns the PushButtonGadget widget ID.

SEE ALSO

XmPushButtonGadget.

NAME

XmCreateQuestionDialog — the MessageBox QuestionDialog convenience creation function

SYNOPSIS

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

Widget XmCreateQuestionDialog(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateQuestionDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A QuestionDialog is used to get the answer to a question from the user. It includes a symbol, a message and three buttons. The default symbol is a question mark. The default button labels are **OK**, **Cancel** and **Help**.

Use *XtManageChild*( ) to pop up the QuestionDialog (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCreateRadioBox — a RowColumn widget convenience creation function

SYNOPSIS

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

Widget XmCreateRadioBox(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateRadioBox( ) creates an instance of a RowColumn widget of type XmWORK_AREA and returns the associated widget ID. Typically, this is a composite widget that contains multiple ToggleButtonGadgets. The RadioBox arbitrates and ensures that at most one ToggleButtonGadget is on at any time.

Unless the application supplies other values in the *arglist*, this function provides initial values for several RowColumn resources. It initializes **XmNpacking** to XmPACK_COLUMN, **XmNradioBehavior** to True, **XmNisHomogeneous** to True and **XmNentryClass** to **XmToggleButtonGadgetClass**.

In a RadioBox, the ToggleButton or ToggleButtonGadget resource **XmNindicatorType** defaults to XmONE_OF_MANY, and the ToggleButton or ToggleButtonGadget resource**XmNvisibleWhenOff** defaults to True.

This routine is provided as a convenience function for creating RowColumn widgets.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the RowColumn widget ID.

SEE ALSO

XmCreateRowColumn( ), *XmCreateWorkArea*( ) and *XmRowColumn*.

NAME

XmCreateRowColumn — the RowColumn widget creation function

SYNOPSIS

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

Widget XmCreateRowColumn(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateRowColumn( ) creates an instance of a RowColumn widget and returns the associated widget ID. If **XmNrowColumnType** is not specified, then it is created with XmWORK_AREA, which is the default.

If this function is used to create a Popup Menu of type XmMENU_POPUP or a Pulldown Menu of type XmMENU_PULLDOWN, a MenuShell widget is not automatically created as the parent of the MenuPane. The application must first create the MenuShell by using either *XmCreateMenuShell*( ) or the standard toolkit create function.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the RowColumn widget ID.

SEE ALSO

XmCreateMenuBar( ), *XmCreateMenuShell*( ), *XmCreateOptionMenu*( ), *XmCreatePopupMenu*( ), *XmCreatePulldownMenu*( ), *XmCreateRadioBox*( ), *XmCreateWorkArea*( ) and *XmRowColumn*.

NAME

XmCreateScale — the Scale widget creation function

SYNOPSIS

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

Widget XmCreateScale(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateScale( ) creates an instance of a Scale widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Scale and its associated resources, see *XmScale*.

RETURN VALUE

Returns the Scale widget ID.

SEE ALSO

XmScale.

NAME

XmCreateScrollBar — the ScrollBar widget creation function

SYNOPSIS

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

Widget XmCreateScrollBar(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateScrollBar( ) creates an instance of a ScrollBar widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of ScrollBar and its associated resources, see *XmScrollBar*.

RETURN VALUE

Returns the ScrollBar widget ID.

SEE ALSO

XmScrollBar.

NAME

XmCreateScrolledList — the List ScrolledList convenience creation function

SYNOPSIS

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

Widget XmCreateScrolledList(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateScrolledList( ) creates an instance of a List widget that is contained within a ScrolledWindow. The ScrolledWindow parent is created managed. All ScrolledWindow subarea widgets are automatically created by this function. The ID returned by this function is that of the List widget. Use this widget ID for all operations on the List widget. Use the widget ID of the List widget's parent for all operations on the ScrolledWindow. To obtain the ID of the ScrolledWindow widget associated with the List widget, use the Xt Intrinsics *XtParent*( ) function. The name of the ScrolledWindow created by this function is formed by concatenating **SW** onto the end of the *name* specified in the argument list.

All arguments to either the List or the ScrolledWindow widget can be specified at creation time using this function. Changes to initial position and size are sent only to the ScrolledWindow widget. Other resources are sent to the List or the ScrolledWindow widget as appropriate. Changes to initial position and size are sent only to the ScrolledWindow widget. Other resources are sent to the List or the ScrolledWindow widget as appropriate.

This function forces the following initial values for ScrolledWindow resources:

- **XmNscrollingPolicy** is set to XmAPPLICATION_DEFINED.
- **XmNvisualPolicy** is set to XmVARIABLE.
- **XmNscrollBarDisplayPolicy** is set to XmSTATIC. (No initial value is forced for the Headers/List.inc **XmNscrollBarDisplayPolicy**.)
- **XmNshadowThickness** is set to 0 (zero).

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns the List widget ID.

SEE ALSO

XmList and *XmScrolledWindow*.

NAME

XmCreateScrolledText — the Text ScrolledText convenience creation function

SYNOPSIS

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

Widget XmCreateScrolledText(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateScrolledText( ) creates an instance of a Text widget that is contained within a ScrolledWindow. All ScrolledWindow subarea widgets are automatically created by this function. The ID returned by this function is that of the Text widget. Use this ID for all normal Text operations, as well as those that are relevant for the ScrolledText widget.

The Text widget defaults to single-line text edit; therefore, no ScrollBars are displayed. The Text resource **XmNeditMode** must be set to XmMULTI_LINE_EDIT to display the ScrollBars. The results of placing a Text widget inside a ScrolledWindow when the Text's **XmNeditMode** is XmSINGLE_LINE_EDIT are undefined.

All arguments to either the Text or the ScrolledWindow widget can be specified at creation time with this function. Changes to initial position and size are sent only to the ScrolledWindow widget. Other resources are sent to the Text or the ScrolledWindow widget as appropriate.

This function forces the following initial values for ScrolledWindow resources:

- **XmNscrollingPolicy** is set to XmAPPLICATION_DEFINED.
- **XmNvisualPolicy** is set to XmVARIABLE.
- **XmNscrollBarDisplayPolicy** is set to XmSTATIC.
- **XmNshadowThickness** is set to 0 (zero).

To obtain the ID of the ScrolledWindow widget associated with the ScrolledText, use the Xt Intrinsics *XtParent* function. The name of the ScrolledWindow created by this function is formed by concatenating the letters SW onto the end of the *name* specified in the argument list.

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns the Text widget ID.

SEE ALSO

XmScrolledWindow and *XmText*.

NAME

XmCreateScrolledWindow — the ScrolledWindow widget creation function

SYNOPSIS

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

Widget XmCreateScrolledWindow(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateScrolledWindow( ) creates an instance of a ScrolledWindow widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of ScrolledWindow and its associated resources, see *XmScrolledWindow*.

RETURN VALUE

Returns the ScrolledWindow widget ID.

SEE ALSO

XmScrolledWindow.

NAME

XmCreateSelectionBox — the SelectionBox widget creation function

SYNOPSIS

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

Widget XmCreateSelectionBox(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateSelectionBox( ) creates an unmanaged SelectionBox. A SelectionBox is used to get a selection from a list of alternatives from the user and includes the following:

- A scrolling list of alternatives
- An editable text field for the selected alternative
- Labels for the list and text field
- Three or four buttons.

The default button labels are **OK**, **Cancel** and **Help**. By default, an **Apply** button is also created. If the parent of the SelectionBox is a DialogShell, it is managed; otherwise it is unmanaged. One additional **WorkArea** child may be added to the SelectionBox after creation.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of SelectionBox and its associated resources, see *XmSelectionBox*.

RETURN VALUE

Returns the SelectionBox widget ID.

SEE ALSO

XmSelectionBox.

NAME

XmCreateSelectionDialog — the SelectionBox SelectionDialog convenience creation function

SYNOPSIS

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

Widget XmCreateSelectionDialog(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateSelectionDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged SelectionBox child of the DialogShell. A SelectionDialog offers the user a choice from a list of alternatives and gets a selection. It includes the following:

- a scrolling list of alternatives
- an editable text field for the selected alternative
- labels for the text field
- four buttons.

The default button labels are **OK**, **Cancel**, **Apply** and **Help**. One additional **WorkArea** child may be added to the SelectionBox after creation.

XmCreateSelectionDialog( ) forces the value of the SelectionBox resource **XmNdialogType** to XmDIALOG_SELECTION.

Use *XtManageChild*( ) to pop up the SelectionDialog (passing the SelectionBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of SelectionBox and its associated resources, see *XmSelectionBox*.

RETURN VALUE

Returns the SelectionBox widget ID.

SEE ALSO

XmSelectionBox.

NAME

XmCreateSeparator — the Separator widget creation function

SYNOPSIS

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

Widget XmCreateSeparator(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateSeparator( ) creates an instance of a Separator widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Separator and its associated resources, see *XmSeparator*.

RETURN VALUE

Returns the Separator widget ID.

SEE ALSO

XmSeparator.

NAME

XmCreateSeparatorGadget — the SeparatorGadget creation function

SYNOPSIS

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

Widget XmCreateSeparatorGadget(
      Widget                     parent,
      String                     name,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmCreateSeparatorGadget( ) creates an instance of a SeparatorGadget widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of SeparatorGadget and its associated resources, see *XmSeparatorGadget*.

RETURN VALUE

Returns the SeparatorGadget widget ID.

SEE ALSO

XmSeparatorGadget.

NAME

XmCreateTemplateDialog — a MessageBox TemplateDialog convenience creation function

SYNOPSIS

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

Widget XmCreateTemplateDialog(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateTemplateDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. The MessageBox widget's **XmNdialogType** resource is set to XmDIALOG_TEMPLATE. By default, the TemplateDialog widget contains only the separator child. You can build a customized dialog by adding children to the TemplateDialog.

You can create the standard MessageBox pushbuttons, **Cancel**, **Help**, and **OK**, by specifying the associated callback and label string resources. Setting **XmNsymbolPixmap** or **XmNmessageString** creates a symbol or message label.

Use *XtManageChild*( ) to pop up the TemplateDialog (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID .

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCreateText — the Text widget creation function

SYNOPSIS

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

Widget XmCreateText(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateText( ) creates an instance of a Text widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns the Text widget ID.

SEE ALSO

XmText.

NAME

XmCreateTextField — the TextField widget creation function

SYNOPSIS

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

Widget XmCreateTextField(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateTextField( ) creates an instance of a TextField widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns the TextField widget ID.

SEE ALSO

XmTextField.

NAME

XmCreateToggleButton — the ToggleButton widget creation function

SYNOPSIS

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

Widget XmCreateToggleButton(
      Widget                      parent,
      String                      name,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmCreateToggleButton( ) creates an instance of a ToggleButton widget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of ToggleButton and its associated resources, see *XmToggleButton*.

RETURN VALUE

Returns the ToggleButton widget ID.

SEE ALSO

XmToggleButton.

NAME

XmCreateToggleButtonGadget — the ToggleButtonGadget creation function

SYNOPSIS

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

Widget XmCreateToggleButtonGadget(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateToggleButtonGadget( ) creates an instance of a ToggleButtonGadget and returns the associated widget ID.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of ToggleButtonGadget and its associated resources, see *XmToggleButtonGadget*.

RETURN VALUE

Returns the ToggleButtonGadget widget ID.

SEE ALSO

XmToggleButtonGadget.

NAME

XmCreateWarningDialog — the MessageBox WarningDialog convenience creation function

SYNOPSIS

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

Widget XmCreateWarningDialog(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateWarningDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A WarningDialog warns users of action consequences and gives them a choice of resolutions. It includes a symbol, a message and three buttons. The default symbol is an exclamation point. The default button labels are **OK**, **Cancel** and **Help**.

Use *XtManageChild*( ) to pop up the WarningDialog (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCreateWorkArea — a function that creates a RowColumn WorkArea

SYNOPSIS

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

Widget XmCreateWorkArea(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateWorkArea( ) creates an instance of a RowColumn widget and returns the associated widget ID. The widget is created with **XmNrowColumnType** set to XmWORK_AREA.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the RowColumn widget ID.

SEE ALSO

XmCreateRadioBox( ), *XmCreateRowColumn*( ) and *XmRowColumn*.

NAME

XmCreateWorkingDialog — the MessageBox WorkingDialog convenience creation function

SYNOPSIS

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

Widget XmCreateWorkingDialog(
      Widget                    parent,
      String                    name,
      ArgList                   arglist,
      Cardinal                  argcount);
```

DESCRIPTION

XmCreateWorkingDialog( ) is a convenience creation function that creates a DialogShell and an unmanaged MessageBox child of the DialogShell. A WorkingDialog informs users that there is a time-consuming operation in progress and allows them to cancel the operation. It includes a symbol, a message and three buttons. The default symbol is an hourglass. The default button labels are **OK**, **Cancel** and **Help**.

Use *XtManageChild*( ) to pop up the WorkingDialog (passing the MessageBox as the widget argument); use *XtUnmanageChild*( ) to pop it down.

parent Specifies the parent widget ID.

name Specifies the name of the created widget.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the MessageBox widget ID.

SEE ALSO

XmMessageBox.

NAME

XmCvtCTToXmString — a compound string function that converts compound text to a compound string

SYNOPSIS

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

XmString XmCvtCTToXmString(
      char                    *text);
```

DESCRIPTION

XmCvtCTToXmString( ) converts a **char *** string in compound text format to a compound string. The application must call *XtAppInitialize*( ) before calling this function.

text Specifies a string in compound text format to be converted to a compound string.

RETURN VALUE

Returns a compound string derived from the compound text. The compound text is assumed to be NULL-terminated; NULLs within the compound text are handled correctly. The handling of HORIZONTAL TABULATION (HT) control characters within the compound text is undefined. The compound text format is described in the **Compound Text** specification.

SEE ALSO

XmCvtXmStringToCT( ).

NAME

XmCvtXmStringToCT — a compound string function that converts a compound string to compound text

SYNOPSIS

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

char *XmCvtXmStringToCT(
     XmString                  string);
```

DESCRIPTION

XmCvtXmStringToCT( ) converts a compound string to a **char *** string in compound text format. The application must call *XtAppInitialize*( ) before calling this function. The converter uses the font list tag associated with a given compound string segment to select a compound text format for that segment. A registry defines a mapping between font list tags and compound text encoding formats. The converter uses the following algorithm for each compound string segment:

1. If the compound string segment tag is mapped to XmFONTLIST_DEFAULT_TAG in the registry, the converter passes the text of the compound string segment to *XmbTextListToTextProperty*( )[2] with an encoding style of **XCompoundTextStyle** and uses the resulting compound text for that segment.

2. If the compound string segment tag is mapped to an MIT registered charset in the registry, the converter creates the compound text for that segment using the charset (from the registry) and the text of the compound string segment as defined in the **Compound Text** specification.

3. If the compound string segment tag is mapped to a charset in the registry that is neither XmFONTLIST_DEFAULT_TAG nor an MIT registered charset, the converter creates the compound text for that segment using the charset (from the registry) and the text of the compound string segment as an *extended segment* with a variable number of octets per character.

4. If the compound string segment tag is not mapped in the registry, the result is implementation dependent.

string Specifies a compound string to be converted to compound text.

RETURN VALUE

Returns a **char *** string in compound text format. This format is described in the **Compound Text** specification.

SEE ALSO

XmCvtCTToXmString( ), **XmFontList** and **XmString**.

[2] An X Windows function.

NAME

XmDeactivateProtocol — a VendorShell function that deactivates a protocol without removing it

SYNOPSIS

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

void XmDeactivateProtocol(
      Widget                    shell,
      Atom                      property,
      Atom                      protocol);
```

DESCRIPTION

XmDeactivateProtocol( ) deactivates a protocol without removing it. It updates the handlers and the *property* if the *shell* is realized. It is sometimes useful to allow a protocol's state information (callback lists and so on) to persist, even though the client may choose to temporarily resign from the interaction. The main use of this capability is to gray or ungray *f.send_msg* entries in the MWM system menu. To support this capability, *protocol* is allowed to be in one of two states: active or inactive. If *protocol* is active and *shell* is realized, *property* contains the *protocol* **Atom**. If *protocol* is inactive, **Atom** is not present in the *property*.

XmDeactivateWMProtocol( ) is a convenience interface. It calls *XmDeactivateProtocol*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

property Specifies the protocol property.

protocol Specifies the protocol atom (or an int type cast to **Atom**).

For a complete definition of VendorShell and its associated resources, see *VendorShell*.

SEE ALSO

mwm, *VendorShell*, *XmActivateProtocol*( ), *XmDeactivateWMProtocol*( ) and *XmInternAtom*( ).

NAME

XmDeactivateWMProtocol — a VendorShell convenience interface that deactivates a protocol without removing it

SYNOPSIS

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

void XmDeactivateWMProtocol(
      Widget                     shell,
      Atom                       protocol);
```

DESCRIPTION

XmDeactivateWMProtocol( ) is a convenience interface. It calls *XmDeactivateProtocol*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

protocol Specifies the protocol atom (or an **int** type cast to **Atom**).

For a complete definition of VendorShell and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmActivateWMProtocol*( ), *XmDeCommon/activate.incotocol*( ) and *XmInternAtom*( ).

NAME

XmDestroyPixmap — a pixmap function that removes a pixmap from the pixmap cache

SYNOPSIS

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

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

DESCRIPTION

XmDestroyPixmap( ) removes pixmaps that are no longer used. Pixmaps are completely freed only when there is no further reference to them.

screen Specifies the display screen for which the pixmap was requested.

pixmap Specifies the pixmap to be destroyed.

RETURN VALUE

Returns True when successful; returns False if there is no matching screen and pixmap in the pixmap cache.

SEE ALSO

XmInstallImage( ), *XmUninstallImage*( ) and *XmFontListEntryGetTag*( ).

NAME

XmDialogShell — the DialogShell widget class

SYNOPSIS

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

DESCRIPTION

Modal and modeless dialogs use DialogShell as the Shell parent. DialogShell widgets cannot be iconified. Instead, all secondary DialogShell widgets associated with an ApplicationShell widget are iconified and de-iconified as a group with the primary widget.

The client indirectly manipulates DialogShell through the convenience interfaces during creation and it can directly manipulate its BulletinBoard-derived child. Much of the functionality of DialogShell assumes that its child is a BulletinBoard subclass, although it can potentially stand alone.

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

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

Note that the **ICCCM** specification allows a window manager to change or control the border width of a reparented top-level window.

Classes

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

The class pointer is **xmDialogShellWidgetClass**.

The class name is *XmDialogShell*.

New Resources

DialogShell defines no new resources but overrides the **XmNdeleteResponse** resource in the *VendorShell* class.

Inherited Resources

DialogShell inherits behavior and resources from the superclasses described in the following tables, which define sets of widget resources used by the programmer to specify data.

For a complete description of each resource, refer to the reference page for that superclass. The 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).

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
XmNdefaultFontList	**XmCDefaultFontList**	**XmFontList**	dynamic	CG
XmNdeleteResponse	**XmCDeleteResponse**	**unsigned char**	XmDESTROY	CSG
XmNkeyboardFocusPolicy	**XmCKeyboardFocusPolicy**	**unsigned char**	XmEXPLICIT	CSG
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG
XmNmwmDecorations	**XmCMwmDecorations**	**int**	−1	CG
XmNmwmFunctions	**XmCMwmFunctions**	**int**	−1	CG
XmNmwmInputMode	**XmCMwmInputMode**	**int**	−1	CG
XmNmwmMenu	**XmCMwmMenu**	**String**	NULL	CG
XmNtextFontList	**XmCTextFontList**	**XmFontList**	dynamic	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**	−1	CSG
XmNiconY	**XmCIconY**	**int**	−1	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
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

SEE ALSO

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

NAME

XmDisplay — the Display widget class

SYNOPSIS

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

DESCRIPTION

The Display 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 Display object for each display it accesses.

A Display 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 a Display object by any other means. An application can use the function *XmGetXmDisplay*( ) to obtain the widget ID of the Display object for a given display.

An application cannot supply initial values for Display 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 Display object and then call *XtSetValues* to set the Display resources.

Display 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 communicate drag and drop state information dynamically between each other, and to update their respective visual symbols 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 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).

XmDisplay **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNdefaultVirtualBindings	**DefaultVirtualBindings**	**String**	dynamic	C
XmNdragInitiatorProtocolStyle	**XmCDragInitiatorProtocolStyle**	**unsigned char**	XmDRAG_PREFER_RECEIVER	CG
XmNdragReceiverProtocolStyle	**XmCDragReceiverProtocolStyle**	**unsigned char**	XmDRAG_PREFER_PREREGISTER	CG

XmNdefaultVirtualBindings

Specifies the default virtual bindings for the display. The 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; 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 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 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 defers 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 to accommodate performance needs.

The actual protocol used between an initiator and a receiver is based on the protocol style of the receiver and initiator. The decision matrix is described in the following table.

Drag Initiator Protocol Style	**Drag Receiver Protocol Style**			
	Preregister	Prefer Preregister	Prefer Dynamic	Dynamic
Preregister	Preregister	Preregister	Preregister	Drop Only
Prefer Preregister	Preregister	Preregister	Preregister	Dynamic
Prefer Receiver	Preregister	Preregister	Dynamic	Dynamic
Prefer Dynamic	Preregister	Dynamic	Dynamic	Dynamic
Dynamic	Drop Only	Dynamic	Dynamic	Dynamic

The value XmDRAG_NONE does not appear in the matrix. When specified for either the initiator or receiver side, XmDRAG_NONE implies that drag and drop transactions are not supported. A value of XmDRAG_DROP_ONLY (Drop Only) results when an initiator and receiver cannot compromise protocol styles. This occurs if one client requires dynamic mode while the other can only support preregister mode, or if either explicitly specified XmDRAG_DROP_ONLY.

Inherited Resources

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

SEE ALSO

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

NAME

XmDragCancel — a Drag and Drop function that terminates a drag transaction

SYNOPSIS

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

void XmDragCancel(
      Widget                    dragcontext);
```

DESCRIPTION

XmDragCancel( ) terminates a drag operation and cancels any pending actions of the specified DragContext. This routine can only be called by the initiator client.

dragcontext Specifies the ID of the DragContext widget associated with the drag and drop transaction to be terminated.

For a complete definition of DragContext and its associated resources, see *XmDragContext*.

SEE ALSO

XmDragContext and *XmDragStart*( ).

NAME

XmDragContext — the DragContext widget class

SYNOPSIS

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

DESCRIPTION

DragContexts are special widgets used in drag and drop transactions. A DragContext is implemented as a widget, but a client does not explicitly create a DragContext widget. Instead, a client initiates a drag and drop transaction by calling *XmDragStart*( ), and this routine initializes and returns a DragContext widget. There is a unique DragContext for each drag operation. The toolkit frees a DragContext when a transaction is complete; therefore, an application programmer should not explicitly destroy a DragContext.

Initiator and receiver clients both use DragContexts to track the state of a transaction. When the initiator and receiver of a transaction are in the same client, they share the same DragContext instance. If they are in different clients, there are two separate DragContexts. In this case, the initiator calls *XmDragStart*( ) and the toolkit provides a DragContext for the receiver client. The only resources pertinent to the receiver are **XmNexportTargets** and **XmNnumExportTargets**. These can both be passed as arguments to the *XmDropSiteRetrieve*( ) function to obtain information about the current drop site.

In general, in order to receive data, a drop site must share at least one target type and operation in common with a drag source. The DragContext resource, **XmNexportTargets**, identifies the selection targets for the drag source. These export targets are compared with the **XmNimportTargets** resource list specified by a drop site. The DragConCommon / text.incesource, **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 symbol. 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 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).

XmDragContext **Resource Set**

Name	**Class**	**Type**	**Default**	**Access**
XmNblendModel	**XmCBlendModel**	**unsigned char**	XmBLEND_ALL	CG
XmNclientData	**XmCClientData**	**XtPointer**	NULL	CSG
XmNconvertProc	**XmCConvertProc**	**XtConvert**	NULL Selection IncrProc	CSG
XmNcursorBackground	**XmCCursorBackground**	**Pixel**	dynamic	CSG
XmNcursorForeground	**XmCCursorForeground**	**Pixel**	dynamic	CSG
XmNdragDropFinishCallback	**XmCCallback**	**XtCallbackList**	NULL	CSG
XmNdragMotionCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNdragOperations	**XmCDragOperations**	**unsigned char**	XmDROP_COPY | XmDROP_MOVE	C
XmNdropFinishCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNdropSiteEnterCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNdropSiteLeaveCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNdropStartCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNexportTargets	**XmCExportTargets**	**Atom ***	NULL	CSG
XmNincremental	**XmCIncremental**	**Boolean**	False	CSG
XmNinvalidCursorForeground	**XmCCursorForeground**	**Pixel**	dynamic	CSG
XmNnoneCursorForeground	**XmCCursorForeground**	**Pixel**	dynamic	CSG
XmNnumExportTargets	**XmCNumExportTargets**	**Cardinal**	0	CSG
XmNoperationChangedCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNoperationCursorIcon	**XmCOperationCursorIcon**	**Widget**	dynamic	CSG
Headers/Xm.incourceCursorIcon	**XmCSourceCursorIcon**	**Widget**	dynamic	CSG
Headers/Xm.incourcePixmapIcon	**XmCSourcePixmapIcon**	**Widget**	dynamic	CSG
Headers/Xm.inctateCursorIcon	**XmCStateCursorIcon**	**Widget**	dynamic	CSG
XmNtopLevelEnterCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNtopLevelLeaveCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNvalidCursorForeground	**XmCCursorForeground**	**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 symbol is generated. The client tracks the drop site status through callback routines and updates the drag-over visual symbols as necessary.

XmNclientData
Specifies the client data to be passed to **XmNconvertProc** when it is invoked.

XmNconvertProc
Specifies a procedure of type *XtConvertSelectionIncrProc*( ) that converts the source data to the formats requested by the receiver client. The *widget* argument passed to this procedure is the DragContext widget. The selection atom passed is _MOTIF_DROP. If **XmNincremental** is False, the procedure should ignore the *Xmax_length*, *client_data* and *request_id* arguments and should handle the conversion atomically. Data returned by **XmNconvertProc** must be allocated using *XtMalloc*( ), and is freed automatically by the toolkit after the transfer. For additional information on selection conversion procedures, see the **ICCCM** specification.

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 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 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 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 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 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 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 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 the **ICCCM** specification. 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 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 symbol 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 symbol 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 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 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
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

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 is 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 symbol transition effects are 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 is 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.

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.

SEE ALSO

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

NAME

XmDragIcon — the DragIcon widget class

SYNOPSIS

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

DESCRIPTION

A DragIcon is a component of the visual symbol 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 visual symbols consisting of a static portion that represents the object being dragged, and dynamic cues that provide visual symbol feedback during the drag operation. The visual symbol is attained by blending together various *XmDragIcons* specified in the *XmDragContext* associated with the drag operation.

The static portion of the drag-over visual symbol 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 symbol. The *XmDragContext* resources, **XmNsourceCursorIcon** or **XmNsourcePixmapIcon**, specify a DragIcon to use for the static portion of the visual symbol.

A drag-over visual symbol incorporates dynamic cues in order to provide visual symbol feedback in response to the user's actions. For instance, the drag-over visual symbol 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 symbol 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 symbol. 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 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).

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 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

SEE ALSO

Object, *XmCreateDragIcon*, *XmDisplay*, *XmDragContext*, *XmDropSite*, *XmDropTransfer* and *XmScreen*.

NAME

XmDragStart — a Drag and Drop function that initiates a drag and drop transaction

SYNOPSIS

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

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

DESCRIPTION

XmDragStart( ) initiates a drag operation. This routine returns the DragContext widget that it initializes for the associated drag transaction. The toolkit is responsible for freeing the DragContext when the drag and drop transaction is complete.

widget Specifies the ID of the smallest widget or gadget that encloses the source elements selected for a drag operation.

event Specifies the **XEvent** that triggered the drag operation. This event must be a ButtonPress event.

arglist Specifies the argument list. Any *XmDragContext* resources not specified in the argument list are obtained from the resource database or are set to their default values.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*)

For a complete definition of DragContext and its associated resources, see *XmDragContext*.

RETURN VALUE

Returns the ID of the DragContext widget that controls this drag and drop transaction. Returns NULL if the drag cannot be initiated.

SEE ALSO

XmDragCancel( ) and *XmDragContext*.

NAME

XmDrawingArea — the DrawingArea widget class

SYNOPSIS

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

DESCRIPTION

DrawingArea is an empty widget that is easily adaptable to a variety of purposes. It does no drawing and defines no behavior except for invoking callbacks. Callbacks notify the application when graphics need to be drawn (exposure events or widget resize) and when the widget receives input from the keyboard or mouse.

Applications are responsible for defining appearance and behavior as needed in response to DrawingArea callbacks.

DrawingArea is also a composite widget and subclass of *XmManager* that supports minimal geometry management for multiple widget or gadget children.

Classes

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

The class pointer is **xmDrawingAreaWidgetClass**.

The class name is *XmDrawingArea*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmDrawingArea **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
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

XmNexposeCallback

Specifies the list of callbacks called when DrawingArea receives an exposure event. The callback reason is XmCR_EXPOSE. The callback structure also includes the exposure event.

XmNinputCallback

Specifies the list of callbacks 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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 each callback:

```
typedef struct
{
      int                        reason;
      XEvent                    *event;
      Window                     window;
} XmDrawingAreaCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback. This is NULL for the **XmNresizeCallback**.

window Is set to the widget window.

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.

SEE ALSO

Composite, *Constraint*, *Core*, *XmCreateDrawingArea*( ) and *XmManager*.

NAME

XmDrawnButton — the DrawnButton widget class

SYNOPSIS

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

DESCRIPTION

The DrawnButton widget consists of an empty widget window surrounded by a shadow border. It provides the application developer with a graphics area that can have PushButton input semantics.

Callback types are defined for widget exposure and widget resize to allow the application to redraw or reposition its graphics. If the DrawnButton widget has a highlight and shadow thickness, the application should not draw in that area. To avoid drawing in the highlight and shadow area, create the graphics context with a clipping rectangle for drawing in the widget. The clipping rectangle should take into account the size of the widget's highlight thickness and shadow.

Classes

DrawnButton inherits behavior and resources from the *Core*, *XmPrimitive* and *XmLabel* classes.

The class pointer is **xmDrawnButtonWidgetClass**.

The class name is *XmDrawnButton*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
XmNpushButtonEnabled	**XmCPushButtonEnabled**	**Boolean**	False	CSG
XmNresizeCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNshadowType	**XmCShadowType**	**unsigned char**	XmSHADOW_ETCHED_IN	CSG

XmNactivateCallback

Specifies the list of callbacks called when the widget becomes selected. The reason sent by the callback is XmCR_ACTIVATE.

XmNarmCallback

Specifies the list of callbacks called when the widget becomes armed. The reason sent by the callback is XmCR_ARM.

XmNdisarmCallback

Specifies the list of callbacks called when the widget becomes disarmed. The reason sent by the callback is XmCR_DISARM.

XmNexposeCallback

Specifies the list of callbacks 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 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 visual symbols and top shadow visual symbols 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
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	XmFONTLIST_DEFAULT_TAG	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CSG

XmPrimitive **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED_PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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.

Action Routines

The *XmDrawnButton* action routines are:

Activate( )
If **XmNpushButtonEnabled** is True, redraws the shadow in the unselected state; otherwise, redraws the shadow according to **XmNshadowType**. If the pointer is within the DrawnButton, calls the **XmNactivateCallback** callbacks.

Arm( )
If **XmNpushButtonEnabled** is True, redraws the shadow in the selected state; otherwise, redraws the shadow according to **XmNshadowType**. Calls the callbacks for **XmNarmCallback**.

ArmAndActivate( )
If **XmNpushButtonEnabled** is True, redraws the shadow in the selected state; otherwise, redraws the shadow according to **XmNshadowType**. Calls the callbacks for **XmNarmCallback**.

If **XmNpushButtonEnabled** is True, the shadow is redrawn in the unselected state; otherwise, the shadow is redrawn according to **XmNshadowType**. The callbacks for **XmNactivateCallback** and **XmNdisarmCallback** are called. These actions happen either immediately or at a later time.

Disarm( )
Marks the DrawnButton as unselected and calls the callbacks for **XmNdisarmCallback**.

Help( )
Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

MultiActivate( )
If **XmNmultiClick** is XmMULTICLICK_DISCARD, this action does nothing.

If **XmNmultiClick** is XmMULTICLICK_KEEP, this action increments **click_count** in the callback structure. If **XmNpushButtonEnabled** is True, this action redraws the shadow in the unselected state; otherwise, it redraws the shadow according to **XmNshadowType**. If the pointer is within the DrawnButton, this action calls the **XmNactivateCallback** callbacks and calls the callbacks for **XmNdisarmCallback**.

MultiArm( )
If **XmNmultiClick** is XmMULTICLICK_DISCARD, this action does nothing.

If **XmNmultiClick** is XmMULTICLICK_KEEP and if **XmNpushButtonEnabled** is True, this action redraws the shadow in the selected state; otherwise, it redraws the shadow according to **XmNshadowType** and calls the callbacks for **XmNarmCallback**.

SEE ALSO

Core, *XmCreateDrawnButton*( ), *XmLabel*, *XmPrimitive*, *XmPushButton* and *XmSeparator*.

NAME

XmDropSite — the DropSite Registry

SYNOPSIS

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

DESCRIPTION

A client registers a widget or gadget as a drop site using the *XmDropSiteRegister*( ) function. In addition, this routine defines the behavior and capabilities of a drop site by specifying appropriate resources. For example, the **XmNimportTargets** and **XmNnumImportTargets** resources identify respectively the selection target types and number of types supported by a drop site. The visual animation effects associated with a drop site are also described with DropSite resources.

Drop site animation effects that occur in response to the pointer entering a valid drop site are called drag-under effects. A receiver can select from several animation styles supplied by the toolkit or can provide customized animation effects. Drag-under effects supplied by the toolkit include border highlighting, shadow in or out drawing, and pixmap representation.

When a preregister drag protocol style is used, the toolkit generates drag-under visual effects based on the value of the **XmNanimationStyle** resource. In dynamic mode, if the drop site **XmNdragProc** resource is NULL, the toolkit also provides animation effects based on the **XmNanimationStyle** resource. Otherwise, if the **XmNdragProc** routine is specified, the receiver can either assume responsibility for animation effects (through the **XmNdragProc** routine) or rely on the toolkit to provide animation.

Drop sites may overlap. The initial stacking order corresponds to the order in which the drop sites were registered. When a drop site overlaps another drop site, the drag-under effects of the drop site underneath are clipped by the obscuring drop sites.

The *XmDropSiteUpdate*( ) routine sets resources for a widget that is registered as a drop site. *XmDropSiteRetrieve*( ) gets drop site resource values previously specified for a registered widget. These routines are used instead of *XtSetValues*( ) and *XtGetValues*( ).

Classes

XmDropSite does not inherit from any widget class.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. 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).

XmDropSite **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNanimationMask	**XmCAnimationMask**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNanimationPixmap	**XmCAnimationPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNanimationPixmap Depth	**XmCAnimationPixmap Depth**	**int**	0	CSG
XmNanimationStyle	**XmCAnimationStyle**	**unsigned char**	XmDRAG_UNDER _HIGHLIGHT	CSG
XmNdragProc	**XmCDragProc**	**XtCallbackProc**	NULL	CSG
XmNdropProc	**XmCDropProc**	**XtCallbackProc**	NULL	CSG
XmNdropRectangles	**XmCDropRectangles**	**XRectangle ***	dynamic	CSG
XmNdropSiteActivity	**XmCDropSiteActivity**	**unsigned char**	XmDROP_SITE _ACTIVE	CSG
XmNdropSiteOperations	**XmCDropSiteOperations**	**unsigned char**	XmDROP_MOVE | XmDROP_COPY	CSG
XmNdropSiteType	**XmCDropSiteType**	**unsigned char**	XmDROP_SITE _SIMPLE	CG
XmNimportTargets	**XmCImportTargets**	**Atom ***	NULL	CSG
XmNnumDropRectangles	**XmCNumDropRectangles**	**Cardinal**	1	CSG
XmNnumImportTargets	**XmCNumImportTargets**	**Cardinal**	0	CSG

XmNanimationMask

Specifies a mask to use with the pixmap specified by **XmNanimationPixmap** when the animation style is XmDRAG_UNDER_PIXMAP.

XmNanimationPixmap

Specifies a pixmap for drag-under animation when the animation style is XmDRAG_UNDER_PIXMAP. The pixmap is drawn with its origin at the upper-left corner of the bounding box of the drop site. If the drop site window is larger than the animation pixmap, the portion of the window not covered by the pixmap is tiled with the window's background color.

XmNanimationPixmapDepth

Specifies the depth of the pixmap specified by the **XmNanimationPixmap** resource. When the depth is 1, the colors are taken from the foreground and background of the drop site widget. For any other value, drop site animation occurs only if the **XmNanimationPixmapDepth** matches the depth of the drop site window. Colors are derived from the current colormap.

XmNanimationStyle

Specifies the drag-under animation style used when a drag enters a valid drop site. The possible values are

XmDRAG_UNDER_HIGHLIGHT

The drop site uses highlighting effects.

XmDRAG_UNDER_SHADOW_OUT

The drop site uses an outset shadow.

XmDRAG_UNDER_SHADOW_IN

The drop site uses an inset shadow.

XmDRAG_UNDER_PIXMAP

The drop site uses the pixmap specified by **XmNanimationPixmap** to indicate that it can receive the drop.

XmDRAG_UNDER_NONE

The drop site does not use animation effects. A client using a dynamic protocol, may provide drag-under effects in its **XmNdragProc** routine.

XmNdragProc

Specifies the procedure that is invoked when the drop site receives a crossing, motion or operation changed message. This procedure is called only when a dynamic protocol is used. The type of structure whose address is passed to this procedure is **XmDragProcCallbackStruct**. The reason sent to the procedure is one of the following:

XmCR_DROP_SITE_ENTER_MESSAGE
XmCR_DROP_SITE_LEAVE_MESSAGE
XmCR_DRAG_MOTION
XmCR_OPERATION_CHANGED.

The drag procedure may change the values of some members of the **XmDragProcCallbackStruct** passed to it. After the drag procedure returns, the toolkit uses the final values in initializing some members of the callback structure passed to the appropriate callbacks of the initiator (the DragContext's **XmNdropSiteEnterCallback**, **XmNdropSiteLeaveCallback**, **XmNoperationChangedCallback** or **XmNdragMotionCallback** callbacks).

XmNdropProc

Specifies the procedure invoked when a drop (excluding a cancel or interrupt action) occurs on a drop site regardless of the status of the drop site. The type of the structure whose address is passed to this procedure is **XmDropProcCallbackStruct**. The reason sent to the procedure is XmCR_DROP_MESSAGE.

The drop procedure may change the values of some members of the **XmDropProcCallbackStruct** passed to it. After the drop procedure returns, the toolkit uses the final values in initializing some members of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

XmNdropRectangles

Specifies a list of rectangles that describe the shape of a drop site. The locations of the rectangles are relative to the origin of the enclosing object. When **XmNdropRectangles** is NULL, the drop site is assumed to be the sensitive area of the enclosing widget. If **XmNdropSiteType** is XmDROP_SITE_COMPOSITE, this resource cannot be specified by the application.

Retrieving this resource returns allocated memory that needs to be freed with the *XtFree*( ) function.

XmNdropSiteActivity

Indicates whether a drop site is active or inactive. The values are XmDROP_SITE_ACTIVE, XmDROP_SITE_INACTIVE and XmDROP_SITE_IGNORE. An active drop site can receive a drop, whereas an inactive drop site is dormant. An inactive drop site is treated as if it is not a registered drop site and any drag-under visual symbols associated with entering or leaving the drop site do not occur. However, it is still used for clipping drag-under effects. A value of XmDROP_SITE_IGNORE indicates that a drop site should be ignored for all purposes.

XmNdropSiteOperations

Specifies the set of valid operations associated with a drop site. This resource is a bit mask formed by combining one or more of the following values using a bitwise operation such as inclusive OR (|): XmDROP_COPY, XmDROP_LINK and XmDROP_MOVE. The value XmDROP_NOOP for this resource indicates that no operations are valid.

XmNdropSiteType

Specifies the type of the drop site. The possible values are

XmDROP_SITE_SIMPLE
: The widget does not have any additional children that are registered as drop sites.

XmDROP_SITE_COMPOSITE
: The widget has children that are registered as drop sites.

XmNimportTargets

Specifies the list of target atoms that this drop site accepts.

XmNnumDropRectangles

Specifies the number of rectangles in the **XmNdropRectangles** list. If the drop site type is XmDROP_SITE_COMPOSITE, this resource cannot be specified by the application.

XmNnumImportTargets

Specifies the number of atoms in the target atom list.

Callback Information

A pointer to the following structure is passed to the **XmNdragProc** routine when the drop site receives crossing, motion or operation changed messages:

```
typedef struct
{
      int                        reason;
      XEvent                    *event;
      Time                       timeStamp;
      Widget                     dragContext;
      Position                   x;
      Position                   y;
      unsigned char              dropSiteStatus;
      unsigned char              operation;
      unsigned char              operations;
      Boolean                    animate;
} XmDragProcCallbackStruct, *XmDragProcCallback;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

dragContext Specifies the ID of the DragContext widget associated with the transaction.

x Indicates the x coordinate of the pointer relative to the drop site.

y Indicates the y coordinate of the pointer relative to the drop site.

dropSiteStatus

An in/out member that indicates whether or not a drop site is valid.

When **reason** is XmCR_DROP_SITE_ENTER_MESSAGE or XmCR_OPERATION_CHANGED, or **reason** is XmCR_DRAG_MOTION or XmCR_DROP_SITE_LEAVE_MESSAGE and the pointer is not in the same drop site as on the previous invocation of the drag procedure, the toolkit initializes **dropSiteStatus** to XmDROP_SITE_VALID if the DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible and if the initial value of the **operation** member is not XmDROP_NOOP. Otherwise, the toolkit initializes **dropSiteStatus** to XmDROP_SITE_INVALID.

When the **reason** is XmCR_DRAG_MOTION or XmCR_DROP_SITE_LEAVE_MESSAGE and the pointer is within the same drop site as on the previous invocation of the drag procedure, the toolkit initializes **dropSiteStatus** to the value of **dropSiteStatus** at the time the previous invocation of the drag procedure returns.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the **dropSiteStatus** member of the callback structure passed to the appropriate callbacks of the initiator.

operation An in/out member that identifies an operation.

The toolkit initializes **operation** by selecting an operation from the bitwise AND of the initial value of the **operations** member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for XmDROP_MOVE, then for XmDROP_COPY, then for XmDROP_LINK, and initializes **operation** to the first operation it finds in the set. If the toolkit finds none of these operations in the set, it initializes **operation** to XmDROP_NOOP.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the **operation** member of the callback structure passed to the appropriate callbacks of the initiator.

operations An in/out member that indicates the set of operations supported for the source data.

If the user does not select an operation (by pressing a modifier key), the toolkit initializes **operations** to the value of the DragContext's **XmNdragOperations** resource. If the user does select an operation, the toolkit initializes **operations** to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes **operations** to XmDROP_NOOP.

The drag procedure may change the value of this member. After the drag procedure returns, the toolkit uses the final value in initializing the **operations** member of the callback struct passed to the appropriate callbacks of the initiator.

animate An out member that indicates whether the toolkit or the receiver client provides drag-under effects for a valid drop site. If *animate* is set to True, the toolkit provides drop site animation per the **XmNanimationStyle** resource value; if it is set to False, the receiver generates drag-under animation effects.

A pointer to the following structure is passed to the **XmNdropProc** routine when the drop site receives a drop message:

```
typedef struct
{
      int                       reason;
      XEvent                   *event;
      Time                      timeStamp;
      Widget                    dragContext;
      Position                  x;
      Position                  y;
      unsigned char             dropSiteStatus;
      unsigned char             operation;
      unsigned char             operations;
      unsigned char             dropAction;
} XmDropProcCallbackStruct, *XmDropProcCallback;
```

reason Indicates why the callback was invoked.

event Specifies the **XEvent** that triggered the callback.

timeStamp Specifies the timestamp of the logical event.

dragContext Specifies the ID of the DragContext widget associated with the transaction.

x Indicates the x coordinate of the pointer relative to the drop site.

y Indicates the y coordinate of the pointer relative to the drop site.

dropSiteStatus
An in/out member that indicates whether or not a drop site is valid.

The toolkit initializes **dropSiteStatus** to XmDROP_SITE_VALID if the DragContext's **XmNexportTargets** and the DropSite's **XmNimportTargets** are compatible and if the initial value of the **operation** member is not XmDROP_NOOP. Otherwise, the toolkit initializes **dropSiteStatus** to XmDROP_SITE_INVALID.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the **dropSiteStatus** member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

operation An in/out member that identifies an operation.

The toolkit initializes **operation** by selecting an operation from the bitwise AND of the initial value of the **operations** member and the value of the DropSite's **XmNdropSiteOperations** resource. The toolkit searches this set first for XmDROP_MOVE, then for XmDROP_COPY, then for XmDROP_LINK, and initializes **operation** to the first operation it finds in the set. If it finds none of these operations in the set, it initializes **operation** to XmDROP_NOOP.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the **operation** member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

operations An in/out member that indicates the set of operations supported for the source data.

If the user does not select an operation (by pressing a modifier key), the toolkit initializes **operations** to the value of the DragContext's **XmNdragOperations** resource. If the user does select an operation, the toolkit initializes **operations** to the bitwise AND of the corresponding operation and the value of the DragContext's **XmNdragOperations** resource. If the resulting set of operations is empty, the toolkit initializes **operations** to XmDROP_NOOP.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the **operations** member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

dropAction An in/out member that identifies the action associated with the drop. The possible values are:

XmDROP
A drop was attempted. If the drop site is valid, drop transfer handling proceeds.

XmDROP_HELP
The user has requested help on the drop site.

The drop procedure may change the value of this member. After the drop procedure returns, the toolkit uses the final value in initializing the *dropAction* member of the **XmDropStartCallbackStruct** passed to the initiator's drop start callbacks (the DragContext's **XmNdropStartCallback** callbacks).

SEE ALSO

XmDragContext, *XmDragIcon*, *XmDropSiteConfigureStackingOrder*( ), *XmDropSiteEndUpdate*( ), *XmDropSiteQueryStackingOrder*( ), *XmDropSiteRegister*( ), *XmDropSiteStartUpdate*( ), *XmDropSiteUpdate*( ), *XmDropSiteUnregister*( ), *XmDropTransfer* and *XmTargetsAreCompatible*( ).

NAME

XmDropSiteConfigureStackingOrder — a Drag and Drop function that reorders a stack of widgets that are registered drop sites

SYNOPSIS

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

void XmDropSiteConfigureStackingOrder(
      Widget                      widget,
      Widget                      sibling,
      Cardinal                    stack_mode);
```

DESCRIPTION

XmDropSiteConfigureStackingOrder( ) changes the stacking order of the drop site specified by *widget*. The stacking order controls the manner in which drag-under effects are clipped by overlapping siblings, regardless of whether they are active. The stack mode is relative either to the entire stack, or to another drop site within the stack. The stack order can be modified only if the drop sites are siblings in both the widget and drop site hierarchy, and the widget parent of the drop sites is registered as a composite drop site.

widget Specifies the drop site to be restacked.

sibling Specifies a sibling drop site for stacking operations. If specified, then *widget* is restacked relative to this drop site's stack position.

stack_mode Specifies the new stack position for the specified widget. The values are XmABOVE and XmBELOW. If a sibling is specified, then *widget* is restacked as follows:

XmABOVE
The widget is placed just above the sibling.

XmBELOW
The widget is placed just below the sibling.

If the *sibling* argument is not specified, then *widget* is restacked as follows:

XmABOVE
The widget is placed at the top of the stack.

XmBELOW
The widget is placed at the bottom of the stack.

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

SEE ALSO

XmDropSite, *XmDropSiteRetrieve*( ) and *XmDropSiteQueryStackingOrder*( ).

NAME

XmDropSiteEndUpdate — a Drag and Drop function that facilitates processing updates to multiple drop sites

SYNOPSIS

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

void XmDropSiteEndUpdate(
      Widget                    widget);
```

DESCRIPTION

XmDropSiteEndUpdate( ) is used in conjunction with *XmDropSiteStartUpdate*( ) to process updates to multiple drop sites within the same hierarchy. *XmDropSiteStartUpdate*( ) and *XmDropSiteEndUpdate*( ) signal the beginning and the end respectively of a series of calls to *XmDropSiteUpdate*( ). Calls to *XmDropSiteStartUpdate*( ) and *XmDropSiteEndUpdate*( ) can be recursively stacked. Using these routines optimizes the processing of update information.

widget Specifies the ID of any widget within a given hierarchy. The function uses this widget to identify the shell that contains the drop sites.

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

SEE ALSO

XmDropSiteStartUpdate( ) and *XmDropSiteUpdate*( ).

NAME

XmDropSiteQueryStackingOrder — a Drag and Drop function that returns the parent, a list of children and the number of children for a specified widget

SYNOPSIS

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

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

DESCRIPTION

XmDropSiteQueryStackingOrder( ) obtains the parent, a list of children registered as drop sites and the number of children registered as drop sites for a given widget. The children are listed in current stacking order, from bottom-most (first child) to the top-most (last child). This function allocates memory for the returned data that must be freed by calling *XtFree*( ).

widget Specifies the widget ID. For this widget, you obtain the list of its children, its parent and the number of children.

parent_return Returns the widget ID of the drop site parent of the specified widget.

child_returns Returns a pointer to the list of drop site children associated with the specified widget. The function allocates memory to hold the list. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XtFree*( ).

num_child_returns
Returns the number of drop site children for the specified widget.

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

RETURN VALUE

Returns 0 (zero) if the routine fails; returns a non-zero value if it succeeds.

SEE ALSO

XmDropSite and *XmDropSiteConfigureStackingOrder*( ).

NAME

XmDropSiteRegister — a Drag and Drop function that identifies a drop site and assigns resources that specify its behavior

SYNOPSIS

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

void XmDropSiteRegister(
      Widget                     widget,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmDropSiteRegister( ) identifies the specified widget or gadget as a drop site and sets resource values that define the drop site's behavior. The routine assigns default values to any resources that are not specified in the argument list. The toolkit generates a warning message if a drop site is registered with **XmNdropSiteActivity** set to XmDROP_SITE_ACTIVE and the **XmNdropProc** resource is NULL.

If the drop site is a descendant of a widget that is registered as a drop site, the **XmNdropSiteType** resource of the ancestor drop site must be specified as XmDROP_SITE_COMPOSITE. The ancestor must be registered before the descendant. The drop site is stacked above all other sibling drop sites already registered.

widget Specifies the ID of the widget to be registered.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

SEE ALSO

XmDisplay, *XmDropSite*, *XmDropSiteEndUpdate*( ), *XmDropSiteStartUpdate*( ), *XmDropSiteUpdate*( ), *XmDropSiteUnregister*( ) and *XmScreen*.

NAME

XmDropSiteRegistered — a Drag and Drop function that determines if a drop site has been registered

SYNOPSIS

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

Boolean XmDropSiteRegistered(
      Widget                    widget);
```

DESCRIPTION

XmDropSiteRegistered( ) determines if the specified widget has a drop site registered. If a drop site is registered, this function returns True.

widget Specifies the ID of the widget being queried.

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

RETURN VALUE

If the widget is not a registered drop site, this function returns False. Otherwise, it returns True.

SEE ALSO

XmDisplay, *XmDropSite*, *XmDropSiteEndUpdate*( ), *XmDropSiteStartUpdate*( ), *XmDropSiteUpdate*( ), *XmDropSiteUnregister*( ) and *XmScreen*.

NAME

XmDropSiteRetrieve — a Drag and Drop function that retrieves resource values set on a drop site

SYNOPSIS

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

void XmDropSiteRetrieve(
      Widget                     widget,
      ArgList                    arglist,
      Cardinal                   argcount);
```

DESCRIPTION

XmDropSiteRetrieve( ) extracts values for the given resources from the drop site specified by *widget*. An initiator can also obtain information about the current drop site by passing the associated DragContext widget as the *widget* argument to this routine. The initiator can retrieve all of the drop site resources except **XmNdragProc** and **XmNdropProc** using this method.

widget Specifies the ID of the widget that encloses the drop site.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

SEE ALSO

XmDropSite and *XmDropSiteUpdate*( ).

NAME

XmDropSiteStartUpdate — a Drag and Drop function that facilitates processing updates to multiple drop sites

SYNOPSIS

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

void XmDropSiteStartUpdate(
      Widget                    widget);
```

DESCRIPTION

XmDropSiteStartUpdate( ) is used in conjunction with *XmDropSiteEndUpdate*( ) to process updates to multiple drop sites within the same shell widget. *XmDropSiteStartUpdate*( ) and *XmDropSiteEndUpdate*( ) signal the beginning and the end respectively of a series of calls to *XmDropSiteUpdate*( ). Calls to *XmDropSiteStartUpdate*( ) and *XmDropSiteEndUpdate*( ) can be recursively stacked. Using these routines optimizes the processing of update information.

widget Specifies the ID of any widget within a given hierarchy. The function uses this widget to identify the shell that contains the drop sites.

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

SEE ALSO

XmDropSite, *XmDropSiteEndUpdate*( ) and *XmDropSiteUpdate*( ).

NAME

XmDropSiteUnregister — a Drag and Drop function that frees drop site information

SYNOPSIS

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

void XmDropSiteUnregister(
      Widget                    widget);
```

DESCRIPTION

XmDropSiteUnregister( ) informs the toolkit that the specified widget is no longer a registered drop site. The function frees all associated drop site information.

widget Specifies the ID of the widget, registered as a drop site, that is to be unregistered.

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

SEE ALSO

XmDropSite and *XmDropSiteRegister*( ).

NAME

XmDropSiteUpdate — a Drag and Drop function that sets resource values for a drop site

SYNOPSIS

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

void XmDropSiteUpdate(
      Widget                      widget,
      ArgList                     arglist,
      Cardinal                    argcount);
```

DESCRIPTION

XmDropSiteUpdate( ) modifies drop site resources associated with the specified widget. This routine updates the drop site resources specified in the *arglist*.

widget Specifies the ID of the widget registered as a drop site.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*)

For a complete definition of DropSite and its associated resources, see *XmDropSite*.

SEE ALSO

XmDropSite, *XmDropSiteEndUpdate*( ), *XmDropSiteRegister*( ), *XmDropSiteStartUpdate*( ) and *XmDropSiteUnregister*( ).

NAME

XmDropTransfer — the DropTransfer widget class

SYNOPSIS

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

DESCRIPTION

DropTransfer provides a set of resources that identifies the procedures and associated information required by the toolkit 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 **ICCCM** specification.

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 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).

XmDropTransfer **Resource Set**

Name	Class	Type	Default	Access
XmNdropTransfers	**XmCDropTransfers**	**XmDropTransferEntryRec ***	NULL	CG
XmNincremental	**XmCIncremental**	**Boolean**	False	CSG
XmNnumDropTransfers	**XmCNumDropTransfers**	**Cardinal**	0	CSG
XmNtransferProc	**XmCTransferProc**	**XtSelectionCallbackProc**	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 the**ICCCM** specification. 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 the **ICCCM** specification.

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

SEE ALSO

Object, *XmDisplay*, *XmDragContext*, *XmDragIcon*, *XmDropSite*, *XmDropTransferAdd*( ) and *XmDropTransferStart*( ).

NAME

XmDropTransferAdd — a Drag and Drop function that enables additional drop transfer entries to be processed after initiating a drop transfer

SYNOPSIS

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

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

DESCRIPTION

XmDropTransferAdd( ) identifies a list of additional drop transfer entries to be processed after a drop transfer is started.

drop_transfer Specifies the ID of the DropTransfer widget returned by *XmDropTransferStart*( ).

transfers Specifies the additional drop transfer entries that the receiver wants processed.

num_transfers
Specifies the number of items in the *transfers* array.

For a complete definition of DropTransfer and its associated resources, see *XmDropTransfer*.

SEE ALSO

XmDragContext, *XmDropTransfer* and *XmDropTransferStart*( ).

NAME

XmDropTransferStart — a Drag and Drop function that initiates a drop transfer

SYNOPSIS

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

Widget XmDropTransferStart(
      Widget                        widget,
      ArgList                       arglist,
      Cardinal                      argcount);
```

DESCRIPTION

XmDropTransferStart( ) initiates a drop transfer and uses the specified argument list to initialize an *XmDropTransfer* object. The DropTransfer object can be manipulated with *XtSetValues*( ) and *XtGetValues*( ) until the last call to the **XmNtransferProc** procedure is made. After that point, the result of using the widget pointer is undefined. The DropTransfer object is freed by the toolkit when a transfer is complete.

widget Specifies the ID of the DragContext widget associated with the transaction.

arglist Specifies the argument list.

argcount Specifies the number of attribute and value pairs in the argument list (*arglist*).

For a complete definition of DropTransfer and its associated resources, see *XmDropTransfer*.

RETURN VALUE

Returns the ID of the DropTransfer widget.

SEE ALSO

XmDragContext, *XmDropTransfer* and *XmDropTransferAdd*( ).

NAME

XmFileSelectionBox — the FileSelectionBox widget class

SYNOPSIS

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

DESCRIPTION

FileSelectionBox traverses through directories, views the files and subdirectories in them, and then selects files.

A FileSelectionBox has five main areas:

- a text input field for displaying and editing a directory mask used to select the files to be displayed
- a scrollable list of filenames
- a scrollable list of subdirectories
- a text input field for displaying and editing a filename
- a group of PushButtons labeled **OK**, **Filter**, **Cancel** and **Help**.

Additional children may be added to the FileSelectionBox after creation. 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 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 subdirectories in the base directory that match the search pattern.

A procedure specified by the **XmNqualifySearchDataProc** resource extracts the base directory and search pattern from the directory mask. If the directory specification is empty, the current working directory is used. If the search pattern is empty, a pattern that matches all files is used.

An application can supply its own **XmNqualifySearchDataProc** as well as its own procedures to search for subdirectories and files. The default **XmNqualifySearchDataProc** works as follows: The directory mask is a pathname that can contain zero or more *wildcard* characters in its directory portion, its file portion or both. The directory components of the directory mask — up to, but not including, the first component with a wildcard character — specify the directory to be searched, relative to the current working directory. The remaining components specify the search pattern. If the directory mask is empty or if its first component contains a wildcard character, the current working directory is searched. If no component of the directory mask contains a wildcard character, the entire directory mask is the directory specification, and all files in that directory are matched.

The user can select a new directory to examine by scrolling through the list of directories and selecting the desired directory or by editing the directory mask. Selecting a new directory from the directory list does not change the search pattern. A user can select a new search pattern by editing the directory mask. Double clicking or pressing **KActivate** on a directory in the directory list initiates a search for files and subdirectories in the new directory, using the current search pattern.

The user can select a file by scrolling through the list of filenames and selecting the desired file or by entering the filename directly into the text edit area. Selecting a file from the list causes that filename to appear in the file selection text edit area.

The user may select a new file as many times as desired. The application is not notified until the user takes one of the following actions:

- Selects the **OK** PushButton.
- Presses **KActivate** while the selection text edit area has the keyboard focus.
- Double clicks or presses **KActivate** on an item in the file list.

FileSelectionBox initiates a directory and file search when any of the following occurs:

-
- The 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**).

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	*XmLabel*	title above selection box
Separator	*XmSeparatorGadget*	optional dividing line
Text	*XmText* or *XmTextField*	text within selection box

Classes

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

The class pointer is **xmFileSelectionBoxWidgetClass**.

The class name is *XmFileSelectionBox*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
XmNdirListLabelString	**XmCDirListLabelString**	**XmString**	dynamic	CSG
XmNdirMask	**XmCDirMask**	**XmString**	dynamic	CSG
XmNdirSearchProc	**XmCDirSearchProc**	**XmSearchProc**	default procedure	CSG
XmNdirSpec	**XmCDirSpec**	**XmString**	dynamic	CSG
XmNfileListItems	**XmCItems**	**XmStringTable**	dynamic	SG
XmNfileListItemCount	**XmCItemCount**	**int**	dynamic	SG
XmNfileListLabelString	**XmCFileListLabelString**	**XmString**	dynamic	CSG
XmNfileSearchProc	**XmCFileSearchProc**	**XmSearchProc**	default procedure	CSG
XmNfileTypeMask	**XmCFileTypeMask**	**unsigned char**	XmFILE_REGULAR	CSG
XmNfilterLabelString	**XmCFilterLabelString**	**XmString**	dynamic	CSG
XmNlistUpdated	**XmCListUpdated**	**Boolean**	dynamic	SG
XmNnoMatchString	**XmCNoMatchString**	**XmString**	" [ ] "	CSG
XmNpattern	**XmCPattern**	**XmString**	dynamic	CSG
XmNqualifySearchDataProc	**XmCQualifySearchDataProc**	**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.

XmNdirListItemCount

Specifies the number of items in the directory list. The value must not be negative.

XmNdirListLabelString

Specifies the label string of the directory list. The default for this resource depends on the locale. In the C locale the default is **Directories**.

XmNdirMask

Specifies the directory mask used in determining the files and directories to be displayed. The default value is determined by the **XmNqualifySearchDataProc** and depends on the initial values of **XmNdirMask**, **XmNdirectory** and **XmNpattern**.

XmNdirSearchProc

Specifies a directory search procedure to replace the default directory search procedure. FileSelectionBox's default directory search procedure fulfills the needs of most applications. Because it is impossible to cover the requirements of all applications, you can replace the default search procedure.

The directory search procedure is called with two arguments: the FileSelectionBox widget and a pointer to an **XmFileSelectionBoxCallbackStruct** structure. The callback structure is generated by the **XmNqualifySearchDataProc** and contains all information required to conduct a directory search, including the directory mask and a qualified base directory and search pattern. Once called, it is up to the search routine to generate a new list of directories and update the FileSelectionBox widget by using *XtSetValues*( ).

The search procedure must set **XmNdirectoryValid** and **XmNlistUpdated**. If it generates a new list of directories, it must also set **XmNdirListItems** and **XmNdirListItemCount**.

If the search procedure cannot search the specified directory, it must warn the user and set **XmNdirectoryValid** and **XmNlistUpdated** to False, unless it prompts and subsequently obtains a valid directory. If the directory is valid but is the same as the current **XmNdirectory**, the search procedure must set **XmNdirectoryValid** to True, but it may elect not to generate a new list of directories. In this case, it must set **XmNlistUpdated** to False.

If the search procedure generates a new list of directories, it must set **XmNdirListItems** to the new list of directories and **XmNdirListItemCount** to the number of items in the list. If there are no directories, it sets **XmNdirListItems** to NULL and **XmNdirListItemCount** to 0 (zero). In either case, it must set **XmNdirectoryValid** and **XmNlistUpdated** to True.

The search procedure ordinarily should not change the callback structure. But if the original directory is not valid, the search procedure may obtain a new directory from the user. In this case, it should set the **dir** member of the callback structure to the new directory, call the **XmNqualifySearchDataProc** with the callback struct as the input argument and copy the qualified data returned by the **XmNqualifySearchDataProc** into the callback struct.

XmNdirSpec
Specifies the full file path specification. This is the **XmNtextString** resource in SelectionBox, renamed for FileSelectionBox. The default value is determined by the FileSelectionBox after conducting the initial directory and file search.

XmNfileListItems
Specifies the items in the file list. This is the **XmNlistItems** resource in SelectionBox, renamed for FileSelectionBox.

XmNfileListItemCount
Specifies the number of items in the file list. This is the **XmNlistItemCount** resource in SelectionBox, renamed for FileSelectionBox. The value must not be negative.

XmNfileListLabelString
Specifies the label string of the file list. This is the **XmNlistLabelString** resource in SelectionBox, renamed for FileSelectionBox. The default for this resource depends on the locale. In the C locale the default is **Files**.

XmNfileSearchProc
Specifies a file search procedure to replace the default file search procedure. FileSelectionBox's default file search procedure fulfills the needs of most applications. Because it is impossible to cover the requirements of all applications, you can replace the default search procedure.

The file search procedure is called with two arguments: the FileSelectionBox widget and a pointer to an **XmFileSelectionBoxCallbackStruct** structure. The callback structure is generated by the **XmNqualifySearchDataProc** (and possibly modified by the **XmNdirSearchProc**). It contains all information required to conduct a file search, including the directory mask and a qualified base directory and search pattern. Once this procedure is called, it is up to the search routine to generate a new list of files and update the FileSelectionBox widget by using *XtSetValues*( ).

The search procedure must set **XmNlistUpdated**. If it generates a new list of files, it must also set **XmNfileListItems** and **XmNfileListItemCount**.

It is recommended that the search procedure always generate a new list of files. If the **mask** member of the callback structure is the same as the **mask** member of the callback struct in the preceding call to the search procedure, the procedure may elect not to generate a new list of files. In this case it must set **XmNlistUpdated** to False.

If the search procedure generates a new list of files, it must set **XmNfileListItems** to the new list of files and **XmNfileListItemCount** to the number of items in the list. If there are no files, it sets **XmNfileListItems** to NULL and **XmNfileListItemCount** to 0 (zero). In either case it must set **XmNlistUpdated** to True.

In constructing the list of files, the search procedure should include only files of the types specified by the widget's **XmNfileTypeMask**.

Setting **XmNdirSpec** is optional, but recommended. Set this attribute to the full file specification of the directory searched. The directory specification is displayed below the directory and file lists.

XmNfileTypeMask

Specifies the type of files listed in the file list. 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**.

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.

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)(
      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)(
      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	N/A
XmNapplyLabelString	**XmCApplyLabelString**	**XmString**	dynamic	N/A
XmNcancelCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNcancelLabelString	**XmCCancelLabelString**	**XmString**	dynamic	N/A
XmNchildPlacement	**XmCChildPlacement**	**unsigned char**	XmPLACE _ABOVE_ SELECTION	CSG
XmNdialogType	**XmCDialogType**	**unsigned char**	XmDIALOG _COMMAND	G
XmNhelpLabelString	**XmCHelpLabelString**	**XmString**	dynamic	N/A
XmNlistItemCount	**XmCItemCount**	**int**	0	CSG
XmNlistItems	**XmCItems**	**XmStringTable**	NULL	CSG
XmNlistLabelString	**XmCListLabelString**	**XmString**	NULL	N/A
XmNlistVisibleItemCount	**XmCVisibleItemCount**	**int**	dynamic	CSG
XmNminimizeButtons	**XmCMinimizeButtons**	**Boolean**	False	N/A
XmNmustMatch	**XmCMustMatch**	**Boolean**	False	N/A
XmNnoMatchCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNokCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNokLabelString	**XmCOkLabelString**	**XmString**	dynamic	N/A
XmNselectionLabelString	**XmCSelectionLabelString**	**XmString**	dynamic	CSG
XmNtextAccelerators	**XmCTextAccelerators**	**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
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
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
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 each callback:

```
typedef struct
{
      int                         reason;
      XEvent                     *event;
      XmString                    value;
      int                         length;
      XmString                    mask;
      int                         mask_length;
      XmString                    dir;
      int                         dir_length;
      XmString                    pattern;
      int                         pattern_length;
} XmFileSelectionBoxCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

value Specifies the current value of **XmNdirSpec**.

length Specifies the number of bytes in **value**.

mask Specifies the current value of **XmNdirMask**

mask_length Specifies the number of bytes in **mask**.

dir Specifies the current base directory.

dir_length Specifies the number of bytes in **dir**.

pattern Specifies the current search pattern.

pattern_length
Specifies the number of bytes in **pattern**.

Action Routines

The XmFileSelectionBox action routines are:

SelectionBoxUpOrDown(0 | 1 | 2 | 3)

If neither the selection text nor the directory mask (filter) text has the focus, this action does nothing.

If the selection text has the focus, the term *list* in the following description refers to the file list, and the term *text* refers to the selection text. If the directory mask text has the focus, *list* refers to the directory list, and *text* refers to the directory mask text.

When called with an argument of 0 (zero), this action selects the previous item in the list and replaces the text with that item.

When called with an argument of 1, this action selects the next item in the list and replaces the text with that item.

When called with an argument of 2, this action selects the first item in the list and replaces the text with that item.

When called with an argument of 3, this action selects the last item in the list and replaces the text with that item.

SelectionBoxRestore( )

If neither the selection text nor the directory mask (filter) text has the focus, this action does nothing.

If the selection text has the focus, this action replaces the selection text with the selected item in the file list. If no item in the file list is selected, it clears the selection text.

If the directory mask text has the focus, this action replaces the directory mask text with a new directory mask constructed from the **XmNdirectory** and **XmNpattern** resources.

SEE ALSO

Composite, *Constraint*, *Core*, *XmBulletinBoard*, *XmCreateFileSelectionBox*( ), *XmCreateFileSelectionDialog*( ), *XmFileSelectionBoxGetChild*( ), *XmFileSelectionDoSearch*( ), *XmManager* and *XmSelectionBox*.

NAME

XmFileSelectionBoxGetChild — a FileSelectionBox function used to access a component

SYNOPSIS

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

Widget XmFileSelectionBoxGetChild(
      Widget                    widget,
      unsigned char             child);
```

DESCRIPTION

XmFileSelectionBoxGetChild( ) is used to access a component within a FileSelectionBox. The arguments given to the function are the FileSelectionBox widget and a value indicating which component to access.

widget Specifies the FileSelectionBox widget ID.

child Specifies a component within the FileSelectionBox. The following are legal values for this argument:

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

For a complete definition of FileSelectionBox and its associated resources, see *XmFileSelectionBox*.

RETURN VALUE

Returns the widget ID of the specified FileSelectionBox component. An application should not assume that the returned widget is of any particular class.

SEE ALSO

XmFileSelectionBox.

NAME

XmFileSelectionDoSearch — a FileSelectionBox function that initiates a directory search

SYNOPSIS

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

void XmFileSelectionDoSearch(
      Widget                     widget,
      XmString                   dirmask);
```

DESCRIPTION

XmFileSelectionDoSearch( ) initiates a directory and file search in a FileSelectionBox widget. For a description of the actions that the FileSelectionBox takes when doing a search, see *XmFileSelectionBox*.

widget Specifies the FileSelectionBox widget ID.

dirmask Specifies the directory mask used in determining the directories and files displayed in the FileSelectionBox lists. This value is used as the **mask** member of the input data **XmFileSelectionBoxCallbackStruct** structure passed to the FileSelectionBox's **XmNqualifySearchDataProc**. The **dir** and **pattern** members of that structure are NULL.

For a complete definition of FileSelectionBox and its associated resources, see *XmFileSelectionBox*.

SEE ALSO

XmFileSelectionBox.

NAME

XmFontListAppendEntry — a font list function that appends an entry to a font list

SYNOPSIS

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

XmFontList XmFontListAppendEntry(
      XmFontList                 oldlist,
      XmFontListEntry            entry);
```

DESCRIPTION

XmFontListAppendEntry( ) creates a new font list that contains the contents of *oldlist*. This function copies the contents of the font list entry being added into this new font list. If *oldlist* is NULL, *XmFontListAppendEntry*( ) creates a new font list containing only the single entry specified.

This function deallocates the original font list after extracting the required information. The caller must free the font list entry by using *XmFontListEntryFree*( ).

oldlist Specifies the font list to which an entry is to be added.

entry Specifies the font list entry to be added.

RETURN VALUE

If *entry* is NULL, returns *oldlist*; otherwise, returns a new font list.

SEE ALSO

Section 4.2 on page 60, *XmFontListEntryCreate*( ), *XmFontListEntryFree*( ), *XmFontListEntryLoad*( ), *XmFontListFree*( ) and *XmFontListRemoveEntry( )*.

NAME

XmFontListCopy — a font list function that copies a font list

SYNOPSIS

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

XmFontList XmFontListCopy(
      XmFontList                fontlist);
```

DESCRIPTION

XmFontListCopy( ) creates a new font list consisting of the contents of the *fontlist* argument.

fontlist Specifies a font list to be copied.

RETURN VALUE

Returns NULL if *fontlist* is NULL; otherwise, returns a new font list.

SEE ALSO

Section 4.2 on page 60 and *XmFontListFree*( ).

NAME

XmFontListEntryCreate — a font list function that creates a font list entry

SYNOPSIS

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

DESCRIPTION

XmFontListEntryCreate( ) creates a font list entry that contains either a font or font set and is identified by a tag.

tag Specifies a NULL terminated string for the tag of the font list entry. The tag may be specified as XmFONTLIST_DEFAULT_TAG, which is used to identify the default font list element in a font list.

type Specifies whether the *font* argument is a font structure or a font set. Valid values are XmFONT_IS_FONT and XmFONT_IS_FONTSET.

font Specifies either an **XFontSet** returned by *XCreateFontSet*( ) or a pointer to an **XFontStruct** returned by *XLoadQueryFont*( ).

The toolkit does not copy the X Font structure specified by the *font* argument. Therefore, an application programmer must not free **XFontStruct** or **XFontSet** until all font lists or font entries that reference it have been freed.

RETURN VALUE

Returns a font list entry.

SEE ALSO

Section 4.2 on page 60, *XmFontListAppendEntry*( ), *XmFontListEntryFree*( ), *XmFontListEntryGetFont*( ), *XmFontListEntryGetTag*( ), *XmFontListEntryLoad*( ) and *XmFontListRemoveEntry*( ).

NAME

XmFontListEntryFree — a font list function that recovers memory used by a font list entry

SYNOPSIS

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

void XmFontListEntryFree(
      XmFontListEntry          *entry);
```

DESCRIPTION

XmFontListEntryFree( ) recovers memory used by a font list entry. This routine does not free the **XFontSet** or **XFontStruct** associated with the font list entry.

entry Specifies a pointer to the font list entry to be freed.

SEE ALSO

Section 4.2 on page 60, *XmFontListAppendEntry*( ), *XmFontListEntryCreate*( ), *XmFontListEntryLoad*( ), *XmFontListNextEntry*( ) and *XmFontListRemoveEntry*( ).

NAME

XmFontListEntryGetFont — a font list function that retrieves font information from a font list entry

SYNOPSIS

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

XtPointer XmFontListEntryGetFont(
      XmFontListEntry          entry,
      XmFontType              *type_return);
```

DESCRIPTION

XmFontListEntryGetFont( ) retrieves font information for a specified font list entry. If the font list entry contains a font, *type_return* returns XmFONT_IS_FONT and the function returns a pointer to an **XFontStruct**. If the font list entry contains a font set, *type_return* returns XmFONT_IS_FONTSET and the function returns the **XFontSet**.

entry Specifies the font list entry.

type_return Specifies a pointer to the type of the font element for the current entry. Valid values are XmFONT_IS_FONT and XmFONT_IS_FONTSET.

The returned **XFontSet** or **XFontStruct** is not a copy of the toolkit data and must not be freed.

RETURN VALUE

Returns an **XFontSet** or a pointer to an **XFontStruct** structure.

SEE ALSO

Section 4.2 on page 60, *XmFontListEntryCreate*( ), *XmFontListEntryGetTag*( )
XmFontListEntryLoad( ) and *XmFontListNextEntry*( ).

NAME

XmFontListEntryGetTag — a font list function that retrieves the tag of a font list entry

SYNOPSIS

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

char *XmFontListEntryGetTag(
      XmFontListEntry           entry);
```

DESCRIPTION

XmFontListEntryGetTag( ) retrieves a copy of the tag of the specified font list entry. This routine allocates memory for the tag string that must be freed by the application.

entry Specifies the font list entry.

RETURN VALUE

Returns the tag for the font list entry.

SEE ALSO

Section 4.2 on page 60, *XmFontListEntryCreate*( ), *XmFontListEntryGetFont*( ), *XmFontListEntryLoad*( ) and *XmFontListNextEntry*( ).

NAME

XmFontListEntryLoad — a font list function that loads a font or creates a font set and creates an accompanying font list entry

SYNOPSIS

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

XmFontListEntry XmFontListEntryLoad(
      Display                  *display,
      char                     *font_name,
      XmFontType                type,
      char                     *tag);
```

DESCRIPTION

XmFontListEntryLoad( ) loads a font or creates a font set based on the value of the *type* argument. It creates and returns a font list entry that contains the font or font set and the specified tag.

If the value of *type* is XmFONT_IS_FONT, the function uses the *XtCvtStringToFontStruct*( ) routine to convert the value of *font_name* to a font struct. If the value of *type* is XmFONT_IS_FONTSET, the function uses the *XtCvtStringToFontSet*( ) converter to create a font set in the current locale. *XmFontListEntryLoad*( ) creates a font list entry that contains the font or font set derived from the converter. For more information about *XtCvtStringToFontStruct*( ) and *XtCvtStringToFontSet*( ), see the **X Toolkit Intrinsics** specification.

display Specifies the display where the font list is used.

font_name Specifies an X Logical Font Description (XLFD) string, which is interpreted either as a font name or as a base font name list. A base font name list is a comma-separated and NULL-terminated string.

type Specifies whether the *font_name* argument refers to a font name or to a base font name list. Valid values are XmFONT_IS_FONT and XmFONT_IS_FONTSET.

tag Specifies the tag of the font list entry to be created. The tag may be specified as XmFONTLIST_DEFAULT_TAG, which is used to identify the default font list element in a font list when specified as part of a resource.

RETURN VALUE

If the specified font is not found, or the specified font set cannot be created, returns NULL; otherwise, returns a font list entry.

SEE ALSO

Section 4.2 on page 60, *XmFontListAppendEntry*( ), *XmFontListEntryCreate*( ), *XmFontListEntryFree*( ), *XmFontListEntryGetFont*( ), *XmFontListEntryGetTag*( ) and *XmFontListRemoveEntry*( ).

NAME

XmFontListFree — a font list function that recovers memory used by a font list

SYNOPSIS

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

void XmFontListFree(
      XmFontList                list);
```

DESCRIPTION

XmFontListFree( ) recovers memory used by a font list. This routine does not free the XFontSet or XFontStruct associated with the specified font list.

list Specifies the font list to be freed.

SEE ALSO

Section 4.2 on page 60, *XmFontListAppendEntry*( ), *XmFontListCopy*( ) and *XmFontListRemoveEntry*( ).

NAME

XmFontListFreeFontContext — a font list function that instructs the toolkit that the font list context is no longer needed

SYNOPSIS

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

void XmFontListFreeFontContext(
      XmFontContext             context);
```

DESCRIPTION

XmFontListFreeFontContext( ) instructs the toolkit that the context is no longer needed and is not used without reinitialization.

context Specifies the font list context structure allocated by the *XmFontListInitFontContext*( ) function.

SEE ALSO

XmFontListInitFontContext( ) and *XmFontListNextEntry*( ).

NAME

XmFontListInitFontContext — a font list function that allows applications to access the entries in a font list

SYNOPSIS

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

Boolean XmFontListInitFontContext(
      XmFontContext            *context,
      XmFontList                fontlist);
```

DESCRIPTION

XmFontListInitFontContext( ) establishes a context to allow applications to access the contents of a font list. This context is used when reading the font list entry tag, font, or font set associated with each entry in the font list. A Boolean status is returned to indicate whether or not the font list is valid.

context Specifies a pointer to the allocated context.

fontlist Specifies the font list.

RETURN VALUE

Returns True if the context was allocated; otherwise, returns False.

SEE ALSO

Section 4.2 on page 60, *XmFontListFreeFontContext*( ) and *XmFontListNextEntry*( ).

NAME

XmFontListNextEntry — a font list function that returns the next entry in a font list

SYNOPSIS

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

XmFontListEntry XmFontListNextEntry(
      XmFontContext             context);
```

DESCRIPTION

XmFontListNextEntry( ) returns the next entry in the font list. The application uses the *XmFontListInitFontContext*( ) routine to create a font list context. The first call to *XmFontListNextEntry*( ) sets the context to the first entry in the font list. The application then calls *XmFontListNextEntry*( ) repeatedly with the same context. Each succeeding call accesses the next entry of the font list. When finished, the application calls *XmFontListFreeFontContext*( ) to free the allocated font list context.

context Specifies the font list context.

RETURN VALUE

Returns NULL if the context does not refer to a valid entry or if it is at the end of the font list; otherwise, it returns a font list entry.

SEE ALSO

Section 4.2 on page 60, *XmFontListEntryFree*( ), *XmFontListEntryGetFont*( ), *XmFontListEntryGetTag*( ), *XmFontListFreeFontContext*( ) and *XmFontListInitFontContext*( ).

NAME

XmFontListRemoveEntry — a font list function that removes a font list entry from a font list

SYNOPSIS

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

XmFontList XmFontListRemoveEntry(
      XmFontList               oldlist,
      XmFontListEntry          entry);
```

DESCRIPTION

XmFontListRemoveEntry( ) creates a new font list that contains the contents of *oldlist* minus those entries specified in *entry*. The routine removes any entries from *oldlist* that match the components (tag, type font / font set) of the specified entry. The function deallocates the original font list after extracting the required information. The caller uses *XmFontListEntryFree*( ) to recover memory allocated for the specified entry. This routine does not free the **XFontSet** or **XFontStruct** associated with the font list entry that is removed.

oldlist Specifies the font list.

entry Specifies the font list entry to be removed.

RETURN VALUE

If *oldlist* is NULL, the function returns NULL. If *entry* is NULL or no entries are removed, the function returns *oldlist*. Otherwise, it returns a new font list.

SEE ALSO

Section 4.2 on page 60, *XmFontListAppendEntry*( ), *XmFontListEntryCreate*( ), *XmFontListEntryFree*( ), *XmFontListEntryLoad*( ) and *XmFontListFree*( ).

NAME

XmForm — the Form widget class

SYNOPSIS

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

DESCRIPTION

Form is a container widget with no input semantics of its own. Constraints are placed on children of the Form to define attachments for each of the child's four sides. These attachments can be to the Form, to another child widget or gadget, to a relative position within the Form, or to the initial position of the child. The attachments determine the layout behavior of the Form when resizing occurs.

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 is 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 is always at its preferred width (if resizable, otherwise at is current width). This may cause it to be clipped by the Form or by other children.

- If a child's left (or right) attachment is set to XmATTACH_SELF, its corresponding left (or right) offset is forced to 0 (zero). The attachment is then changed to XmATTACH_POSITION, with a position that corresponds to the *x* value of the child's left (or right) edge. To fix the position of a side at a specific *x* value, use XmATTACH_FORM or XmATTACH_OPPOSITE_FORM with the *x* value as the left (or right) offset.

- Unmapping a child has no effect on the Form except that the child is not mapped.

- Unmanaging a child unmaps it. If no other child is attached to it, or if all children attached to it and all children recursively attached to them are also all unmanaged, all of those children are treated as if they did not exist in determining the size of the Form.
- When using *XtSetValues*( ) to change the **XmNx** resource of a child, you must simultaneously set its left attachment to either XmATTACH_SELF or XmATTACH_NONE. Otherwise, the request is not granted. If **XmNresizable** is False, the request is granted only if the child's size can remain the same.
- A left (or right) attachment of XmATTACH_WIDGET, where **XmNleftWidget** (or **XmNrightWidget**) is NULL, acts like an attachment of XmATTACH_FORM.
- If an attachment is made to a widget that is not a child of the Form, but an ancestor of the widget is a child of the Form, the attachment is made to the ancestor.

All these considerations are true of top and bottom attachments as well, with top acting like left, bottom acting like right, *y* acting like *x*, and height acting like width.

Classes

Form inherits behavior and resources from *Core*, *Composite*, *Constraint*, *XmManager* and *XmBulletinBoard*.

The class pointer is **xmFormWidgetClass**.

The class name is *XmForm*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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 may depend on the value of the **XmNstringDirection** resource.)

The default left attachment is applied whenever initialization or *XtSetValues*( ) leaves the child without either a left or right attachment. The default top attachment is applied whenever initialization or *XtSetValues*( ) leaves the child without either a top or bottom attachment.

If this Boolean resource is set to False, **XmNleftAttachment** and **XmNtopAttachment** default to XmATTACH_FORM, **XmNleftOffset** defaults to the current *x* value of the left side of the child, and **XmNtopOffset** defaults to the current *y* value of the child. The effect is to position the child according to its absolute distance from the left or top side of the Form.

If this resource is set to True, **XmNleftAttachment** and **XmNtopAttachment** default to XmATTACH_POSITION, **XmNleftPosition** defaults to a value proportional to the current *x* value of the left side of the child divided by the width of the Form, and **XmNtopPosition** defaults to a value proportional to the current *y* value of the child divided by the height of the Form. The effect is to position the child relative to the left or top side of the Form and in proportion to the width or height of the Form.

XmNverticalSpacing

Specifies the offset for top and bottom attachments. This resource is only used if no offset resource is specified (when attaching to a widget), or if no margin resource is specified (when attaching to the Form).

XmForm **Constraint Resource Set**

Name	**Class**	**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 effect of a non-zero value for this resource is undefined if

XmNbottomAttachment is set to XmATTACH_POSITION. 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.

XmNleftAttachment

Specifies attachment of the near (left) side of the child. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) It can have the following values:

XmATTACH_NONE

Do not attach the left side of the child. If **XmNrightAttachment** is also XmATTACH_NONE, this value is ignored and the child is given a default left attachment.

XmATTACH_FORM

Attach the left side of the child to the left side of the Form.

XmATTACH_OPPOSITE_FORM

Attach the left side of the child to the right side of the Form. **XmNleftOffset** can be used to determine the visibility of the child.

XmATTACH_WIDGET

Attach the left side of the child to the right side of the widget or gadget specified in the **XmNleftWidget** resource. If **XmNleftWidget** is NULL, XmATTACH_WIDGET is replaced by XmATTACH_FORM, and the child is attached to the left side of the Form.

XmATTACH_OPPOSITE_WIDGET

Attach the left side of the child to the left side of the widget or gadget specified in the **XmNleftWidget** resource.

XmATTACH_POSITION

Attach the left side of the child to a position that is relative to the left side of the Form and in proportion to the width of the Form. This position is determined by the **XmNleftPosition** and **XmNfractionBase** resources.

XmATTACH_SELF

Attach the left side of the child to a position that is proportional to the current *x* value of the left side of the child divided by the width of the Form. This position is determined by the **XmNleftPosition** and **XmNfractionBase** resources. **XmNleftPosition** is set to a value proportional to the current *x* value of the left side of the child divided by the width of the Form.

XmNleftOffset

Specifies the constant offset between the near (left) side of the child and the object to which it is attached. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The effect of a non-zero value for this resource is undefined if **XmNleftAttachment** is set to XmATTACH_POSITION. The relationship established remains, regardless of any resizing operations that occur. When this resource is explicitly set, the value of **XmNhorizontalSpacing** is ignored.

XmNleftPosition

This resource is used to determine the position of the near (left) side of the child when the child's **XmNleftAttachment** is set to XmATTACH_POSITION. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.)

In this case, the position of the left side of the child is relative to the left side of the Form and is a fraction of the width of the Form. This fraction is the value of the child's **XmNleftPosition** resource divided by the value of the Form's **XmNfractionBase**. For example, if the child's **XmNleftPosition** is 50, the Form's **XmNfractionBase** is 100, and the Form's width is 200, the position of the left side of the child is 100.

XmNleftWidget

Specifies the widget or gadget to which the near (left) side of the child is attached. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The **XmNleftWidget** resource is used if the **XmNleftAttachment** resource is set to either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.

XmNresizable

This Boolean resource specifies whether or not a child's request for a new size is (conditionally) granted by the Form. If this resource is set to True the request is granted if possible. If this resource is set to False the request is always refused.

If a child has both left and right attachments, its width is completely controlled by the Form, regardless of the value of the child's **XmNresizable** resource. If a child has a left or right attachment but not both, the child's **XmNwidth** is used in setting its width if the value of the child's **XmNresizable** resource is True. These conditions are also true for top and bottom attachments, with height acting like width.

XmNrightAttachment

Specifies attachment of the far (right) side of the child. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) It can have the following values:

XmATTACH_NONE

Do not attach the right side of the child.

XmATTACH_FORM

Attach the right side of the child to the right side of the Form.

XmATTACH_OPPOSITE_FORM

Attach the right side of the child to the left side of the Form. **XmNrightOffset** can be used to determine the visibility of the child.

XmATTACH_WIDGET

Attach the right side of the child to the left side of the widget or gadget specified in the **XmNrightWidget** resource. If **XmNrightWidget** is NULL, XmATTACH_WIDGET is replaced by XmATTACH_FORM, and the child is attached to the right side of the Form.

XmATTACH_OPPOSITE_WIDGET

Attach the right side of the child to the right side of the widget or gadget specified in the **XmNrightWidget** resource.

XmATTACH_POSITION

Attach the right side of the child to a position that is relative to the left side of the Form and in proportion to the width of the Form. This position is determined by the **XmNrightPosition** and **XmNfractionBase** resources.

XmATTACH_SELF

Attach the right side of the child to a position that is proportional to the current *x* value of the right side of the child divided by the width of the Form. This position is determined by the **XmNrightPosition** and **XmNfractionBase** resources. **XmNrightPosition** is set to a value proportional to the current *x* value of the right side of the child divided by the width of the Form.

XmNrightOffset

Specifies the constant offset between the far (right) side of the child and the object to which it is attached. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The effect of a non-zero value for this resource is undefined if **XmNrightAttachment** is set to XmATTACH_POSITION. The relationship established remains, regardless of any resizing operations that occur. When this resource is explicitly set, the value of **XmNhorizontalSpacing** is ignored.

XmNrightPosition

This resource is used to determine the position of the far (right) side of the child when the child's **XmNrightAttachment** is set to XmATTACH_POSITION. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.)

In this case the position of the right side of the child is relative to the left side of the Form and is a fraction of the width of the Form. This fraction is the value of the child's **XmNrightPosition** resource divided by the value of the Form's **XmNfractionBase**. For example, if the child's **XmNrightPosition** is 50, the Form's **XmNfractionBase** is 100, and the Form's width is 200, the position of the right side of the child is 100.

XmNrightWidget

Specifies the widget or gadget to which the far (right) side of the child is attached. (Note that whether this resource actually applies to the left or right side of the child and its attachment may depend on the value of the **XmNstringDirection** resource.) The **XmNrightWidget** resource is used if the **XmNrightAttachment** resource is set to either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.

XmNtopAttachment

Specifies attachment of the top side of the child. It can have the 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 effect of a non-zero value for this resource is undefined if **XmNtopAttachment** is set to XmATTACH_POSITION. 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.

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**	False	N/A
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	N/A
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
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
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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
XmNinitialResourcesPersistent	**XmCInitialResourcesPersistent**	**Boolean**	True	C
XmNmappedWhenManaged	**XmCMappedWhenManaged**	**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

SEE ALSO

Composite, *Constraint*, *Core*, *XmBulletinBoard*, *XmCreateForm*( ), *XmCreateFormDialog*( ) and *XmManager*.

NAME

XmFrame — the Frame widget class

SYNOPSIS

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

DESCRIPTION

Frame is a very simple manager used to enclose a single 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.

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 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).

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.

XmNshadowType

Describes the drawing style for Frame. This resource can have the following values:

XmSHADOW_IN

Draws Headers/Frame.inco that it appears inset. This means that the bottom shadow visual symbols and top shadow visual symbols 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**
XmNframeChildType	**XmCFrameChildType**	**unsigned char**	XmFRAME_WORKAREA_CHILD	CSG

XmNframeChildType

Specifies whether a child is a title or work area. Frame supports a single title 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.

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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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

SEE ALSO

Composite, *Constraint*, *Core*, *XmCreateFrame*( ) and *XmManager*.

NAME

XmGadget — the Gadget widget class

SYNOPSIS

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

DESCRIPTION

Gadget is a widget class used as a supporting superclass for other gadget classes. It handles shadow-border drawing and highlighting, traversal activation and deactivation, and various callback lists needed by gadgets.

The color and pixmap resources defined by *XmManager* are directly used by gadgets. If *XtSetValues* is used to change one of the resources for a manager widget, all of the gadget children within the manager also change.

Classes

Gadget inherits behavior and resources from *Object* and *RectObj*.

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 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).

XmGadget **Resource Set**

Name	**Class**	**Type**	**Default**	**Access**
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	CSG
XmNuserData	**XmCUserData**	**XtPointer**	NULL	CSG

XmNhelpCallback

Specifies the list of callbacks called when the help key sequence is pressed. The reason sent by the callback is XmCR_HELP.

XmNhighlightOnEnter

Specifies if the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is XmEXPLICIT, this resource is ignored, and the widget is highlighted when it has the focus. If the shell's focus policy is XmPOINTER and if this resource is True, the highlighting rectangle is drawn when the the cursor moves into the widget. If the shell's focus policy is XmPOINTER and if this resource is False, the highlighting rectangle is not drawn when the the cursor moves into the widget. The default is False.

XmNhighlightThickness
Specifies the thickness of the highlighting rectangle.

XmNnavigationType
Determines whether the widget is a tab group:

XmNONE
Indicates that the widget is not a tab group.

XmTAB_GROUP
Indicates that the widget is a tab group, unless the **XmNnavigationType** of another widget in the hierarchy is XmEXCLUSIVE_TAB_GROUP.

XmSTICKY_TAB_GROUP
Indicates that the widget is a tab group, even if the **XmNnavigationType** of another widget in the hierarchy is XmEXCLUSIVE_TAB_GROUP.

XmEXCLUSIVE_TAB_GROUP
Indicates that the widget is a tab group and that widgets in the hierarchy whose **XmNnavigationType** is **XmTAB_GROUP** are not tab groups.

When a parent widget has an **XmNnavigationType** of XmEXCLUSIVE_TAB_GROUP, traversal of non-tab-group widgets within the group is based on the order of those widgets in their parent's **XmNchildren** list.

XmNshadowThickness
Specifies the size of the drawn border shadow.

XmNtraversalOn
Specifies traversal activation for this gadget.

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**
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. For this callback, **reason** is set to XmCR_HELP.

event Points to the **XEvent** that triggered the callback.

SEE ALSO

Object, *RectObj* and *XmManager*.

NAME

XmGetAtomName — a function that returns the string representation for an atom

SYNOPSIS

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

String XmGetAtomName(
      Display                  *display,
      Atom                      atom);
```

DESCRIPTION

XmGetAtomName( ) returns the string representation for an atom. It mirrors the Xlib interfaces for atom management but provides client-side storing in cache. When and where storing in cache is provided in Xlib, the routines will become pseudonyms for the Xlib routines.

display Specifies the connection to the X server.

atom Specifies the atom for the property name you want returned.

RETURN VALUE

Returns a string.

NAME

XmGetColors — a function that generates foreground, select and shadow colors

SYNOPSIS

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

void XmGetColors(
      Screen                    *screen,
      Colormap                   colormap,
      Pixel                      background,
      Pixel                     *foreground,
      Pixel                     *top_shadow,
      Pixel                     *bottom_shadow,
      Pixel                     *select);
```

DESCRIPTION

XmGetColors( ) takes a screen, a colormap, and a background pixel, and returns pixel values for foreground, select and shadow colors.

screen Specifies the screen for which these colors should be allocated.

colormap Specifies the colormap from which these colors should be allocated.

background Specifies the background on which the colors should be based.

foreground Specifies a pointer to the returned foreground pixel value. If this argument is NULL no value is returned for this color.

top_shadow Specifies a pointer to the returned top shadow pixel value. If this argument is NULL, no value is returned for this color.

bottom_shadow
Specifies a pointer to the returned bottom shadow pixel value. If this argument is NULL, no value is returned for this color.

select Specifies a pointer to the returned select pixel value. If this argument is NULL, no value is returned for this color.

NAME

XmGetDestination — a function that returns the widget ID of the widget to be used as the current destination for quick paste and certain clipboard operations

SYNOPSIS

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

Widget XmGetDestination(
     Display                    *display);
```

DESCRIPTION

XmGetDestination( ) returns the widget that is the current destination on the specified display. The destination is generally the last editable widget on which a select, edit, insert or paste operation was performed and is the destination for quick paste and certain clipboard functions. The destination is NULL if the application makes this call before any of the specified operations have been performed on an editable widget.

display Specifies the display whose destination widget is to be queried.

RETURN VALUE

Returns the widget ID for the current destination or NULL if there is no current destination.

NAME

XmGetFocusWidget — returns the ID of the widget that has keyboard focus

SYNOPSIS

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

Widget XmGetFocusWidget(
      Widget                    widget);
```

DESCRIPTION

XmGetFocusWidget( ) examines the hierarchy that contains the specified widget and returns the ID of the widget that has keyboard focus. The function extracts the widget ID from the associated Shell widget; therefore, the specified widget can be located anywhere in the hierarchy.

widget Specifies a widget ID within a given hierarchy.

RETURN VALUE

Returns the ID of the widget with keyboard focus. If no child of the Shell has focus, the function returns NULL.

SEE ALSO

XmProcessTraversal( ).

NAME

XmGetMenuCursor — a RowColumn function that returns the cursor ID for the current menu cursor

SYNOPSIS

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

Cursor XmGetMenuCursor(
      Display                  *display);
```

DESCRIPTION

XmGetMenuCursor( ) queries the menu cursor currently being used by this client on the specified display and returns the cursor ID.

display Specifies the display whose menu cursor is to be queried.

For a complete definition of the menu cursor resource, see *XmRowColumn*.

RETURN VALUE

Returns the cursor ID for the current menu cursor or the value None if a cursor is not yet defined. A cursor is not defined if the application makes this call before the client has created any menus on the specified display.

SEE ALSO

XmRowColumn.

NAME

XmGetPixmap — a pixmap function that generates a pixmap, stores it in a pixmap cache and returns the pixmap

SYNOPSIS

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

Pixmap XmGetPixmap(
      Screen                   *screen,
      char                     *image_name,
      Pixel                     foreground,
      Pixel                     background);
```

DESCRIPTION

XmGetPixmap( ) uses the argument data to perform a lookup in the pixmap cache to see if a pixmap has already been generated that matches the data. If one is found, a reference count is incremented and the pixmap is returned. Applications should use *XmDestroyPixmap*( ) when the pixmap is no longer needed.

screen Specifies the display screen on which the pixmap is to be drawn. The depth of the pixmap is the default depth for this screen.

image_name Specifies the name of the image to be used to generate the pixmap.

foreground Combines the image with the *foreground* color to create the pixmap if the image referenced is a bit-per-pixel image.

background Combines the image with the *background* color to create the pixmap if the image referenced is a bit-per-pixel image.

If a pixmap is not found, *image_name* is used to perform a lookup in the image cache. If an image is found, it is used to generate the pixmap, which is then stored in cache and returned.

If an image is not found, the *image_name* is used as a filename, and a search is made for an **X10** or **X11** bitmap file. If it is found, the file is read, converted into an image, and cached in the image cache. The image is then used to generate a pixmap, which is cached and returned.

If *image_name* has a leading slash (/), it specifies a full pathname, and *XmGetPixmap*( ) opens the file as specified. Otherwise, *image_name* specifies a filename. In this case, *XmGetPixmap*( ) looks for the file along a search path specified by the *XBMLANGPATH* environment variable or by a default search path, which varies depending on whether or not the *XAPPLRESDIR* environment variable is set.

The *XBMLANGPATH* environment variable specifies a search path for X bitmap files. It can contain the substitution field **%B**, where the *image_name* argument to *XmGetPixmap*( ) is substituted for **%B**. It can also contain the substitution fields accepted by *XtResolvePathname*. The substitution field **%T** is always mapped to bitmaps, and **%S** is always mapped to NULL.

If *XBMLANGPATH* is not set but the environment variable *XAPPLRESDIR* is set, the following pathnames are searched:

```
%B
$XAPPLRESDIR/%L/bitmaps/%N/%B
$XAPPLRESDIR/%l/bitmaps/%N/%B
$XAPPLRESDIR/bitmaps/%N/%B
$XAPPLRESDIR/%L/bitmaps/%B
$XAPPLRESDIR/%l/bitmaps/%B
$XAPPLRESDIR/bitmaps/%B
$HOME/bitmaps/%B
$HOME/%B
/usr/lib/X11/%L/bitmaps/%N/%B
/usr/lib/X11/%l/bitmaps/%N/%B
/usr/lib/X11/bitmaps/%N/%B
/usr/lib/X11/%L/bitmaps/%B
/usr/lib/X11/%l/bitmaps/%B
/usr/lib/X11/bitmaps/%B
/usr/include/X11/bitmaps/%B
```

If neither *XBMLANGPATH* nor *XAPPLRESDIR* is set, the following pathnames are searched:

```
%B
$HOME/%L/bitmaps/%N/%B
$HOME/%l/bitmaps/%N/%B
$HOME/bitmaps/%N/%B
$HOME/%L/bitmaps/%B
$HOME/%l/bitmaps/%B
$HOME/bitmaps/%B
$HOME/%B
/usr/lib/X11/%L/bitmaps/%N/%B
/usr/lib/X11/%l/bitmaps/%N/%B
/usr/lib/X11/bitmaps/%N/%B
/usr/lib/X11/%L/bitmaps/%B
/usr/lib/X11/%l/bitmaps/%B
/usr/lib/X11/bitmaps/%B
/usr/include/X11/bitmaps/%B
```

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

The following substitutions are used in these paths:

%B	the image name, from the *image_name* argument
%N	the class name of the application
%L	the display's language string
%l	the language component of the display's language string.

RETURN VALUE

Returns a pixmap when successful; returns [XmUNSPECIFIED_PIXMAP] if the image corresponding to *image_name* cannot be found.

SEE ALSO

XmDestroyPixmap( ), *XmInstallImage*( ) and *XmUninstallImage*( ).

NAME

XmGetPixmapByDepth — a pixmap function that generates a pixmap, stores it in a pixmap cache and returns the pixmap

SYNOPSIS

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

Pixmap XmGetPixmapByDepth(
      Screen                    *screen,
      char                      *image_name,
      Pixel                      foreground,
      Pixel                      background,
      int                        depth);
```

DESCRIPTION

XmGetPixmapByDepth( ) uses the argument data to perform a lookup in the pixmap cache to see if a pixmap has already been generated that matches the data. If one is found, a reference count is incremented and the pixmap is returned. Applications should use *XmDestroyPixmap*( ) when the pixmap is no longer needed.

screen Specifies the display screen on which the pixmap is to be drawn.

image_name Specifies the name of the image to be used to generate the pixmap.

foreground Combines the image with the *foreground* color to create the pixmap if the image referenced is a bit-per-pixel image.

background Combines the image with the *background* color to create the pixmap if the image referenced is a bit-per-pixel image.

depth Specifies the depth of the pixmap.

If a matching pixmap is not found, *image_name* is used to perform a lookup in the image cache. If an image is found, it is used to generate the pixmap, which is then stored in cache and returned.

If an image is not found, *image_name* is used as a filename, and a search is made for an X10 or X11 bitmap file. If it is found, the file is read, converted into an image, and stored in the image cache. The image is then used to generate a pixmap, which is stored in cache and returned.

If *image_name* has a leading slash (/), it specifies a full pathname, and *XmGetPixmapByDepth*( ) opens the file as specified. Otherwise, *image_name* specifies a filename. In this case, *XmGetPixmapByDepth*( ) looks for the file along a search path specified by the *XBMLANGPATH* environment variable or by a default search path, which varies depending on whether or not the *XAPPLRESDIR* environment variable is set.

The *XBMLANGPATH* environment variable specifies a search path for X bitmap files. It can contain the substitution field **%B**, where the *image_name* argument to *XmGetPixmapByDepth*( ) is substituted for **%B**. It can also contain the substitution fields accepted by *XtResolvePathname*ˆ( ). The substitution field **%T** is always mapped to bitmaps, and **%S** is always mapped to NULL.

If *XBMLANGPATH* is not set, but the environment variable *XAPPLRESDIR* is set, the following pathnames are searched:

%B
$XAPPLRESDIR/%L/bitmaps/%N/%B
$XAPPLRESDIR/%l/bitmaps/%N/%B
$XAPPLRESDIR/bitmaps/%N/%B
$XAPPLRESDIR/%L/bitmaps/%B
$XAPPLRESDIR/%l/bitmaps/%B
$XAPPLRESDIR/bitmaps/%B
$HOME/bitmaps/%B
$HOME/%B
/usr/lib/X11/%L/bitmaps/%N/%B
/usr/lib/X11/%l/bitmaps/%N/%B
/usr/lib/X11/bitmaps/%N/%B
/usr/lib/X11/%L/bitmaps/%B
/usr/lib/X11/%l/bitmaps/%B
/usr/lib/X11/bitmaps/%B
/usr/include/X11/bitmaps/%B

If neither *XBMLANGPATH* nor *XAPPLRESDIR* is set, the following pathnames are searched:

%B
$HOME/%L/bitmaps/%N/%B
$HOME/%l/bitmaps/%N/%B
$HOME/bitmaps/%N/%B
$HOME/%L/bitmaps/%B
$HOME/%l/bitmaps/%B
$HOME/bitmaps/%B
$HOME/%B
/usr/lib/X11/%L/bitmaps/%N/%B
/usr/lib/X11/%l/bitmaps/%N/%B
/usr/lib/X11/bitmaps/%N/%B
/usr/lib/X11/%L/bitmaps/%B
/usr/lib/X11/%l/bitmaps/%B
/usr/lib/X11/bitmaps/%B
/usr/include/X11/bitmaps/%B

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

The following substitutions are used in these paths:

%B	the image name, from the *image_name* argument
%N	the class name of the application
%L	the display's language string
%l	the language component of the display's language string.

The contents of the file must conform to the rules for X11 bitmap files. In other words, Motif can read any X11-conformant bitmap file.

RETURN VALUE

Returns a pixmap when successful; returns [XmUNSPECIFIED_PIXMAP] if the image corresponding to *image_name* cannot be found.

SEE ALSO

XmDestroyPixmap( ), *XmInstallImage*( ) and *XmUninstallImage*( ).

NAME

XmGetPostedFromWidget — a RowColumn function that returns the widget from which a menu was posted

SYNOPSIS

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

Widget XmGetPostedFromWidget(
      Widget                    menu);
```

DESCRIPTION

XmGetPostedFromWidget( ) returns the widget from which a menu was posted. An application can use this routine during the activate callback to determine the context in which the menu callback should be interpreted.

menu Specifies the widget ID of the menu.

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the widget ID of the widget from which the menu was posted. If the menu is a Popup Menu, the returned widget is the widget from which the menu was popped up. If the menu is a Pulldown Menu, the returned widget is the MenuBar or OptionMenu from which the widget was pulled down.

SEE ALSO

XmRowColumn.

NAME

XmGetSecondaryResourceData — a function that provides access to secondary widget resource data

SYNOPSIS

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

Cardinal XmGetSecondaryResourceData(
      WidgetClass                widget_class,
      XmSecondaryResourceData **secondary_data_return);
```

DESCRIPTION

Some Motif widget classes (such as Gadget, Text, and VendorShell) have resources that are not accessible through the functions *XtGetResourceList*( ) and *XtGetConstraintResourceList*( ). In order to retrieve the descriptions of these resources, an application must use *XmGetSecondaryResourceData*( ).

When a widget class has such resources, this function provides descriptions of the resources in one or more data structures. *XmGetSecondaryResourceData*( ) takes a widget class argument and returns the number of these data structures associated with the widget class. If the return value is greater than 0 (zero), the function allocates and fills an array of pointers to the corresponding data structures. It returns this array at the address that is the value of the *secondary_data_return* argument.

The type **XmSecondaryResourceData** is a pointer to a structure with two members that are useful to an application: **resources**, of type **XtResourceList**, and **num_resources**, of type **Cardinal**. The **resources** member is a list of the widget resources that are not accessible using Xt functions. The **num_resources** member is the length of the **resources** list.

If the return value is greater than 0 (zero), *XmGetSecondaryResourceData*( ) allocates memory that the application must free. Use *XtFree*( ) to free the resource list in each structure (the value of the *resources* member), the structures themselves, and the array of pointers to the structures (the array whose address is *secondary_data_return*).

widget_class Specifies the widget class for which secondary resource data is to be retrieved.

secondary_data_return
Specifies a pointer to an array of **XmSecondaryResourceData** pointers to be returned by this function. If the widget class has no secondary resource data, for example, if the value returned by the function is 0 (zero), the function returns no meaningful value for this argument.

RETURN VALUE

Returns the number of secondary resource data structures associated with this widget class.

EXAMPLE

The following example uses *XmGetSecondaryResourceData*( ) to print the names of the secondary resources of the Motif Text widget and then frees the data allocated by the function:

```
XmSecondaryResourceData * block_array ;
Cardinal num_blocks, i, j ;
if (num_blocks = XmGetSecondaryResourceData (xmTextWidgetClass,
                                             &block_array)) {
  for (i = 0; i < num_blocks; i++) {
    for (j = 0 ; j < block_array[i]->num_resources; j++) {
      printf("%s\n", block_array[i]->resources[j].resource_name);
    }
    XtFree((char*)block_array[i]->resources);
    XtFree((char*)block_array[i]);
  }
  XtFree((char*)block_array);
}
```

NAME

XmGetTabGroup — returns the widget ID of a tab group

SYNOPSIS

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

Widget XmGetTabGroup(
      Widget                    widget);
```

DESCRIPTION

XmGetTabGroup( ) returns the widget ID of the tab group that contains the specified widget.

widget Specifies a widget ID within a tab group.

RETURN VALUE

Returns the widget ID of a tab group or shell, determined as follows:

- If *widget* is a tab group or shell, returns *widget*.
- If neither *widget* nor any ancestor up to the nearest shell is a tab group, returns the nearest ancestor of *widget* that is a shell.
- Otherwise, returns the nearest ancestor of *widget* that is a tab group.

SEE ALSO

XmAddTabGroup( ), *XmManager* and *XmPrimitive*.

NAME

XmGetTearOffControl — a RowColumn function that obtains the widget ID for the tear-off control in a menu

SYNOPSIS

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

Widget XmGetTearOffControl(
      Widget                    menu);
```

DESCRIPTION

XmGetTearOffControl( ) provides the application with the means for obtaining the widget ID of the internally created tear-off control in a tear-off menu.

RowColumn creates a tear-off control for a PulldownMenu or PopupMenu when the **XmNtearOffModel** resource is initialised or set to XmTEAR_OFF_ENABLED. The tear-off control is a widget that appears as the first element in the menu. The user tears off the menu by means of mouse or keyboard events in the tear-off control.

The tear-off control has Separator-like behavior. Once the application has obtained the widget ID of the tear-off control, it can set resources to specify the appearance of the control. The application or user can also set these resources in a resource file by using the name of the control, which is **TearOffControl**. For a list of the resources the application or user can set, see *XmRowColumn*.

menu Specifies the widget ID of the RowColumn PulldownMenu or PopupMenu.

For more information on tear-off menus and a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the widget ID for the tear-off control, or NULL if no tear-off control exists. An application should not assume that the returned widget is of any particular class.

SEE ALSO

XmRowColumn.

NAME

XmGetVisibility — a function that determines if a widget is visible

SYNOPSIS

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

XmVisibility XmGetVisibility(
      Widget                    widget);
```

DESCRIPTION

XmGetVisibility( ) returns the visibility state of the specified widget.

widget Specifies the ID of the widget.

RETURN VALUE

Returns one of the following values:

[XmVISIBILITY_UNOBSCURED]
Indicates that the widget is mapped, not obscured, and is completely visible on the screen.

[XmVISIBILITY_PARTIALLY_OBSCURED]
Indicates that the widget is mapped, and is not completely visible on the screen (partially obscured).

[XmVISIBILITY_FULLY_OBSCURED]
Indicates that the widget is not at all visible on the screen.

SEE ALSO

Headers/Xm.incTraversable( ), *XmManager* and *XmProcessTraversal*( ).

NAME

XmGetXmDisplay — a Display function that returns the *XmDisplay* object ID for a specified display

SYNOPSIS

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

Widget XmGetXmDisplay(
      Display                  *display);
```

DESCRIPTION

XmGetXmDisplay( ) returns the *XmDisplay* object ID associated with a display. The application can access Display resources with *XtGetValues*( ).

display Specifies the display for which the *XmDisplay* object ID is to be returned.

For a complete definition of Display and its associated resources, see *XmDisplay*.

RETURN VALUE

Returns the *XmDisplay* object ID for the specified display.

SEE ALSO

XmDisplay.

NAME

XmGetXmScreen — a Screen function that returns the *XmScreen* object ID for a specified screen

SYNOPSIS

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

Widget XmGetXmScreen(
      Screen                    *screen);
```

DESCRIPTION

XmGetXmScreen( ) returns the *XmScreen* object ID associated with a screen. The application can access and manipulate Screen resources with *XtGetValues*( ) and *XtSetValues*( ).

screen Specifies the screen for which the *XmScreen* ID is to be returned.

For a complete definition of Screen and its associated resources, see *XmScreen*.

RETURN VALUE

Returns the *XmScreen* object ID.

SEE ALSO

XmScreen.

NAME

XmInstallImage — a pixmap function that adds an image to the image cache

SYNOPSIS

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

Boolean XmInstallImage(
      XImage                        *image,
      char                          *image_name);
```

DESCRIPTION

XmInstallImage( ) stores an image in an image cache that can later be used to generate a pixmap. Part of the installation process is to extend the resource converter used to reference these images. The resource converter is given the image name so that the image can be referenced in a **.Xdefaults** file. Since an image can be referenced by a widget through its pixmap resources, it is up to the application to ensure that the image is installed before the widget is created.

image Points to the image structure to be installed. The installation process does not make a local copy of the image. Therefore, the application should not destroy the image until it is uninstalled from the cache.

image_name Specifies a string that the application uses to name the image. After installation, this name can be used in **.Xdefaults** for referencing the image. A local copy of the name is created by the image cache functions.

The image cache functions provide a set of eight pre-installed images. These names can be used within a **.Xdefaults** file for generating pixmaps for the resource for which they are provided.

Image Name	Description
background	A tile of solid background
25_foreground	A tile of 25% foreground, 75% background
50_foreground	A tile of 50% foreground, 50% background
75_foreground	A tile of 75% foreground, 25% background
horizontal	A tile of horizontal lines of the two colors
vertical	A tile of vertical lines of the two colors
slant_right	A tile of slanting lines of the two colors
slant_left	A tile of slanting lines of the two colors

RETURN VALUE

Returns True when successful; returns False if NULL *image*, NULL *image_name* or duplicate *image_name* is used as an argument value.

SEE ALSO

XmUninstallImage( ), *XmGetPixmap*( ) and *XmDestroyPixmap*( ).

NAME

XmInternAtom — a function that returns an atom for a given name

SYNOPSIS

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

Atom XmInternAtom(
      Display                    *display,
      String                      name,
      Boolean                     only_if_exists);
```

DESCRIPTION

XmInternAtom( ) returns an atom for a given name. It mirrors the Xlib interfaces for atom management, but provides client-side storing in cache. When and where storing in cache is provided in Xlib, the routines will become pseudonyms for the Xlib routines.

display Specifies the connection to the X server.

name Specifies the name associated with the atom you want returned.

only_if_exists
Specifies a Boolean value that indicates whether **XInternAtom** creates the atom.

RETURN VALUE

Returns an atom.

NAME

XmIsMotifWMRunning — a function that determines whether the window manager is running

SYNOPSIS

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

Boolean XmIsMotifWMRunning(
      Widget                    shell);
```

DESCRIPTION

XmIsMotifWMRunning( ) lets a user know whether the Motif Window Manager is running on a screen that contains a specific widget hierarchy. This function first sees whether the _MOTIF_WM_INFO property is present on the root window of the shell's screen. If it is, its window field is used to query for the presence of the specified window as a child of root.

shell Specifies the shell whose screen is tested for *mwm*'s presence.

RETURN VALUE

Returns True if MWM is running.

NAME

XmIsTraversable — a function that identifies whether a widget can be traversed

SYNOPSIS

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

Boolean XmIsTraversable(
      Widget                    widget);
```

DESCRIPTION

XmIsTraversable( ) determines whether the specified widget is eligible to receive focus through keyboard traversal.

RETURN VALUE

Returns True if the widget is eligible to receive focus through keyboard navigation; otherwise, returns False. Note that the returned result assumes that an **XmNtraverseObscuredCallback** procedure succeeds if it must be called to make the target widget visible.

SEE ALSO

XmGetVisibility( ) and *XmProcessTraversal*( ).

NAME

XmLabel — the Label widget class

SYNOPSIS

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

DESCRIPTION

Label is an instantiable widget and is also used as a superclass for other button widgets, such as PushButton and ToggleButton. The Label widget does not accept any button or key input, and the help callback is the only callback defined. Label also receives enter and leave events.

Label can contain either text or a pixmap. Label text is a compound string. Refer to the Programmers' Guide for more information on compound strings. The text can be multilingual, multiline or multifont. When a Label is insensitive, its text is stippled, or the user-supplied insensitive pixmap is displayed.

Label supports both accelerators and mnemonics primarily for use in Label subclass widgets that are contained in menus. Mnemonics are available in a menu system when the button is visible. Accelerators in a menu system are accessible even when the button is not visible. The Label widget displays the mnemonic by underlining the first matching character in the text string. The accelerator is displayed as a text string adjacent to the label text or pixmap.

Label consists of many margin fields surrounding the text or pixmap. These margin fields are resources that may be set by the user, but Label subclasses and Manager parents also modify some of these fields. They tend to modify the **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop**, and **XmNmarginBottom** resources and leave the **XmNmarginWidth** and **XmNmarginHeight** resources as set by the application.

In a Label, **XmNtraversalOn** and **XmNhighlightOnEnter** are forced to False inside Popup MenuPanes, Pulldown MenuPanes, and OptionMenus. Otherwise, these resources default to False.

Classes

Label inherits behavior and resources from *Core* and *XmPrimitive*.

The class pointer is **xmLabelWidgetClass**.

The class name is *XmLabel*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	XmFONTLIST_DEFAULT_TAG	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	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 PushButton and ToggleButton in Pulldown and Popup MenuPanes.

XmNacceleratorText

Specifies the text displayed for the accelerator. The text is displayed adjacent to the label string or pixmap. Accelerator text for buttons is displayed only for PushButtons and ToggleButtons in Pulldown and Popup Menus.

XmNalignment

Specifies the label alignment for text or pixmap.

XmALIGNMENT_BEGINNING (left alignment)

Causes the left sides of the lines of text to be vertically aligned with the left edge of the widget window. For a pixmap, its left side is vertically aligned with the left edge of the widget window.

XmALIGNMENT_CENTER (center alignment)

Causes the centers of the lines of text to be vertically aligned in the center of the widget window. For a pixmap, its center is vertically aligned with the center of the widget window.

XmALIGNMENT_END (right alignment)

Causes the right sides of the lines of text to be vertically aligned with the right edge of the widget window. For a pixmap, its right side is vertically aligned with the right edge of the widget window.

The preceding descriptions for text are correct when **XmNstringDirection** is XmSTRING_DIRECTION_L_TO_R. When that resource is XmSTRING_DIRECTION_R_TO_L, the descriptions for XmALIGNMENT_BEGINNING and XmALIGNMENT_END are switched.

If the parent is a RowColumn whose **XmNisAligned** resource is True, **XmNalignment** is forced to the same value as the RowColumn's **XmNentryAlignment** if the RowColumn's **XmNrowColumnType** is XmWORK_AREA or if the widget is a subclass of XmLabel. Otherwise, the default is XmALIGNMENT_CENTER.

XmNfontList

Specifies the font of the text used in the widget. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or XmMenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNbuttonFontList** (for button subclasses) or **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList** 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.

XmNlabelPixmap

Specifies the pixmap when **XmNlabelType** is XmPIXMAP.

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** for more information on the creation and structure of compound strings.

XmNlabelType

Specifies the label type:

XmSTRING

Text displays **XmNlabelString**.

XmPIXMAP

Icon data in pixmap displays **XmNlabelInsensitivePixmap** or **XmNlabelPixmap**.

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 the amount of spacing between the top of the label (specified by **XmNmarginTop**) and the bottom edge of the top shadow, and the amount of spacing between the bottom of the label (specified by **XmNmarginBottom**) and the top edge of the bottom shadow.

XmNmarginLeft

Specifies the amount of spacing between the left edge of the label text and the right side of the left margin (specified by **XmNmarginWidth**). This may be modified by Label's subclasses. For example, ToggleButton may increase this field to make room for the toggle indicator and for spacing between the indicator and label. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginRight

Specifies the amount of spacing between the right edge of the label text and the left side of the right margin (specified by **XmNmarginWidth**). This may be modified by Label's subclasses. For example, CascadeButton may increase this field to make room for the cascade pixmap. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginTop

Specifies the amount of spacing between the top of the label text and the bottom of the top margin (specified by **XmNmarginHeight**). This may be modified by Label's subclasses. For example, CascadeButton may increase this field to make room for the cascade pixmap.

XmNmarginWidth

Specifies the amount of spacing between the left side of the label (specified by **XmNmarginLeft**) and the right edge of the left shadow, and the amount of spacing between the right side of the label (specified by **XmNmarginRight**) and the left edge of the right shadow.

XmNmnemonic

Provides the user with an alternative means of selecting a button. A button in a MenuBar, a Popup MenuPane, or a Pulldown MenuPane can have a mnemonic.

This resource contains a keysym as listed in the X11 keysym table. The first character in the label string that exactly matches the mnemonic in the character set specified in **XmNmnemonicCharSet** is underlined when the button is displayed.

When a mnemonic has been specified, the user activates the button by pressing the mnemonic key while the button is visible. If the button is a CascadeButton in a MenuBar and the MenuBar does not have the focus, the user must use the **MAlt** modifier while pressing the mnemonic. The user can activate the button by pressing either the shifted or the unshifted mnemonic key.

XmNmnemonicCharSet

Specifies the character set of the mnemonic for the label. The default is XmFONTLIST_DEFAULT_TAG.

XmNrecomputeSize

Specifies a Boolean value that indicates whether the widget attempts to be big enough to contain the label. If True, an *XtSetValues*( ) with a resource value that would change the size of the widget causes the widget to shrink or expand to fit the label string or pixmap exactly. If False, the widget never attempts to change size on its own.

XmNstringDirection

Specifies the direction in which the string is to be drawn:

XmSTRING_DIRECTION_L_TO_R
left to right

XmSTRING_DIRECTION_R_TO_L
right to left.

The default for this resource is determined at creation time. If no value is specified for this resource and the widget's parent is a manager, the value is inherited from the parent; otherwise, it defaults to XmSTRING_DIRECTION_L_TO_R.

Inherited Resources

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

XmPrimitive Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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

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, 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, this action disarms the current menu item and arms the item below it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's bottom edge, this action wraps to the topmost menu item in the column to the right, if one exists. When the current menu item is at the bottom, rightmost corner of the MenuPane, this action wraps to the tear-off control, if present, or to the top, leftmost menu item.

MenuTraverseLeft( )

When the current menu item is in a MenuBar, 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, 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, this action unposts the current submenu only.

In Popup or Torn-off MenuPanes, when the current menu item is at the left edge, this action wraps within the MenuPane. If the current menu item is at the left edge of the MenuPane and not in the top row, this action wraps to the rightmost menu item in the row above. If the current menu item is in the upper, leftmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, rightmost menu item in the MenuPane.

MenuTraverseRight( )

If the current menu item is in a MenuBar, 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, 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, this action unposts all submenus and traverses to the MenuBar item to the right. If that MenuBar item has a submenu, it posts the submenu and arms the submenu's first traversable item.

In Popup or Torn-off menus, if the current menu item is not a CascadeButton and is at the right edge of a row (except the bottom row), this action wraps to the leftmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, rightmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, leftmost menu item of the MenuPane.

MenuTraverseUp( )

When the current menu item is in a MenuPane, this action disarms the current menu item and arms the item above it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's top edge, this action wraps to the bottommost menu item in the column to the left, if one exists. When the current menu item is at the top, leftmost corner of the MenuPane, this action wraps to the tear-off control, if present, or to the bottom, rightmost menu item.

SEE ALSO

Core, *XmCreateLabel*( ), *XmFontListAppendEntry*( ), *XmStringCreate*( ) and *XmPrimitive*.

NAME

XmLabelGadget — the LabelGadget widget class

SYNOPSIS

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

DESCRIPTION

LabelGadget is an instantiable widget and is also used as a superclass for other button gadgets, such as PushButtonGadget and ToggleButtonGadget.

LabelGadget can contain either text or a pixmap. LabelGadget text is a compound string. Refer to the Programmers' Guide for more information on compound strings. The text can be multilingual, multiline or multifont. When a LabelGadget is insensitive, its text is stippled, or the user-supplied insensitive pixmap is displayed.

LabelGadget supports both accelerators and mnemonics primarily for use in LabelGadget subclass widgets that are contained in menus. Mnemonics are available in a menu system when the button is visible. Accelerators in a menu system are accessible even when the button is not visible. The LabelGadget displays the mnemonic by underlining the first matching character in the text string. The accelerator is displayed as a text string adjacent to the label text or pixmap.

LabelGadget consists of many margin fields surrounding the text or pixmap. These margin fields are resources that may be set by the user, but LabelGadget subclasses and Manager parents also modify some of these fields. They tend to modify the **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop** and **XmNmarginBottom** resources and leave the **XmNmarginWidth** and **XmNmarginHeight** resources as set by the application.

In a LabelGadget, **XmNtraversalOn** and **XmNhighlightOnEnter** are forced to False inside Popup MenuPanes, Pulldown MenuPanes and OptionMenus. Otherwise these resources default to False.

Classes

LabelGadget inherits behavior and resources from *Object*, *RectObj* and *XmGadget*.

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 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).

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
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	dynamic	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	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 PushButtonGadget and ToggleButtonGadget in Pulldown and Popup menus.

XmNacceleratorText

Specifies the text displayed for the accelerator. The text is displayed adjacent to the label string or pixmap. Accelerator text for buttons is displayed only for PushButtonGadgets and ToggleButtonGadgets in Pulldown and Popup Menus.

XmNalignment

Specifies the label alignment for text or pixmap.

XmALIGNMENT_BEGINNING (left alignment)

Causes the left sides of the lines of text to be vertically aligned with the left edge of the gadget. For a pixmap, its left side is vertically aligned with the left edge of the gadget.

XmALIGNMENT_CENTER (center alignment)

Causes the centers of the lines of text to be vertically aligned in the center of the gadget. For a pixmap, its center is vertically aligned with the center of the gadget.

XmALIGNMENT_END (right alignment)

Causes the right sides of the lines of text to be vertically aligned with the right edge of the gadget. For a pixmap, its right side is vertically aligned with the right edge of the gadget.

The preceding descriptions for text are correct when **XmNstringDirection** is XmSTRING_DIRECTION_L_TO_R; the descriptions for XmALIGNMENT_BEGINNING and XmALIGNMENT_END are switched when the resource is XmSTRING_DIRECTION_R_TO_L.

If the parent is a RowColumn whose **XmNisAligned** resource is True, **XmNalignment** is forced to the same value as the RowColumn's **XmNentryAlignment** if the RowColumn's **XmNrowColumnType** is XmWORK_AREA or if the gadget is a subclass of *XmLabelGadget*. Otherwise, the default is XmALIGNMENT_CENTER.

XmNfontList
Specifies the font of the text used in the gadget. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the *XmBulletinBoard*, *VendorShell* or *XmMenuShell* widget class. If such an ancestor is found, the font list is initialized to the **XmNbuttonFontList** (for button gadget subclasses) or **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList** 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.

XmNlabelPixmap
Specifies the pixmap when **XmNlabelType** is XmPIXMAP.

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 Section 4.3 on page 61 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 **XmNlabelInsensitivePixmap** or **XmNlabelPixmap**.

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 the amount of spacing between the top of the label (specified by **XmNmarginTop**) and the bottom edge of the top shadow, and the amount of spacing between the bottom of the label (specified by **XmNmarginBottom**) and the top edge of the bottom shadow.

XmNmarginLeft
Specifies the amount of spacing between the left edge of the label text and the right side of the left margin (specified by **XmNmarginWidth**). This may be modified by LabelGadget's subclasses. For example, ToggleButtonGadget may increase this field to make room for the toggle indicator and for spacing between the indicator and label. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginRight
Specifies the amount of spacing between the right edge of the label text and the left side of the right margin (specified by **XmNmarginWidth**). This may be modified by LabelGadget's

subclasses. For example, CascadeButtonGadget may increase this field to make room for the cascade pixmap. Whether this actually applies to the left or right side of the label may depend on the value of **XmNstringDirection**.

XmNmarginTop

Specifies the amount of spacing between the top of the label text and the bottom of the top margin (specified by **XmNmarginHeight**). This may be modified by LabelGadget's subclasses. For example, CascadeButtonGadget may increase this field to make room for the cascade pixmap.

XmNmarginWidth

Specifies the amount of spacing between the left side of the label (specified by **XmNmarginLeft**) and the right edge of the left shadow, and the amount of spacing between the right side of the label (specified by **XmNmarginRight**) and the left edge of the right shadow.

XmNmnemonic

Provides the user with an alternative means of selecting a button. A button in a MenuBar, a Popup MenuPane or a Pulldown MenuPane can have a mnemonic.

This resource contains a keysym as listed in the X11 keysym table. The first character in the label string that exactly matches the mnemonic in the character set specified in **XmNmnemonicCharSet** is underlined when the button is displayed.

When a mnemonic has been specified, the user activates the button by pressing the mnemonic key while the button is visible. If the button is a CascadeButtonGadget in a MenuBar and the MenuBar does not have the focus, the user must use the **MAlt** modifier while pressing the mnemonic. The user can activate the button by pressing either the shifted or the unshifted mnemonic key.

XmNmnemonicCharSet

Specifies the character set of the mnemonic for the label. The default is XmFONTLIST_DEFAULT_TAG.

XmNrecomputeSize

Specifies a Boolean value that indicates whether the gadget attempts to be big enough to contain the label. If True, an *XtSetValues* with a resource value that would change the size of the gadget causes the gadget to shrink or expand to fit the label string or pixmap exactly. If False, the gadget never attempts to change size on its own.

XmNstringDirection

Specifies the direction in which the string is to be drawn.

XmSTRING_DIRECTION_L_TO_R

Left to right

XmSTRING_DIRECTION_R_TO_L

Right to left

The default for this resource is determined at creation time. If no value is specified for this resource and the widget's parent is a manager, the value is inherited from the parent; otherwise, it defaults to XmSTRING_DIRECTION_L_TO_R.

Inherited Resources

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

XmGadget **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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

SEE ALSO

Object, *RectObj*, *XmCreateLabelGadget*( ), *XmStringCreate*( ) and *XmGadget*.

NAME

XmList — the List widget class

SYNOPSIS

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

DESCRIPTION

List allows a user to select one or more items from a group of choices. Items are selected from the list in a variety of ways, using both the pointer and the keyboard. List operates on an array of compound strings defined by the application. Each compound string becomes an item in the List, with the first compound string becoming the item in position 1, the second becoming the item in position 2, and so on.

Specifying the number of items that are visible sets the size of the List. If the number of visible items is not specified, the height of the list controls the number of visible items. Each item assumes the height of the tallest element in the list. To create a list that allows the user to scroll easily through a large number of items, use the *XmCreateScrolledList*( ) convenience function.

To select items, move the pointer or cursor to the desired item and press the **BSelect** mouse button or the key defined as **KSelect**. There are several styles of selection behavior, and they all highlight the selected item or items by displaying them in inverse colors. An appropriate callback is invoked to notify the application of the user's choice. The application then takes whatever action is required for the specified selection.

Selection

Each list has one of four selection models:

- Single Select
- Browse Select
- Multiple Select
- Extended Select.

In Single Select and Browse Select, at most one item is selected at a time. In Single Select, pressing **BSelect** on an item toggles its selection state and deselects any other selected item. In Browse Select, pressing **BSelect** on an item selects it and deselects any other selected item; dragging **BSelect** moves the selection along with the cursor.

In Multiple Select, any number of items can be selected at a time. Pressing **BSelect** on an item toggles its selection state but does not deselect any other selected items.

In Extended Select, any number of items can be selected at a time, and the user can easily select ranges of items. Pressing **BSelect** on an item selects it and deselects any other selected item. Dragging **BSelect** or pressing or dragging **BExtend** following a **BSelect** action selects all items between the item under the pointer and the item on which **BSelect** was pressed. This action also deselects any other selected items outside that range.

Extended Select also allows the user to select and deselect discontiguous ranges of items. Pressing **BToggle** on an item toggles its selection state but does not deselect any other selected items. Dragging **BToggle** or pressing or dragging **BExtend** following a **BToggle** action sets the selection state of all items between the item under the pointer and the item on which **BToggle** was pressed to the state of the item on which **BToggle** was pressed. This action does not deselect any other selected items outside that range.

All selection operations available from the mouse are also available from the keyboard. List has two keyboard selection modes, Normal Mode and Add Mode. In Normal Mode, navigation operations and **KSelect** select the item at the location cursor and deselect any other selected items. In Add Mode, navigation operations have no effect on selection, and **KSelect** toggles the selection state of the item at the location cursor without deselecting any other selected items, except in Single Select.

Single and Multiple Select use Add Mode; Browse Select uses Normal Mode.

Extended Select can use either mode; the user changes modes by pressing **KAddMode**. In Extended Select Normal Mode, pressing **KSelect** has the same effect as pressing **BSelect**; **KExtend** and shifted navigation have the same effect as pressing **BExtend** following a **BSelect** action. In Extended Select Add Mode, pressing **KSelect** has the same effect as pressing **BToggle**; **KExtend** and shifted navigation have the same effect as pressing **BExtend** following a **BToggle** action.

Normal Mode is indicated by a solid location cursor, and Add Mode is indicated by a dashed location cursor.

Classes

List inherits behavior and resources from *Core* and *XmPrimitive*.

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 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).

XmList **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNautomaticSelection	**XmCAutomaticSelection**	**Boolean**	False	CSG
XmNbrowseSelectionCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNdefaultActionCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNextendedSelectionCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNfontList	**XmCFontList**	**XmFontList**	dynamic	CSG
XmNitemCount	**XmCItemCount**	**int**	0	CSG
XmNitems	**XmCItems**	**XmStringTable**	NULL	CSG
XmNlistMarginHeight	**XmCListMarginHeight**	**Dimension**	0	CSG
XmNlistMarginWidth	**XmCListMarginWidth**	**Dimension**	0	CSG
XmNlistSizePolicy	**XmCListSizePolicy**	**unsigned char**	XmVARIABLE	CG
XmNlistSpacing	**XmCListSpacing**	**Dimension**	0	CSG
XmNmultipleSelectionCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNscrollBarDisplayPolicy	**XmCScrollBarDisplayPolicy**	**unsigned char**	XmAS_NEEDED	CSG
XmNselectedItemCount	**XmCSelectedItemCount**	**int**	0	CSG
XmNselectedItems	**XmCSelectedItems**	**XmStringTable**	NULL	CSG
XmNselectionPolicy	**XmCSelectionPolicy**	**unsigned char**	XmBROWSE _SELECT	CSG
XmNsingleSelectionCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CSG
XmNtopItemPosition	**XmCTopItemPosition**	**int**	1	CSG
XmNvisibleItemCount	**XmCVisibleItemCount**	**int**	dynamic	CSG

XmNautomaticSelection

Invokes either **XmNbrowseSelectionCallback** or **XmNextendedSelectionCallback** when BSelect is pressed and the items that are shown as selected change if the value is True and the selection mode is either XmBROWSE_SELECT or XmEXTENDED_SELECT respectively. If False, no selection callbacks are invoked until the user releases the mouse button. See **Action Routines** on page 382 for further details on the interaction of this resource with the selection modes.

XmNbrowseSelectionCallback

Specifies a list of callbacks called when an item is selected in the browse selection mode. The reason is XmCR_BROWSE_SELECT.

XmNdefaultActionCallback

Specifies a list of callbacks called when an item is double clicked. The reason is XmCR_DEFAULT_ACTION.

XmNextendedSelectionCallback

Specifies a list of callbacks called when items are selected using the extended selection mode. The reason is XmCR_EXTENDED_SELECT.

XmNfontList

Specifies the font list associated with the list items. This is used in conjunction with the **XmNvisibleItemCount** resource to determine the height of the List widget. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the *XmBulletinBoard* or *VendorShell* widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList** 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 Section 4.3 on page 61 for more information on the creation and structure of compound strings.

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 the 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 the 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.

XmNmultipleSelectionCallback

Specifies a list of callbacks called when an item is selected in multiple selection mode. The reason is XmCR_MULTIPLE_SELECT.

XmNscrollBarDisplayPolicy

Controls the display of vertical ScrollBars in a list whose parent is the 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.

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 currently selected, either by the user or by the application.

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 capability.

XmNsingleSelectionCallback

Specifies a list of callbacks called when an item is selected in single selection mode. The reason is XmCR_SINGLE_SELECT.

XmNstringDirection

Specifies the initial direction to draw the string. The values for this resource are XmSTRING_DIRECTION_L_TO_R and XmSTRING_DIRECTION_R_TO_L. The value of this resource is determined at creation time. If the widget's parent is a manager, this value is inherited from the widget's parent; otherwise it is set to XmSTRING_DIRECTION_L_TO_R.

XmNtopItemPosition

Specifies the position of the item that is the first visible item in the list. Setting this resource is equivalent to calling the *XmListSetPos*( ) function. The position of the first item in the list is 1; the position of the second item is 2; and so on. A position of 0 (zero) specifies the last item in the list. The value must not be negative.

XmNvisibleItemCount

Specifies the number of items that can fit in the visible space of the list work area. The List uses this value to determine its height. The value must be greater than 0 (zero).

Inherited Resources

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

XmPrimitive **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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

List defines a new callback structure. The application must first look at the reason field and use only the structure members that are valid for that particular reason, because not all fields are relevant for every possible reason. The callback structure is defined as follows:

```
typedef struct
{
      int                        reason;
      XEvent                    *event;
      XmString                   item;
      int                        item_length;
      int                        item_position;
      XmString                  *selected_items;
      int                        selected_item_count;
      int                       *selected_item_positions;
      char                       selection_type;
} XmListCallbackStruct;
```

reason Indicates why the callback was invoked.

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

item The last item selected at the time of the **event** that caused the callback. **item** points to a temporary storage space that is reused after the callback is finished. Therefore, if an application needs to save the item, it should copy the item into its own data space.

item_length The length in bytes of **item**.

item_position
The position plus one of **item** in the List's **XmNitems** array. An **item_position** of 1 symbolizes the first element in the list.

selected_items
A list of items selected at the time of the **event** that caused the callback. The member **selected_items** points to a temporary storage space which 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 which 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 non-contiguous selection (XmADDITION).

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

Reason	Valid Fields
XmCR_SINGLE_SELECT	**reason, event, item, item_length, item_position**
XmCR_DEFAULT_ACTION	**reason, event, item, item_length, item_position selected_items, selected_item_count, selected_item_positions**
XmCR_BROWSE_SELECT	**reason, event, item, item_length, item_position**
XmCR_MULTIPLE_SELECT	**reason, event, item, item_length, item_position, selected_items, selected_item_count, selected_item_positions**
XmCR_EXTENDED_SELECT	**reason, event, item, item_length, item_position, selected_items, selected_item_count, selected_item_positions, selection_type**

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 performed; changes the selection state of the first item and all items between it and the current anchor point to the state of the item at the current anchor point; calls the **XmNextendedSelectionCallback** callbacks.

ListBeginExtend( )

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action does the following: if an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done, and changes the selection state of the item under the pointer and all items between it and the current anchor point to the state of the item at the current anchor point. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListBeginLine( )

Moves the horizontal scroll region to the beginning of the line.

ListBeginSelect( )

If **XmNselectionPolicy** is set to XmSINGLE_SELECT, deselects any current selection and toggles the selection state of the item under the pointer.

If **XmNselectionPolicy** is set to XmBROWSE_SELECT, deselects any current selection and selects the item under the pointer. If **XmNautomaticSelection** is set to True, calls the **XmNbrowseSelectionCallback** callbacks.

If **XmNselectionPolicy** is set to XmMULTIPLE_SELECT, toggles the selection state of the item under the pointer. Any previous selections remain.

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action deselects any current selection, selects the item under the pointer, and sets the current anchor at that item. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListBeginToggle( )

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action moves the current anchor to the item under the pointer without changing the current selection. If the item is unselected, this action selects it; if the item is selected, this action unselects it. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListButtonMotion( )

If **XmNselectionPolicy** is set to XmBROWSE_SELECT, this action deselects any current selection and selects the item under the pointer. If **XmNautomaticSelection** is set to True and the pointer has entered a new list item, this action calls the **XmNbrowseSelectionCallback** callbacks.

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action does the following: if an extended selection is being made and an extended selection has previously been made from the current anchor point, restores the selection state of the items in that range to their state before the previous extended selection was done and changes the selection state of the item under the pointer and all items between it and the current anchor point to the state of the item at the current anchor point. If **XmNautomaticSelection** is set to True and the pointer has entered a new list item, calls the **XmNextendedSelectionCallback** callbacks.

If the pointer leaves a scrolled list, this action scrolls the list in the direction of the pointer motion.

ListEndData( )

Moves the location cursor to the last item in the list. In Normal Mode, this also deselects any current selection, selects the last item in the list, and calls the appropriate selection callbacks (**XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to XmBROWSE_SELECT, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to XmEXTENDED_SELECT).

ListEndDataExtend( )

If **XmNselectionPolicy** is set to XmMULTIPLE_SELECT or XmEXTENDED_SELECT, this action moves the location cursor to the last item in the list.

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action does the following: If an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was done; changes the selection state of the last item and all items between it and the current anchor point to the state of the item at the current anchor point; calls the **XmNextendedSelectionCallback** callbacks.

ListEndExtend( )

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action moves the location cursor to the last item selected or deselected and, if **XmNautomaticSelection** is set to False, calls the **XmNextendedSelectionCallback** callbacks.

ListEndLine( )

Moves the horizontal scroll region to the end of the line.

ListEndSelect( )

If **XmNselectionPolicy** is set to XmSINGLE_SELECT or XmMULTIPLE_SELECT, this action moves the location cursor to the last item selected or deselected and calls the appropriate selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to XmSINGLE_SELECT, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to XmMULTIPLE_SELECT).

If **XmNselectionPolicy** is set to XmBROWSE_SELECT or XmEXTENDED_SELECT, moves the location cursor to the last item selected or deselected and, if **XmNautomaticSelection** is set to False, calls the appropriate selection callbacks (**XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to XmBROWSE_SELECT, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to XmEXTENDED_SELECT).

ListEndToggle( )

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, moves the location cursor to the last item selected or deselected and, if **XmNautomaticSelection** is set to False, calls the **XmNextendedSelectionCallback** callbacks.

ListExtendNextItem( )

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action does the following: if an extended selection has been made from the current anchor point, restores the selection state of the items in that range to their state before the extended selection was performed;

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 performed; 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.

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 performed; changes the selection state of the item at the location cursor and all items between it and the current anchor point to the state of the item at the current anchor point. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListKbdBeginSelect( )

If the **XmNselectionPolicy** is set to XmSINGLE_SELECT, deselects any current selection and toggles the state of the item at the location cursor.

If the **XmNselectionPolicy** is set to XmBROWSE_SELECT, deselects any current selection and selects the item at the location cursor. If **XmNautomaticSelection** is set to True, calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to XmMULTIPLE_SELECT, toggles the selection state of the item at the location cursor. Any previous selections remain.

If the **XmNselectionPolicy** is set to XmEXTENDED_SELECT, moves the current anchor to the item at the location cursor. In Normal Mode, this action deselects any current selection and selects the item at the location cursor. In Add Mode, this action toggles the selection state of the item at the location cursor and leaves the current selection unchanged. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks.

ListKbdCancel( )

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT and an extended selection is being made from the current anchor point, this action cancels the new selection and restores the selection state of the items in that range to their state before the extended selection was performed. If **XmNautomaticSelection** is set to True, this action calls the **XmNextendedSelectionCallback** callbacks; otherwise, if the parent is a manager, it passes the event to the parent.

ListKbdDeSelectAll( )

If the **XmNselectionPolicy** is set to XmSINGLE_SELECT, XmMULTIPLE_SELECT, or XmEXTENDED_SELECT in Add Mode, this action deselects all items in the list. If the **XmNselectionPolicy** is set to XmEXTENDED_SELECT in Normal Mode, this action deselects all items in the list (except the item at the location cursor if the shell's **XmNkeyboardFocusPolicy** is XmEXPLICIT). This action also calls the appropriate selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to XmSINGLE_SELECT, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to XmMULTIPLE_SELECT, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to XmEXTENDED_SELECT).

ListKbdEndExtend( )

If **XmNselectionPolicy** is set to XmEXTENDED_SELECT, and if **XmNautomaticSelection** is set to False, this action calls the **XmNextendedSelectionCallback** callbacks.

ListKbdEndSelect( )

If **XmNselectionPolicy** is set to XmSINGLE_SELECT or XmMULTIPLE_SELECT or if **XmNautomaticSelection** is set to False, calls the appropriate selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to XmSINGLE_SELECT, **XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to XmBROWSE_SELECT, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to XmMULTIPLE_SELECT, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to XmEXTENDED_SELECT).

ListKbdSelectAll( )

If **XmNselectionPolicy** is set to XmSINGLE_SELECT or XmBROWSE_SELECT, this action selects the item at the location cursor. If **XmNselectionPolicy** is set to XmEXTENDED_SELECT or XmMULTIPLE_SELECT, it selects all items in the list. This action also calls the appropriate selection callbacks (**XmNsingleSelectionCallback** when **XmNselectionPolicy** is set to XmSINGLE_SELECT, **XmNbrowseSelectionCallback** when **XmNselectionPolicy** is set to XmBROWSE_SELECT, **XmNmultipleSelectionCallback** when **XmNselectionPolicy** is set to XmMULTIPLE_SELECT, **XmNextendedSelectionCallback** when **XmNselectionPolicy** is set to XmEXTENDED_SELECT).

ListLeftChar( )

Scrolls the list one character to the left.

ListLeftPage( )

Scrolls the list one page to the left.

ListNextItem( )

Moves the location cursor to the next item in the list.

If the **XmNselectionPolicy** is set to XmBROWSE_SELECT, this action also selects the next item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action in Normal Mode also selects the next item, deselects any current selection, moves the current anchor to the next item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode, this action does not affect the selection or the anchor.

ListNextPage( )

Scrolls the list to the next page, moving the location cursor to a new item.

If the **XmNselectionPolicy** is set to XmBROWSE_SELECT, this action also selects the new item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action in Normal Mode also selects the new item, deselects any current selection, moves the current anchor to the new item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode, this action does not affect the selection or the anchor.

ListPrevItem( )

Moves the location cursor to the previous item in the list.

If the **XmNselectionPolicy** is set to XmBROWSE_SELECT, this action also selects the previous item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action in Normal Mode also selects the previous item, deselects any current selection, moves the current anchor to the previous item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode, this action does not affect the selection or the anchor.

ListPrevPage( )

Scrolls the list to the previous page, moving the location cursor to a new item.

If the **XmNselectionPolicy** is set to XmBROWSE_SELECT, this action also selects the new item, deselects any current selection, and calls the **XmNbrowseSelectionCallback** callbacks.

If the **XmNselectionPolicy** is set to XmEXTENDED_SELECT, this action in Normal Mode also selects the new item, deselects any current selection, moves the current anchor to the new item, and calls the **XmNextendedSelectionCallback** callbacks. In Add Mode this action does not affect the selection or the anchor.

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.

SEE ALSO

Core, *XmCreateList*( ), *XmCreateScrolledList*( ), *XmFontListAppendEntry*( ), *XmListAddItem*( ), *XmListAddItems*( ), *XmListAddItemUnselected*( ), *XmListDeleteAllItems*( ), *XmListDeleteItem*( ), *XmListDeleteItems*( ), *XmListDeleteItemsPos*( ), *XmListDeletePos*( ), *XmListDeselectAllItems*( ), *XmListDeselectItem*( ), *XmListDeselectPos*( ), *XmListGetMatchPos*( ), *XmListGetSelectedPos*( ), *XmListItemExists*( ), *XmListItemPos*( ), *XmListReplaceItems*( ), *XmListReplaceItemsPos*( ), *XmListSelectItem*( ), *XmListSelectPos*( ), *XmListSetAddMode*( ), *XmListSetBottomItem*( ), *XmListSetBottomPos*( ), *XmListSetHorizPos*( ), *XmListSetItem*( ), *XmListSetPos*( ), *XmPrimitive* and *XmStringCreate*( ).

NAME

XmListAddItem — a List function that adds an item to the list

SYNOPSIS

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

void XmListAddItem(
      Widget                    widget,
      XmString                  item,
      int                       position);
```

DESCRIPTION

XmListAddItem( ) adds an item to the list at the given position. When the item is inserted into the list, it is compared with the current **XmNselectedItems** list. If the new item matches an item on the selected list, it appears selected.

widget Specifies the ID of the List to which an item is added.

item Specifies the item to be added to the list.

position Specifies the position of the new item in the list. A value of 1 makes the new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the new item the last item in the list.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListAddItemUnselected — a List function that adds an item to the list

SYNOPSIS

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

void XmListAddItemUnselected(
      Widget                    widget,
      XmString                  item,
      int                       position);
```

DESCRIPTION

XmListAddItemUnselected( ) adds an item to the list at the given position. The item does not appear selected, even if it matches an item in the current **XmNselectedItems** list.

widget Specifies the ID of the List from whose list an item is added.

item Specifies the item to be added to the list.

position Specifies the position of the new item in the list. A value of 1 makes the new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the new item the last item in the list.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListAddItems — a List function that adds items to the list

SYNOPSIS

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

void XmListAddItems(
      Widget                     widget,
      XmString                  *items,
      int                        item_count,
      int                        position);
```

DESCRIPTION

XmListAddItems( ) adds the specified items to the list at the given position. The first *item_count* items of the *items* array are added to the list. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. If any of the new items matches an item on the selected list, it appears selected.

widget Specifies the ID of the List to which an item is added.

items Specifies a pointer to the items to be added to the list.

item_count Specifies the number of items in *items*. This number must be non-negative.

position Specifies the position of the first new item in the list. A value of 1 makes the first new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the first new item follow the last item in the list.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListAddItemsUnselected — a List function that adds items to a list

SYNOPSIS

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

void XmListAddItemsUnselected(
      Widget                    widget,
      XmString                 *items,
      int                       item_count,
      int                       position);
```

DESCRIPTION

XmListAddItemsUnselected( ) adds the specified items to the list at the given position. The inserted items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget Specifies the ID of the List widget to add items to.

items Specifies a pointer to the items to be added to the list.

item_count Specifies the number of elements in *items*. This number must be non-negative.

position Specifies the position of the first new item in the list. A value of 1 makes the first new item the first item in the list; a value of 2 makes it the second item; and so on. A value of 0 (zero) makes the first new item follow the last item of the list.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeleteAllItems — a List function that deletes all items from the list

SYNOPSIS

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

void XmListDeleteAllItems(
      Widget                    widget);
```

DESCRIPTION

XmListDeleteAllItems( ) deletes all items from the list.

widget Specifies the ID of the List from whose list the items are deleted.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeleteItem — a List function that deletes an item from the list

SYNOPSIS

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

void XmListDeleteItem(
      Widget                    widget,
      XmString                  item);
```

DESCRIPTION

XmListDeleteItem( ) deletes the first item in the list that matches *item*. A warning message appears if the item does not exist.

widget Specifies the ID of the List from whose list an item is deleted.

item Specifies the text of the item to be deleted from the list.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeleteItems — a List function that deletes items from the list

SYNOPSIS

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

void XmListDeleteItems(
      Widget                     widget,
      XmString                  *items,
      int                        item_count);
```

DESCRIPTION

XmListDeleteItems( ) deletes the specified items from the list. For each element of *items*, the first item in the list that matches that element is deleted. A warning message appears if any of the items do not exist.

widget Specifies the ID of the List from whose list an item is deleted.

items Specifies a pointer to items to be deleted from the list.

item_count Specifies the number of elements in *items*. This number must be non-negative.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeleteItemsPos — a List function that deletes items from the list starting at the given position

SYNOPSIS

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

void XmListDeleteItemsPos(
      Widget                    widget,
      int                       item_count,
      int                       position);
```

DESCRIPTION

XmListDeleteItemsPos( ) deletes the specified number of items from the list starting at the specified position.

widget Specifies the ID of the List from whose list an item is deleted.

item_count Specifies the number of items to be deleted. This number must be non-negative.

position Specifies the position in the list of the first item to be deleted. A value of 1 indicates that the first deleted item is the first item in the list; a value of 2 indicates that it is the second item; and so on.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeletePos — a List function that deletes an item from a list at a specified position

SYNOPSIS

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

void XmListDeletePos(
      Widget                    widget,
      int                       position);
```

DESCRIPTION

XmListDeletePos( ) deletes an item at a specified position. A warning message appears if the position does not exist.

widget Specifies the ID of the List from which an item is to be deleted.

position Specifies the position of the item to be deleted. A value of 1 indicates that the first item in the list is deleted; a value of 2 indicates that the second item is deleted; and so on. A value of 0 (zero) indicates that the last item in the list is deleted.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeletePositions — a List function that deletes items from a list based on an array of positions

SYNOPSIS

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

void XmListDeletePositions(
      Widget                     widget,
      int                       *position_list,
      int                        position_count);
```

DESCRIPTION

XmListDeletePositions( ) deletes non-contiguous items from a list. The function deletes all items whose corresponding positions appear in the *position_list* array. A warning message is displayed if a specified position is invalid; that is, the value is 0, a negative integer or a number greater than the number of items in the list.

widget Specifies the ID of the List widget.

position_list Specifies an array of the item positions to be deleted. The position of the first item in the list is 1; the position of the second item is 2; and so on.

position_count
Specifies the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeselectAllItems — a List function that unhighlights and removes all items from the selected list

SYNOPSIS

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

void XmListDeselectAllItems(
      Widget                    widget);
```

DESCRIPTION

XmListDeselectAllItems( ) unhighlights and removes all items from the selected list.

widget Specifies the ID of the List widget from whose list all selected items are deselected.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeselectItem — a List function that deselects the specified item from the selected list

SYNOPSIS

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

void XmListDeselectItem(
      Widget                      widget,
      XmString                    item);
```

DESCRIPTION

XmListDeselectItem( ) unhighlights and removes from the selected list the first item in the list that matches *item*.

widget Specifies the ID of the List from whose list an item is deselected.

item Specifies the item to be deselected from the list.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListDeselectPos — a List function that deselects an item at a specified position in the list

SYNOPSIS

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

void XmListDeselectPos(
      Widget                    widget,
      int                       position);
```

DESCRIPTION

XmListDeselectPos( ) unhighlights the item at the specified position and deletes it from the list of selected items.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be deselected. A value of 1 indicates that the first item in the list is deselected; a value of 2 indicates that the second item is deselected; and so on. A value of 0 (zero) indicates that the last item in the list is deselected.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListGetKbdItemPos — a List function that returns the position of the item at the location cursor

SYNOPSIS

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

int XmListGetKbdItemPos(
      Widget                    widget);
```

DESCRIPTION

XmListGetKbdItemPos( ) returns the position of the list item at the location cursor.

widget Specifies the ID of the List widget.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns the position of the current keyboard item. A value of 1 indicates that the location cursor is at the first item of the list; a value of 2 indicates that it is at the second item; and so on. A value of 0 (zero) indicates the List widget is empty.

SEE ALSO

XmList.

NAME

XmListGetMatchPos — a List function that returns all instances of an item in the list.

SYNOPSIS

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

Boolean XmListGetMatchPos(
      Widget                     widget,
      XmString                   item,
      int                      **position_list,
      int                       *position_count);
```

DESCRIPTION

XmListGetMatchPos( ) is a Boolean function that returns an array of positions where a specified item is found in a List.

widget Specifies the ID of the List widget.

item Specifies the item to search for.

position_list Returns an array of positions at which the item occurs in the List. The position of the first item in the list is 1; the position of the second item is 2; and so on. When the return value is True, *XmListGetMatchPos*( ) allocates memory for this array. The caller is responsible for freeing this memory.

position_count
Returns the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns True if the specified item is present in the list, and False if it is not.

SEE ALSO

XmList.

NAME

XmListGetSelectedPos — a List function that returns the position of every selected item in the list

SYNOPSIS

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

Boolean XmListGetSelectedPos(
      Widget                    widget,
      int                     **position_list,
      int                      *position_count);
```

DESCRIPTION

XmListGetSelectedPos( ) is a Boolean function that returns an array of the positions of the selected items in a List.

widget Specifies the ID of the List widget.

position_list Returns an array of the positions of the selected items in the List. The position of the first item in the list is 1; the position of the second item is 2; and so on. When the return value is True, *XmListGetSelectedPos*( ) allocates memory for this array. The caller is responsible for freeing this memory.

position_count
Returns the number of elements in the *position_list*.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns True if the list has any selected items, and False if it does not.

SEE ALSO

XmList.

NAME

XmListItemExists — a List function that checks if a specified item is in the list

SYNOPSIS

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

Boolean XmListItemExists(
      Widget                    widget,
      XmString                  item);
```

DESCRIPTION

XmListItemExists( ) is a Boolean function that checks if a specified item is present in the list.

widget Specifies the ID of the List widget.

item Specifies the item whose presence is checked.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns True if the specified item is present in the list.

SEE ALSO

XmList.

NAME

XmListItemPos — a List function that returns the position of an item in the list

SYNOPSIS

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

int XmListItemPos(
      Widget                     widget,
      XmString                   item);
```

DESCRIPTION

XmListItemPos( ) returns the position of the first instance of the specified item in a list.

widget Specifies the ID of the List widget.

item Specifies the item whose position is returned.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns the position in the list of the first instance of the specified item. The position of the first item in the list is 1; the position of the second item is 2; and so on. This function returns 0 (zero) if the item is not found.

SEE ALSO

XmList.

NAME

XmListPosSelected — a List function that determines if the list item at a specified position is selected.

SYNOPSIS

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

Boolean XmListPosSelected(
      Widget                     widget,
      int                        position);
```

DESCRIPTION

XmPosSelected( ) determines if the list item at the specified position is selected or not.

widget Specifies the ID of the List widget.

position Specifies the position of the list item. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) specifies the last item in the list.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns True if the list item is selected; otherwise, returns False if the item is not selected or the specified position is invalid.

SEE ALSO

XmList.

NAME

XmListPosToBounds — a List function that returns the bounding box of an item at a specified position in a list.

SYNOPSIS

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

Boolean XmListPosToBounds(
      Widget                    widget,
      int                       position,
      Position                 *x,
      Position                 *y,
      Dimension                *width,
      Dimension                *height);
```

DESCRIPTION

XmListPosToBounds( ) returns the coordinates of an item within a list and the dimensions of its bounding box. The function returns the associated x and y coordinates of the upper-left corner of the bounding box relative to the upper-left corner of the List widget, as well as the width and the height of the box. The caller can pass a NULL value for the *x*, *y*, *width* or *height* arguments to indicate that the return value for that argument is not requested.

widget Specifies the ID of the List widget.

position Specifies the position of the specified item. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) specifies the last item in the list.

x Specifies a pointer to the returned x coordinate of the item.

y Specifies the pointer to the returned y coordinate of the item.

width Specifies the pointer to the returned width of the item.

height Specifies the pointer to the returned height of the item.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

If the item at the specified position is not visible, returns False, and the returned values (if any) are undefined. Otherwise, this function returns True.

SEE ALSO

XmList and *XmListYToPos*( ).

NAME

XmListReplaceItems — a List function that replaces the specified elements in the list

SYNOPSIS

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

void XmListReplaceItems(
      Widget                     widget,
      XmString                  *old_items,
      int                        item_count,
      XmString                  *new_items);
```

DESCRIPTION

XmListReplaceItems( ) replaces each specified item of the list with a corresponding new item.

widget Specifies the ID of the List widget.

old_items Specifies the items to be replaced.

item_count Specifies the number of items in *old_items* and *new_items*. This number must be non-negative.

new_items Specifies the replacement items.

Every occurrence of each element of *old_items* is replaced with the corresponding element from *new_items*.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListReplaceItemsPos — a List function that replaces the specified elements in the list

SYNOPSIS

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

void XmListReplaceItemsPos(
      Widget                     widget,
      XmString                  *new_items,
      int                        item_count,
      int                        position);
```

DESCRIPTION

XmListReplaceItemsPos( ) replaces the specified number of items of the List with new items, starting at the specified position in the List.

widget Specifies the ID of the List widget.

new_items Specifies the replacement items.

item_count Specifies the number of items in *new_items* and the number of items in the list to replace. This number must be non-negative.

position Specifies the position of the first item in the list to be replaced. A value of 1 indicates that the first item replaced is the first item in the list; a value of 2 indicates that it is the second item; and so on.

Beginning with the item specified in *position*, *item_count* items in the list are replaced with the corresponding elements from *new_items*.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListReplaceItemsPosUnselected — a List function that replaces items in a list without selecting the replacement items

SYNOPSIS

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

void XmListReplaceItemsPosUnselected(
      Widget                     widget,
      XmString                  *new_items,
      int                        item_count,
      int                        position);
```

DESCRIPTION

XmListReplaceItemsPosUnselected( ) replaces the specified number of items in the list with new items, starting at the given position. The replacement items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget Specifies the ID of the List widget to replace items in.

new_items Specifies a pointer to the replacement items.

item_count Specifies the number of elements in *new_items* and the number of items in the list to replace. This number must be non-negative.

position Specifies the position of the first item in the list to be replaced. A value of 1 indicates that the first item replaced is the first item in the list; a value of 2 indicates that it is the second item; and so on.

Beginning with the item specified in *position*, *item_count* items in the list are replaced with the corresponding elements from *new_items*. That is, the item at *position* is replaced with the first element of *new_items*; the item after *position* is replaced with the second element of *new_items*; and so on, until *item_count* is reached.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListReplaceItemsUnselected — a List function that replaces items in a list

SYNOPSIS

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

void XmListReplaceItemsUnselected(
      Widget                     widget,
      XmString                  *old_items,
      int                        item_count,
      XmString                  *new_items);
```

DESCRIPTION

XmListReplaceItemsUnselected( ) replaces each specified item in the list with a corresponding new item. The replacement items remain unselected, even if they currently appear in the **XmNselectedItems** list.

widget Specifies the ID of the List widget in which items are to be replaced.

old_items Specifies a pointer to the list items to be replaced.

item_count Specifies the number of elements in *old_items* and *new_items*. This number must be non-negative.

new_items Specifies a pointer to the replacement items. Every occurrence of each element of *old_items* is replaced with the corresponding element from *new_items*. That is, the first element of *old_items* is replaced with the first element of *new_items*. The second element of *old_items* is replaced with the second element of *new_items*, and so on until *item_count* is reached. If an element in *old_items* does not exist in the list, the corresponding entry in *new_items* is skipped.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListReplacePositions — a List function that replaces items in a list based on position

SYNOPSIS

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

void XmListReplacePositions(
      Widget                     widget,
      int                       *position_list,
      XmString                  *item_list,
      int                        item_count);
```

DESCRIPTION

XmListReplacePositions( ) replaces non-contiguous items in a list. The item at each position specified in *position_list* is replaced with the corresponding entry in *item_list*. When the items are inserted into the list, they are compared with the current **XmNselectedItems** list. Any of the new items that match items on the selected list appear selected. A warning message is displayed if a specified position is invalid; that is, the value is 0 (zero), a negative integer or a number greater than the number of items in the list.

widget Specifies the ID of the List widget.

position_list Specifies an array of the positions of items to be replaced. The position of the first item in the list is 1; the position of the second item is 2; and so on.

item_list Specifies an array of the replacement items.

item_count Specifies the number of elements in *position_list* and *item_list*. This number must be non-negative.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListSelectItem — a List function that selects an item in the list

SYNOPSIS

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

void XmListSelectItem(
      Widget                    widget,
      XmString                  item,
      Boolean                   notify);
```

DESCRIPTION

XmListSelectItem( ) highlights and adds to the selected list the first item in the list that matches *item*.

widget Specifies the ID of the List widget from whose list an item is selected.

item Specifies the item to be selected in the List widget.

notify Specifies a Boolean value that when True invokes the selection callback for the current mode. From an application interface view, calling this function with *notify* True is indistinguishable from a user-initiated selection action.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList and *XmListSelectPos*( ).

NAME

XmListSelectPos — a List function that selects an item at a specified position in the list

SYNOPSIS

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

void XmListSelectPos(
      Widget                      widget,
      int                         position,
      Boolean                     notify);
```

DESCRIPTION

XmListSelectPos( ) highlights a List item at the specified position and adds it to the list of selected items.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be selected. A value of 1 indicates that the first item in the list is selected; a value of 2 indicates that the second item is selected; and so on. A value of 0 (zero) indicates that the last item in the list is selected.

notify Specifies a Boolean value that when True invokes the selection callback for the current mode. From an application interface view, calling this function with *notify* True is indistinguishable from a user-initiated selection action.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList and *XmListSelectItem*( ).

NAME

XmListSetAddMode — a List function that sets add mode in the list

SYNOPSIS

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

void XmListSetAddMode(
      Widget                     widget,
      Boolean                    state);
```

DESCRIPTION

XmListSetAddMode( ) allows applications control over Add Mode in the extended selection model. This function ensures that the mode it sets is compatible with the selection policy (**XmNselectionPolicy**) of the widget. For example, it cannot put the widget into add mode when the value of **XmNselectionPolicy** is XmBROWSE_SELECT.

widget Specifies the ID of the List widget.

state Specifies whether to activate or deactivate Add Mode. If *state* is True, Add Mode is activated. If *state* is False, Add Mode is deactivated.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListSetBottomItem — a List function that makes an existing item the last visible item in the list

SYNOPSIS

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

void XmListSetBottomItem(
      Widget                     widget,
      XmString                   item);
```

DESCRIPTION

XmListSetBottomItem( ) makes the first item in the list that matches *item* the last visible item in the list.

widget Specifies the ID of the List widget from whose list an item is made the last visible.

item Specifies the item.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListSetBottomPos — a List function that makes a specified item the last visible item in the list

SYNOPSIS

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

void XmListSetBottomPos(
      Widget                    widget,
      int                       position);
```

DESCRIPTION

XmListSetBottomPos( ) makes the item at the specified position the last visible item in the List.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be made the last visible item in the list. A value of 1 indicates that the first item in the list is the last visible item; a value of 2 indicates that the second item is the last visible item; and so on. A value of 0 (zero) indicates that the last item in the list is the last visible item.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListSetHorizPos — a List function that scrolls to the specified position in the list

SYNOPSIS

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

void XmListSetHorizPos(
      Widget                    widget,
      int                       position);
```

DESCRIPTION

XmListSetHorizPos( ) sets the **XmNvalue** resource of the horizontal ScrollBar to the specified position and updates the visible portion of the list with the new value if the List widget's **XmNlistSizePolicy** is set to XmCONSTANT or XmRESIZE_IF_POSSIBLE and the horizontal ScrollBar is currently visible. This is equivalent to moving the horizontal ScrollBar to the specified position.

widget Specifies the ID of the List widget.

position Specifies the horizontal position.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListSetItem — a List function that makes an existing item the first visible item in the list

SYNOPSIS

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

void XmListSetItem(
      Widget                     widget,
      XmString                   item);
```

DESCRIPTION

XmListSetItem( ) makes the first item in the list that matches *item* the first visible item in the list.

widget Specifies the ID of the List widget from whose list an item is made the first visible.

item Specifies the item.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListSetKbdItemPos — a List function that sets the location cursor at a specified position

SYNOPSIS

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

Boolean XmListSetKbdItemPos(
      Widget                    widget,
      int                       position);
```

DESCRIPTION

XmListSetKbdItemPos( ) sets the location cursor at the item specified by *position*. This function does not determine if the item at the specified position is selected or not.

widget Specifies the ID of the List widget.

position Specifies the position of the item at which the location cursor is set. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) sets the location cursor at the last item in the list.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns False if no item exists at the specified position or if the list is empty; otherwise, returns True.

SEE ALSO

XmList.

NAME

XmListSetPos — a List function that makes the item at the given position the first visible position in the list

SYNOPSIS

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

void XmListSetPos(
      Widget                    widget,
      int                       position);
```

DESCRIPTION

XmListSetPos( ) makes the item at the given position the first visible position in the list.

widget Specifies the ID of the List widget.

position Specifies the position of the item to be made the first visible item in the list. A value of 1 indicates that the first item in the list is the first visible item; a value of 2 indicates that the second item is the first visible item; and so on. A value of 0 (zero) indicates that the last item in the list is the first visible item.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListUpdateSelectedList — a List function that updates the **XmNselectedItems** resource

SYNOPSIS

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

void XmListUpdateSelectedList(
      Widget                    widget);
```

DESCRIPTION

XmListUpdateSelectedList( ) frees the contents of the current **XmNselectedItems** list. The routine traverses the **XmNitems** list and adds each currently selected item to the **XmNselectedItems** list. For each selected item, there is a corresponding entry in the updated **XmNselectedItems** list.

widget Specifies the ID of the List widget to update.

For a complete definition of List and its associated resources, see *XmList*.

SEE ALSO

XmList.

NAME

XmListYToPos — a List function that returns the position of the item at a specified y coordinate

SYNOPSIS

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

int XmListYToPos(
      Widget                    widget,
      Position                  y);
```

DESCRIPTION

XmListYToPos( ) returns the position of the item at the given y coordinate within the list.

widget Specifies the ID of the List widget.

y Specifies the y coordinate in the list's coordinate system.

For a complete definition of List and its associated resources, see *XmList*.

RETURN VALUE

Returns the position of the item at the specified y coordinate. A value of 1 indicates the first item in the list; a value of 2 indicates the second item; and so on. A value of 0 (zero) indicates that no item exists at the specified y coordinate.

SEE ALSO

XmList and *XmListPosToBounds*( ).

NAME

XmMainWindow — the MainWindow widget class

SYNOPSIS

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

DESCRIPTION

MainWindow provides a standard layout for the primary window of an application. This layout includes a MenuBar, a CommandWindow, a work region, a MessageWindow and ScrollBars. Any or all of these areas are optional. The work region and ScrollBars in the MainWindow behave identically to the work region and ScrollBars in the ScrolledWindow widget. The user can think of the MainWindow as an extended ScrolledWindow with an optional MenuBar and optional CommandWindow and MessageWindow.

In a fully loaded MainWindow, the MenuBar spans the top of the window horizontally. The CommandWindow spans the MainWindow horizontally just below the MenuBar and the work region lies below the CommandWindow. The MessageWindow is below the work region. Any space remaining below the MessageWindow is managed in a manner identical to ScrolledWindow. The behavior of ScrolledWindow can be controlled by the ScrolledWindow resources. To create a MainWindow, first create the work region elements, a MenuBar, a CommandWindow, a MessageWindow, a horizontal ScrollBar and a vertical ScrollBar widget, and then call *XmMainWindowSetAreas*( ) with those widget IDs.

MainWindow can also create three Separator widgets that provide a visual separation of MainWindow's four components.

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	*XmSeparator*	optional first separator
Separator2	*XmSeparator*	optional second separator
Separator3	*XmSeparator*	optional third separator
VertScrollBar	*XmScrollBar*	vertical scroll bar

Classes

MainWindow inherits behavior and resources from *Core*, *Composite*, *Constraint*, *XmManager* and *ScrolledWindow*.

The class pointer is **xmMainWindowWidgetClass**.

The class name is *XmMainWindow*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmMainWindow **Resource Set**

Name	Class	Type	Default	Access
XmNcommandWindow	**XmCCommandWindow**	**Widget**	NULL	CSG
XmNcommandWindowLocation	**XmCCommandWindowLocation**	**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
Headers/Xm.inchowSeparator	**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**
XmNclipWindow	**XmCClipWindow**	**Widget**	dynamic	G
XmNhorizontalScrollBar	**XmCHorizontalScrollBar**	**Widget**	dynamic	CSG
XmNscrollBarDisplayPolicy	**XmCScrollBarDisplayPolicy**	**unsigned char**	dynamic	CSG
XmNscrollBarPlacement	**XmCScrollBarPlacement**	**unsigned char**	XmBOTTOM _RIGHT	CSG
XmNscrolledWindowMargin Height	**XmCScrolledWindowMargin Height**	**Dimension**	0	N/A
XmNscrolledWindowMargin Width	**XmCScrolledWindowMargin Width**	**Dimension**	0	N/A
XmNscrollingPolicy	**XmCScrollingPolicy**	**unsigned char**	XmAPPLICATION _DEFINED	CG
XmNspacing	**XmCSpacing**	**Dimension**	4	CSG
XmNtraverseObscuredCallback	**XmCCallback**	**XtCallbackList**	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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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

SEE ALSO

Composite, *Constraint*, *Core*, *XmCreateMainWindow*( ), *XmMainWindowSep1*( ), *XmMainWindowSep2*( ), *XmMainWindowSep3*( ), *XmMainWindowSetAreas*( ), *XmManager* and *XmScrolledWindow*

NAME

XmMainWindowSep1 — a MainWindow function that returns the widget ID of the first Separator widget

SYNOPSIS

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

Widget XmMainWindowSep1(
      Widget                    widget);
```

DESCRIPTION

XmMainWindowSep1( ) returns the widget ID of the first Separator widget in the MainWindow. The first Separator widget is located between the MenuBar and the Command widget. This Separator is visible only when **XmNshowSeparator** is True.

widget Specifies the MainWindow widget ID.

For a complete definition of MainWindow and its associated resources, see *XmMainWindow*.

RETURN VALUE

Returns the widget ID of the first Separator.

SEE ALSO

XmMainWindow.

NAME

XmMainWindowSep2 — a MainWindow function that returns the widget ID of the second Separator widget

SYNOPSIS

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

Widget XmMainWindowSep2(
      Widget                    widget);
```

DESCRIPTION

XmMainWindowSep2( ) returns the widget ID of the second Separator widget in the MainWindow. The second Separator widget is located between the Command widget and the ScrolledWindow. This Separator is visible only when **XmNshowSeparator** is True.

widget Specifies the MainWindow widget ID.

For a complete definition of MainWindow and its associated resources, see *XmMainWindow*.

RETURN VALUE

Returns the widget ID of the second Separator.

SEE ALSO

XmMainWindow.

NAME

XmMainWindowSep3 — a MainWindow function that returns the widget ID of the third Separator widget

SYNOPSIS

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

Widget XmMainWindowSep3(
      Widget                    widget);
```

DESCRIPTION

XmMainWindowSep3( ) returns the widget ID of the third Separator widget in the MainWindow. The third Separator widget is located between the message window and the widget above it. This Separator is visible only when **XmNshowSeparator** is True.

widget Specifies the MainWindow widget ID.

For a complete definition of MainWindow and its associated resources, see *XmMainWindow*.

RETURN VALUE

Returns the widget ID of the third Separator.

SEE ALSO

XmMainWindow.

NAME

XmMainWindowSetAreas — a MainWindow function that identifies manageable children for each area

SYNOPSIS

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

void XmMainWindowSetAreas(
      Widget                     widget,
      Widget                     menu_bar,
      Widget                     command_window,
      Widget                     horizontal_scrollbar,
      Widget                     vertical_scrollbar,
      Widget                     work_region);
```

DESCRIPTION

XmMainWindowSetAreas( ) identifies which of the valid children for each area (such as the MenuBar and work region) are to be actively managed by MainWindow. This function also sets up or adds the MenuBar, work window, command window and ScrollBar widgets to the application's main window widget.

Each area is optional; therefore, the user can pass NULL to one or more of the following arguments. The window manager provides the title bar.

widget Specifies the MainWindow widget ID.

menu_bar Specifies the widget ID for the MenuBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNmenuBar**.

command_window
Specifies the widget ID for the command window to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNcommandWindow**.

horizontal_scrollbar
Specifies the ScrollBar widget ID for the horizontal ScrollBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNhorizontalScrollBar**.

vertical_scrollbar
Specifies the ScrollBar widget ID for the vertical ScrollBar to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNverticalScrollBar**.

work_region Specifies the widget ID for the work window to be associated with the MainWindow widget. Set this ID only after creating an instance of the MainWindow widget. The attribute name associated with this argument is **XmNworkWindow**.

For a complete definition of MainWindow and its associated resources, see *XmMainWindow*.

SEE ALSO

XmMainWindow.

NAME

XmManager — the Manager widget class

SYNOPSIS

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

DESCRIPTION

Manager is a widget class used as a supporting superclass for other widget classes. It supports the visual resources, graphics contexts and traversal resources necessary for the graphics and traversal mechanisms.

Classes

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

The class pointer is **xmManagerWidgetClass**.

The class name is *XmManager*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmManager **Resource Set**

Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 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 by means of the keyboard.

Focus is then determined as follows:

- If **XmNinitialFocus** is a traversable non-tab-group widget, that widget receives focus.
- If **XmNinitialFocus** is a traversable tab group, that tab group receives focus. If that tab group is a composite with descendant tab groups or traversable non-tab-group widgets, these procedures are used recursively to assign focus to a descendant of that tab group.
- If **XmNinitialFocus** is NULL, the first traversable non-tab-group widget that is not contained within a nested tab group receives focus.
- If **XmNinitialFocus** is NULL and no traversable non-tab-group widget exists, the first traversable tab group that is not contained within a nested tab group receives focus. If that tab group is a composite with descendant tab groups or traversable non-tab-group widgets, these procedures are used recursively to assign focus to a descendant of that tab group.

If a shell hierarchy regains focus after losing it, focus returns to the widget that had the focus at the time it left the hierarchy.

The use of **XmNinitialFocus** is undefined if the manager is a MenuBar, PulldownMenu, PopupMenu, or OptionMenu.

XmNnavigationType

Determines whether the widget is a tab group.

XmNONE

Indicates that the widget is not a tab group.

XmTAB_GROUP

Indicates that the widget is a tab group, unless the **XmNnavigationType** of another widget in the hierarchy is XmEXCLUSIVE_TAB_GROUP.

XmSTICKY_TAB_GROUP

Indicates that the widget is a tab group, even if the **XmNnavigationType** of another widget in the hierarchy is XmEXCLUSIVE_TAB_GROUP.

XmEXCLUSIVE_TAB_GROUP

Indicates that the widget is a tab group and that widgets in the hierarchy whose **XmNnavigationType** is XmTAB_GROUP are not tab groups.

When a parent widget has an **XmNnavigationType** of XmEXCLUSIVE_TAB_GROUP, traversal of non-tab-group widgets within the group is based on the order of those widgets in their parent's **XmNchildren** list.

XmNshadowThickness

Specifies the thickness of the drawn border shadow. *XmBulletinBoard* and its descendants set this value dynamically. If the widget is a top-level window, this value is set to 1. If it is not a top-level window, this value is set to 0 (zero).

XmNstringDirection

Specifies the initial direction to draw strings. The values for this resource are XmSTRING_DIRECTION_L_TO_R and XmSTRING_DIRECTION_R_TO_L. The value of this resource is determined at creation time. If the widget's parent is a manager, this value is inherited from the widget's parent, otherwise it is set to XmSTRING_DIRECTION_L_TO_R.

XmNtopShadowColor

Specifies the color to use to draw the top and left sides of the border shadow. This color is used if the **XmNtopShadowPixmap** resource is NULL.

XmNtopShadowPixmap

Specifies the pixmap to use to draw the top and left sides of the border shadow.

XmNtraversalOn

Specifies whether traversal is activated for this widget.

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 and bottom shadow 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 visual symbols. If the background is the only color specified for a widget, the top shadow, bottom shadow and foreground colors are generated to give the 3-D appearance.

Colors are generated only at creation. Resetting the background through *XtSetValues*( ) does not regenerate the other colors.

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
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 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.

Action Routines

The *XmManager* action routines are:

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.

ManagerGadgetHelp( )
Calls the callbacks for the current gadget's **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

ManagerGadgetKeyInput( )
Causes the current gadget to process a keyboard event.

ManagerGadgetMultiActivate( )
Causes the current gadget to process a multiple mouse click.

ManagerGadgetMultiArm( )
Causes the current gadget to process a multiple mouse button press.

ManagerGadgetNextTabGroup( )
Traverses to the first item in the next tab group. If the current tab group is the last entry in the tab group list, it wraps to the beginning of the tab group list.

ManagerGadgetPrevTabGroup( )
Traverses to the first item in the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

ManagerGadgetSelect( )
Causes the current gadget to be armed and activated.

ManagerGadgetTraverseDown( )
Traverses to the next item below the current gadget in the current tab group, wrapping if necessary.

ManagerGadgetTraverseHome( )
Traverses to the first widget or gadget in the current tab group.

ManagerGadgetTraverseLeft( )
Traverses to the next item to the left of the current gadget in the current tab group, wrapping if necessary.

ManagerGadgetTraverseNext( )
Traverses to the next item in the current tab group, wrapping if necessary.

ManagerGadgetTraversePrev( )
Traverses to the previous item in the current tab group, wrapping if necessary.

ManagerGadgetTraverseRight( )
Traverses to the next item to the right of the current gadget in the current tab group, wrapping if necessary.

ManagerGadgetTraverseUp( )
Traverses to the next item above the current gadget in the current tab group, wrapping if necessary.

ManagerParentActivate( )
If the parent is a manager, passes the **KActivate** event received by the current widget or gadget to its parent.

ManagerParentCancel( )
If the parent is a manager, passes the **KCancel** event received by the current widget or gadget to its parent.

SEE ALSO

Composite, *Constraint*, *Core*, *XmGadget* and *XmScreen*.

NAME

XmMapSegmentEncoding — a compound string function that returns the compound text encoding format associated with the specified font list tag

SYNOPSIS

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

char *XmMapSegmentEncoding(
      char            *fontlist_tag);
```

DESCRIPTION

XmMapSegmentEncoding( ) searches the segment encoding registry for an entry that matches the specified font list tag and returns a copy of the associated compound text encoding format. The application is responsible for freeing the storage associated with the returned data by calling *XtFree*( ).

fontlist_tag Specifies the compound string font list tag.

RETURN VALUE

Returns a copy of the associated compound text encoding format if the font list tag is found in the registry; otherwise, returns NULL.

SEE ALSO

XmCvtXmStringToCT( ), *XmRegisterSegmentEncoding*( ), Section 4.2 on page 60 and Section 4.3 on page 61.

NAME

XmMenuPosition — a RowColumn function that positions a Popup MenuPane

SYNOPSIS

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

void XmMenuPosition(
      Widget                    menu,
      XButtonPressedEvent      *event);
```

DESCRIPTION

XmMenuPosition( ) positions a Popup MenuPane using the information in the specified event. Unless an application is positioning the MenuPane itself, it must first invoke this function before managing the PopupMenu. The **x_root** and **y_root** fields in the specified event are used to determine the menu position.

menu Specifies the PopupMenu to be positioned.

event Specifies the event passed to the action procedure which manages the PopupMenu.

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

SEE ALSO

XmRowColumn.

NAME

XmMenuShell — the MenuShell widget class

SYNOPSIS

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

DESCRIPTION

The MenuShell widget is a custom OverrideShell widget. An OverrideShell widget bypasses *mwm* when displaying itself. It is designed specifically to contain Popup or Pulldown MenuPanes.

Most application writers never encounter this widget if they use the menu-system convenience functions, *XmCreatePopupMenu*( ) or *XmCreatePulldownMenu*( ) to create a Popup or Pulldown MenuPane. The convenience functions automatically create a MenuShell widget as the parent of the MenuPane. However, if the convenience functions are not used, the application programmer must create the required MenuShell. In this case, it is important to note that the parent of the MenuShell depends on the type of menu system being built.

- If the MenuShell is for the top-level Popup MenuPane, the Shell's parent must be the widget from which the Popup MenuPane is popped up.
- If the MenuShell is for a MenuPane that is pulled down from a Popup or another Pulldown MenuPane, the Shell's parent must be the Popup or Pulldown MenuPane.
- If the MenuShell is for a MenuPane that is pulled down from a MenuBar, the Shell's parent must be the MenuBar.
- If the MenuShell is for a Pulldown MenuPane in an OptionMenu, the Shell's parent must be the OptionMenu's parent.

Setting **XmNheight**, **XmNwidth**, or **XmNborderWidth** for either a MenuShell or its child sets that resource to the same value in both the parent and the child. An application should always specify these resources for the child, not the parent.

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

Classes

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

The class pointer is **xmMenuShellWidgetClass**.

The class name is *XmMenuShell*.

New Resources

MenuShell overrides the **XmNallowShellResize** resource in Shell. The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmMenuShell Resource Set				
Name	**Class**	**Type**	**Default**	**Access**
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	CSG
XmNdefaultFontList	**XmCDefaultFontList**	**XmFontList**	dynamic	CG
XmNlabelFontList	**XmCLabelFontList**	**XmFontList**	dynamic	CSG

XmNbuttonFontList

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

XmNdefaultFontList

Specifies a default font list for Shell'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 Shell's label descendants (Labels and LabelGadgets). If this value is NULL at initialization and if the value of **XmNdefaultFontList** is not NULL, **XmNlabelFontList** is initialized to the value of **XmNdefaultFontList**. If the value of **XmNdefaultFontList** is NULL, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the XmBulletinBoard, VendorShell, or XmMenuShell widget class. If such an ancestor is found, **XmNlabelFontList** is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

Inherited Resources

MenuShell inherits behavior and resources from the superclasses described in the following tables. 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 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).

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 ***	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
XmNinitialResourcesPersistent	**XmCInitialResourcesPersistent**	**Boolean**	True	C
XmNmappedWhenManaged	**XmCMappedWhenManaged**	**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

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.

SEE ALSO

Composite, *Core*, *OverrideShell*, *Shell*, *XmCreateMenuShell*( ), *XmCreatePopupMenu*( ), *XmCreatePulldown*( ) and *XmRowColumn*.

NAME

XmMessageBox — the MessageBox widget class

SYNOPSIS

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

DESCRIPTION

MessageBox is a dialog class used for creating simple message dialogs. Convenience dialogs based on MessageBox are provided for several common interaction tasks, which include giving information, asking questions and reporting errors.

A MessageBox dialog is typically transient in nature, displayed for the duration of a single interaction. MessageBox is a subclass of BulletinBoard and depends on it for much of its general dialog behavior.

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.

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.

At initialization, MessageBox looks for the following bitmap files:

xm_error
xm_information
xm_question
xm_working
xm_warning.

See **XmMainWindow** for a list of the paths that are searched for these files.

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 and resources from *Core*, *Composite*, *Constraint*, *XmManager* and *XmBulletinBoard*.

The class pointer is **xmMessageBoxWidgetClass**.

The class name is *XmMessageBox*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmMessageBox **Resource Set**

Name	Class	Type	Default	Access
XmNcancelCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNcancelLabelString	**XmCCancelLabelString**	**XmString**	dynamic	CSG
XmNdefaultButtonType	**XmCDefaultButtonType**	**unsigned char**	XmDIALOG_OK_BUTTON	CSG
XmNdialogType	**XmCDialogType**	**unsigned char**	XmDIALOG_MESSAGE	CSG
XmNhelpLabelString	**XmCHelpLabelString**	**XmString**	dynamic	CSG
XmNmessageAlignment	**XmCAlignment**	**unsigned char**	XmALIGNMENT_BEGINNING	CSG
XmNmessageString	**XmCMessageString**	**XmString**	""	CSG
XmNminimizeButtons	**XmCMinimizeButtons**	**Boolean**	False	CSG
XmNokCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNokLabelString	**XmCOkLabelString**	**XmString**	dynamic	CSG
XmNsymbolPixmap	**XmCPixmap**	**Pixmap**	dynamic	CSG

XmNcancelCallback

Specifies the list of callbacks called when the user clicks on the cancel button. The reason sent by the callback is XmCR_CANCEL.

XmNcancelLabelString

Specifies the string label for the cancel button. The default for this resource depends on the locale. In the C locale the default is **Cancel**.

XmNdefaultButtonType

Specifies the default PushButton. A value of XmDIALOG_NONE means that there should be no default PushButton. The following types are valid:

XmDIALOG_CANCEL_BUTTON
XmDIALOG_OK_BUTTON
XmDIALOG_HELP_BUTTON
XmDIALOG_NONE.

XmNdialogType

Specifies the type of MessageBox dialog, which determines the default message symbol. The following are the possible values for this resource:

XmDIALOG_ERROR
Indicates an ErrorDialog.

XmDIALOG_INFORMATION
Indicates an InformationDialog.

XmDIALOG_MESSAGE
Indicates a MessageDialog. This is the default MessageBox dialog type. The default message symbol is NULL.

XmDIALOG_QUESTION
Indicates a QuestionDialog.

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*( ).

XmNhelpLabelString
Specifies the string label for the help button. The default for this resource depends on the locale. In the C locale the default is **Help**.

XmNmessageAlignment
Controls the alignment of the message Label. Possible values include the following:

XmALIGNMENT_BEGINNING (default)
XmALIGNMENT_CENTER
XmALIGNMENT_END.

XmNmessageString
Specifies the string to be used as the message.

XmNminimizeButtons
Sets the buttons to the width of the widest button and height of the tallest button if False. If this resource is True, button width and height are set to the preferred size of each button.

XmNokCallback
Specifies the list of callbacks called when the user clicks on the OK button. The reason sent by the callback is XmCR_OK.

XmNokLabelString
Specifies the string label for the OK button. The default for this resource depends on the locale. In the C locale the default is **OK**.

XmNsymbolPixmap
Specifies the pixmap label to be used as the message symbol.

Inherited Resources

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

XmBulletinBoard **Resource Set**

Name	Class	Type	Default	Access
XmNallowOverlap	**XmCAllowOverlap**	**Boolean**	True	CSG
XmNautoUnmanage	**XmCAutoUnmanage**	**Boolean**	False	N/A
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	N/A
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
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
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager **Resource Set**

Name	Class	Type	Default	Access
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 each callback:

```
typedef struct
{
      int                       reason;
      XEvent                   *event;
} XmAnyCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that trigger.ed the callback

SEE ALSO

Composite, *Constraint*, *Core*, *XmBulletinBoard*, *XmCreateErrorDialog*( ), *XmCreateInformationDialog*( ), *XmCreateMessageBox*( ), *XmCreateMessageDialog*( ), *XmCreateQuestionDialog*( ), *XmCreateWarningDialog*( ), *XmCreateWorkingDialog*( ), *XmManager* and *XmMessageBoxGetChild*( ).

NAME

XmMessageBoxGetChild — a MessageBox function that is used to access a component

SYNOPSIS

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

Widget XmMessageBoxGetChild(
      Widget                     widget,
      unsigned char              child);
```

DESCRIPTION

XmMessageBoxGetChild( ) is used to access a component within a MessageBox. The arguments given to the function are the MessageBox widget and a value indicating which component to access.

widget Specifies the MessageBox widget ID.

child Specifies a component within the MessageBox. The following are legal values for this parameter:

XmDIALOG_CANCEL_BUTTON
XmDIALOG_DEFAULT_BUTTON
XmDIALOG_HELP_BUTTON
XmDIALOG_MESSAGE_LABEL
XmDIALOG_OK_BUTTON
XmDIALOG_SEPARATOR
XmDIALOG_SYMBOL_LABEL.

For a complete definition of MessageBox and its associated resources, see *XmMessageBox*.

RETURN VALUE

Returns the widget ID of the specified MessageBox component. An application should not assume that the returned widget is of any particular class.

SEE ALSO

XmMessageBox.

NAME

XmOptionButtonGadget — a RowColumn function that obtains the widget ID for the CascadeButtonGadget in an OptionMenu

SYNOPSIS

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

Widget XmOptionButtonGadget(
      Widget                    option_menu);
```

DESCRIPTION

XmOptionButtonGadget( ) provides the application with the means for obtaining the widget ID for the internally created CascadeButtonGadget. Once the application has obtained the widget ID, it can adjust the visual symbols for the CascadeButtonGadget, if desired.

When an application creates an instance of the OptionMenu widget, the widget creates two internal gadgets. One is a LabelGadget that is used to display RowColumn's **XmNlabelString** resource. The other is a CascadeButtonGadget that displays the current selection and provides the means for posting the OptionMenu's submenu.

option_menu Specifies the OptionMenu widget ID.

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the widget ID for the internal button.

SEE ALSO

XmCreateOptionMenu( ), *XmCascadeButtonGadget*, *XmOptionLabelGadget*( ) and *XmRowColumn*.

NAME

XmOptionLabelGadget — a RowColumn function that obtains the widget ID for the LabelGadget in an OptionMenu

SYNOPSIS

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

Widget XmOptionLabelGadget(
      Widget                    option_menu);
```

DESCRIPTION

XmOptionLabelGadget( ) provides the application with the means for obtaining the widget ID for the internally created LabelGadget. Once the application has obtained the widget ID, it can adjust the visual symbols for the LabelGadget, if desired.

option_menu Specifies the OptionMenu widget ID.

When an application creates an instance of the OptionMenu widget, the widget creates two internal gadgets. One is a LabelGadget that is used to display RowColumn's **XmNlabelString** resource. The other is a CascadeButtonGadget that displays the current selection and provides the means for posting the OptionMenu's submenu.

For a complete definition of RowColumn and its associated resources, see *XmRowColumn*.

RETURN VALUE

Returns the widget ID for the internal label.

SEE ALSO

XmCreateOptionMenu( ), *XmLabelGadget*, *XmOptionButtonGadget*( ) and *XmRowColumn*.

NAME

XmPanedWindow — the PanedWindow widget class

SYNOPSIS

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

DESCRIPTION

PanedWindow is a composite widget that lays out children in a vertically tiled format. Children appear in top-to-bottom fashion, with the first child inserted appearing at the top of the PanedWindow and the last child inserted appearing at the bottom. The PanedWindow grows to match the width of its widest child and all other children are forced to this width. The height of the PanedWindow is equal to the sum of the heights of all its children, the spacing between them, and the size of the top and bottom margins.

The user can also adjust the size of the panes. To facilitate this adjustment, a pane control sash is created for most children. The sash appears as a square box positioned on the bottom of the pane that it controls. The user can adjust the size of a pane by using the mouse or keyboard.

The PanedWindow is also a constraint widget, which means that it creates and manages a set of constraints for each child. You can specify a minimum and maximum size for each pane. The PanedWindow does not allow a pane to be resized below its minimum size or beyond its maximum size. Also, when the minimum size of a pane is equal to its maximum size, no control sash is presented for that pane or for the lowest pane.

The default **XmNinsertPosition** procedure for PanedWindow causes sashes to be inserted at the end of the list of children and causes non-sash widgets to be inserted after other non-sash children but before any sashes.

Descendants

PanedWindow 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
Sash	subclass of *XmPrimitive*	sash
Separator	*XmSeparator*	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 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).

XmPanedWindow **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNmarginHeight	**XmCMarginHeight**	**Dimension**	3	CSG
XmNmarginWidth	**XmCMarginWidth**	**Dimension**	3	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.

XmNrefigureMode
Determines whether the panes' positions are recomputed and repositioned when programmatic changes are being made to the PanedWindow. Setting this resource to True resets the children to their appropriate positions.

XmNsashHeight
Specifies the height of the sash.

XmNsashIndent
Specifies the horizontal placement of the sash along each pane. A positive value causes the sash to be offset from the near (left) side of the PanedWindow, and a negative value causes the sash to be offset from the far (right) side of the PanedWindow. If the offset is greater than the width of the PanedWindow minus the width of the sash, the sash is placed flush against the near side of the PanedWindow.

Whether the placement actually corresponds to the left or right side of the PanedWindow may depend on the value of the **XmNstringDirection** resource.

XmNsashShadowThickness
Specifies the thickness of the shadows of the sashes.

XmNsashWidth
Specifies the width of the sash.

XmNseparatorOn

Determines whether a separator is created between each of the panes. Setting this resource to True creates a Separator at the midpoint between each of the panes.

XmNspacing

Specifies the distance between each child pane.

XmPanedWindow **Constraint Resource Set**				
Name	**Class**	**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 honour 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 above.

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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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

Action Routines

The *XmPanedWindow* action routines are:

Help( )
: Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

NextTabGroup( )
: Moves the keyboard focus to the next tab group. By default, each pane and sash is a tab group.

PrevTabGroup( )
: Moves the keyboard focus to the previous tab group. By default, each pane and sash is a tab group.

SashAction(*action*) or *SashAction*(*Key*, *increment*, *direction*)
: The Start action activates the interactive placement of the pane's borders. The Move action causes the sash to track the position of the pointer. If one of the panes reaches its minimum or maximum size, adjustment continues with the next adjustable pane. The Commit action ends sash motion.

  When sash action is caused by a keyboard event, the sash with the keyboard focus is moved according to the *increment* and *direction* specified. DefaultIncr adjusts the sash by one line. LargeIncr adjusts the sash by one view region. The *direction* is specified as either Up or Down.

  Note that the *SashAction*( ) action routine is not a direct action routine of the *XmPanedWindow*, but rather an action of the Sash control created by the *XmPanedWindow*.

Virtual Bindings

The bindings for virtual keys are vendor specific.

SEE ALSO

Composite, *Constraint*, *Core*, *XmCreatePanedWindow*( ) and *XmManager*.

NAME

XmPrimitive — the Primitive widget class

SYNOPSIS

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

DESCRIPTION

Primitive is a widget class used as a supporting superclass for other widget classes. It handles border drawing and highlighting, traversal activation and deactivation, and various callback lists needed by Primitive widgets.

Classes

Primitive inherits behavior and resources from *Core*.

The class pointer is **xmPrimitiveWidgetClass**.

The class name is *XmPrimitive*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmPrimitive **Resource Set**

Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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.

XmNforeground

Specifies the foreground drawing color used by Primitive widgets.

XmNhelpCallback
Specifies the list of callbacks called when the help key is pressed. The reason sent by the callback is XmCR_HELP.

XmNhighlightColor
Specifies the color of the highlighting rectangle. This color is used if the highlight pixmap resource is XmUNSPECIFIED_PIXMAP.

XmNhighlightOnEnter
Specifies if the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is XmEXPLICIT, this resource is ignored, and the widget is highlighted when it has the focus. If the shell's focus policy is XmPOINTER and if this resource is True, the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is XmPOINTER and if this resource is False, the highlighting rectangle is not drawn when the the cursor moves into the widget. The default is False.

XmNhighlightPixmap
Specifies the pixmap used to draw the highlighting rectangle.

XmNhighlightThickness
Specifies the thickness of the highlighting rectangle.

XmNnavigationType
Determines whether the widget is a tab group.

XmNONE
Indicates that the widget is not a tab group.

XmTAB_GROUP
Indicates that the widget is a tab group, unless the **XmNnavigationType** of another widget in the hierarchy is XmEXCLUSIVE_TAB_GROUP.

XmSTICKY_TAB_GROUP
Indicates that the widget is a tab group, even if the **XmNnavigationType** of another widget in the hierarchy is XmEXCLUSIVE_TAB_GROUP.

XmEXCLUSIVE_TAB_GROUP
Indicates that the widget is a tab group and that widgets in the hierarchy whose **XmNnavigationType** is XmTAB_GROUP are not tab groups.

When a parent widget has an **XmNnavigationType** of XmEXCLUSIVE_TAB_GROUP, traversal of non-tab-group widgets within the group is based on the order of those widgets in their parent's **XmNchildren** list.

XmNshadowThickness
Specifies the size of the drawn border shadow.

XmNtopShadowColor
Specifies the color to use to draw the top and left sides of the border shadow. This color is used if the **XmNtopShadowPixmap** resource is unspecified.

XmNtopShadowPixmap
Specifies the pixmap to use to draw the top and left sides of the border shadow.

XmNtraversalOn
Specifies whether traversal is activated for this widget. In CascadeButton and CascadeButtonGadget, this resource is forced to True unless the parent is an OptionMenu.

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, and bottom shadow 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 visual symbols. If the background is the only color specified for a widget, the top shadow, bottom shadow and foreground colors are generated to give the 3-D appearance.

Colors are generated only at creation. Resetting the background through *XtSetValues*( ) does not regenerate the other colors.

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
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 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.

Action Routines

The *XmPrimitive* action routines are:

PrimitiveHelp( )
Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

PrimitiveNextTabGroup( )
Traverses to the first item in the next tab group. If the current tab group is the last entry in the tab group list, it wraps to the beginning of the tab group list.

PrimitiveParentActivate( )
If the parent is a manager, passes the **KActivate** event received by the widget to the parent.

PrimitiveParentCancel( )
If the parent is a manager, passes the **KCancel** event received by the widget to the parent.

PrimitivePrevTabGroup( )
Traverses to the first item in the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

PrimitiveTraverseDown( )
Traverses to the next item below the current widget in the current tab group, wrapping if necessary.

PrimitiveTraverseHome( )
Traverses to the first widget or gadget in the current tab group.

PrimitiveTraverseLeft( )
Traverses to the next item to the left of the current widget in the current tab group, wrapping if necessary.

PrimitiveTraverseNext( )
Traverses to the next item in the current tab group, wrapping if necessary.

PrimitiveTraversePrev( )
Traverses to the previous item in the current tab group, wrapping if necessary.

PrimitiveTraverseRight( )
Traverses to the next item to the right of the current gadget in the current tab group, wrapping if necessary.

PrimitiveTraverseUp( )
Traverses to the next item above the current gadget in the current tab group, wrapping if necessary.

SEE ALSO

Core.

NAME

XmProcessTraversal — a function that determines which component receives keyboard events when a widget has the focus

SYNOPSIS

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

Boolean XmProcessTraversal(
      Widget                   widget,
      XmTraversalDirection     direction);
```

DESCRIPTION

XmProcessTraversal( ) determines which component of a hierarchy receives keyboard events when the hierarchy that contains the given widget has keyboard focus.

XmProcessTraversal( ) changes focus only when the keyboard focus policy of the widget hierarchy is explicit. If the **XmNkeyboardFocusPolicy** of the nearest shell ancestor of the given widget is not XmEXPLICIT, *XmProcessTraversal*( ) returns False without making any focus changes.

widget Specifies the widget ID of the widget whose hierarchy is to be traversed.

direction Specifies the direction of traversal.

Definitions

To be eligible to receive keyboard focus when the shell's **XmNkeyboardFocusPolicy** is XmEXPLICIT, a widget or gadget must meet the following conditions:

- The widget and its ancestors are not in the process of being destroyed.
- The widget and its ancestors are **sensitive**. A widget is sensitive when its **XmNsensitive** and **XmNancestorSensitive** resources are both True.
- The **XmNtraversalOn** resource for the widget and its ancestors is True.
- The widget is viewable. This means that the widget and its ancestors are managed, realized, and (except for gadgets) mapped. Furthermore, in general, some part of the widget's rectangular area must be unobscured by the widget's ancestors. If an application unmaps a widget that has its **XmNmappedWhenManaged** resource set to True, the result is undefined.

  In a ScrolledWindow with an **XmNscrollingPolicy** of XmAUTOMATIC, a widget that is obscured because it is not within the clip window may be able to receive focus if some part of the widget is within the work area and if an **XmNtraverseObscuredCallback** routine can make the widget at least partially visible by scrolling the window.

Most managers cannot receive focus even if they meet all these conditions. In general only primitives and gadgets are eligible to receive focus. A DrawingArea can receive focus if it meets the conditions above and if, in addition, it has no children whose **XmNsensitive**, **XmNancestorSensitive** and **XmNtraversalOn** resources are all True.

The *direction* argument identifies the kind of traversal action to take. The descriptions of these actions below refer to traversable non-tab-group widgets and traversable tab groups.

- A traversable non-tab-group widget is a widget that is not a tab group and that meets all the conditions for receiving focus described above.
- A traversable tab group widget is a tab group widget that meets the same conditions, except that a manager that is a tab group and meets the other conditions is also eligible for traversal as long as it contains a descendant that can receive focus.

A tab group is a widget whose **XmNnavigationType** is:

- XmTAB_GROUP or XmSTICKY_TAB_GROUP, if the hierarchy (up to the nearest shell ancestor) that contains the widget has no widget whose **XmNnavigationType** is XmEXCLUSIVE_TAB_GROUP
- XmEXCLUSIVE_TAB_GROUP or XmSTICKY_TAB_GROUP, if the hierarchy (up to the nearest shell ancestor) that contains the widget has any widget whose **XmNnavigationType** is XmEXCLUSIVE_TAB_GROUP

Traversal Actions

The hierarchy to be traversed is that containing the *widget* argument. This hierarchy is traversed only up to the nearest shell; *XmProcessTraversal*( ) does not move focus from one shell to another. If the shell containing *widget* does not currently have the focus, any change that *XmProcessTraversal*( ) makes to the element with focus within that shell does not take effect until the next time the shell receives focus.

XmProcessTraversal( ) begins the traversal action from the widget in the hierarchy that currently has keyboard focus or that last had focus when the user traversed away from the shell hierarchy.

The value of the *direction* argument determines which of three kinds of traversal action to take:

- Traversal to a non-tab-group widget. This kind of traversal is possible only when the widget that currently has focus is not a tab group; otherwise, *XmProcessTraversal*( ) returns False for these actions.

  These actions do not move focus from one tab group to another. The actions first determine the containing tab group. This is the tab group containing the widget that currently has focus. The actions traverse only to a non-tab-group widget within the containing tab group.

  A non-tab-group widget is eligible for this kind of traversal if the widget is traversable and has no tab group ancestors up to the containing tab group. If the tab group contains no traversable non-tab-group widgets, *XmProcessTraversal*( ) returns False.

  Following are the possible values of the *direction* argument. Note that when actions wrap, wrapping occurs in the traversal direction. The following describes what happens in a left to right environment:

  XmTRAVERSE_RIGHT

  If the **XmNnavigationType** of the containing tab group is not XmEXCLUSIVE_TAB_GROUP, focus moves to the next traversable non-tab-group widget to the right of the widget that currently has focus. In a left to right environment, at the right side of the tab group this action wraps to the non-tab-group widget at the left side and next toward the bottom. At the rightmost widget in the bottom row of the tab group this action wraps to the non-tab-group widget at the leftmost widget in the upper row.

  In a right to left environment, at the right side of the tab group, this action wraps to the non-tab-group widget at the left side and next toward the top. At the rightmost widget in the upper row of the tab group this action wraps to the non-tab-group widget at the leftmost widget in the bottom row.

  If the **XmNnavigationType** of the containing tab group is XmEXCLUSIVE_TAB_GROUP, focus moves to the next traversable non-tab-group widget in the tab group, proceeding in the order in which the widgets appear in their parents' **XmNchildren** lists. After the last widget in the tab group, this action wraps to the first non-tab-group widget.

XmTRAVERSE_LEFT

If the **XmNnavigationType** of the containing tab group is not XmEXCLUSIVE_TAB_GROUP, focus moves to the next traversable non-tab-group widget to the left of the widget that currently has focus. In a left to right environment, at the left side of the tab group this action wraps to the non-tab-group widget at the right side and next toward the top. At the leftmost widget in the upper row of the tab group this action wraps to the non-tab-group widget at the rightmost widget in the bottom row.

In a right to left environment, at the left side of the tab group this action wraps to the non-tab-group widget at the right side and next toward the bottom. At the leftmost widget in the bottom row of the tab group this action wraps to the non-tab-group widget at the rightmost widget in the upper row.

If the **XmNnavigationType** of the containing tab group is XmEXCLUSIVE_TAB_GROUP, focus moves to the previous traversable non-tab-group widget in the tab group, proceeding in the reverse order in which the widgets appear in their parents' **XmNchildren** lists. After the first widget in the tab group, this action wraps to the last non-tab-group widget.

XmTRAVERSE_DOWN

If the **XmNnavigationType** of the containing tab group is not XmEXCLUSIVE_TAB_GROUP, focus moves to the next traversable non-tab-group widget below the widget that currently has focus. In a left to right environment, at the bottom of the tab group this action wraps to the non-tab-group widget at the top and next toward the right. At the bottom widget in the rightmost column of the tab group this action wraps to the non-tab-group widget at the top widget in the leftmost column.

In a right to left environment, at the bottom of the tab group this action wraps to the non-tab-group widget at the top and next toward the left. At the bottom widget of the leftmost widget of the tab group this action wraps to the non-tab-group widget at the top widget of the rightmost column.

If the **XmNnavigationType** of the containing tab group is XmEXCLUSIVE_TAB_GROUP, focus moves to the next traversable non-tab-group widget in the tab group, proceeding in the order in which the widgets appear in their parents' **XmNchildren** lists. After the last widget in the tab group, this action wraps to the first non-tab-group widget.

XmTRAVERSE_UP

If the **XmNnavigationType** of the containing tab group is not XmEXCLUSIVE_TAB_GROUP, focus moves to the next traversable non-tab-group widget above the widget that currently has focus. In a left to right environment, at the top of the tab group this action wraps to the non-tab-group widget at the bottom and next toward the left. At the top widget of the leftmost column of the tab group this action wraps to the non-tab-group widget at the bottom widget of the rightmost column.

In a right to left environment, at the top of the tab group this action wraps to the non-tab-group widget at the bottom and next toward the right. At the top widget of the right most column of the tab group this action wraps to the non-tab-group widget at the bottom widget of the leftmost column.

If the **XmNnavigationType** of the containing tab group is XmEXCLUSIVE_TAB_GROUP, focus moves to the previous traversable non-tab-group widget in the tab group, proceeding in the reverse order in which the widgets appear in their parents' **XmNchildren** lists. After the first widget in the tab group, this action wraps to the last non-tab-group widget.

XmTRAVERSE_NEXT
Focus moves to the next traversable non-tab-group widget in the tab group, proceeding in the order in which the widgets appear in their parents' **XmNchildren** lists. After the last widget in the tab group, this action wraps to the first non-tab-group widget.

XmTRAVERSE_PREV
Focus moves to the previous traversable non-tab-group widget in the tab group, proceeding in the reverse order in which the widgets appear in their parents' **XmNchildren** lists. After the first widget in the tab group, this action wraps to the last non-tab-group widget.

XmTRAVERSE_HOME
If the **XmNnavigationType** of the containing tab group is not XmEXCLUSIVE_TAB_GROUP, focus moves to the first traversable non-tab-group widget at the initial focus of the tab group.

If the **XmNnavigationType** of the containing tab group is XmEXCLUSIVE_TAB_GROUP, focus moves to the first traversable non-tab-group widget in the tab group, according to the order in which the widgets appear in their parents' **XmNchildren** lists.

- Traversal to a tab group. These actions first determine the current widget hierarchy and the containing tab group. The current widget hierarchy is the widget hierarchy whose root is the nearest shell ancestor of the widget that currently has focus. The containing tab group is is the tab group containing the widget that currently has focus. If the current widget hierarchy contains no traversable tab groups, *XmProcessTraversal*( ) returns False.

  Following are the possible values of the *direction* argument. If any tab group in the current widget hierarchy has an **XmNnavigationType** of 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:

  XmTRAVERSE_NEXT_TAB_GROUP
  Finds the hierarchy that contains *widget*, finds the active tab group (if any), and makes the next tab group the active tab group in the hierarchy.

  XmTRAVERSE_PREV_TAB_GROUP
  Finds the hierarchy that contains *widget*, finds the active tab group (if any), and makes the previous tab group the active tab group in the hierarchy.

- Traversal to any widget. In this case the *widget* argument is the widget to which *XmProcessTraversal*( ) tries to give focus. If the widget is not traversable, *XmProcessTraversal*( ) returns False.

Following are the possible values of the *direction* argument:

XmTRAVERSE_CURRENT
: Finds the hierarchy and the tab group that contain *widget*. If this tab group is not the active tab group, this action makes it the active tab group. If *widget* is an item in the active tab group, this action makes it the active item. If *widget* is the active tab group, this action makes the first traversable item in the tab group the active item.

RETURN VALUE

Returns True if the traversal action succeeded. Returns False if the **XmNkeyboardFocusPolicy** of the nearest shell ancestor of *widget* is not XmEXPLICIT, if the traversal action finds no traversable widget to receive focus, or if the call to the routine has invalid arguments.

SEE ALSO

XmGetVisibility( ) and *XmIsTraversable*( ).

NAME

XmPushButton — the PushButton widget class

SYNOPSIS

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

DESCRIPTION

PushButton issues commands within an application. It consists of a text label or pixmap surrounded by a border shadow. When a PushButton is selected, the shadow changes to give the appearance that it has been pressed in. When a PushButton is unselected, the shadow changes to give the appearance that it is out.

The default behavior associated with a PushButton in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the PushButton. In addition, **BMenu** controls the behavior of the PushButton if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Thickness for a second shadow, used when the PushButton is the default button, may be specified with the **XmNshowAsDefault** resource. If it has a non-zero value, the Label's resources **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop** and **XmNmarginBottom** may be modified to accommodate the second shadow.

If an initial value is specified for **XmNarmPixmap** but not for **XmNlabelPixmap**, the **XmNarmPixmap** value is used for **XmNlabelPixmap**.

Classes

PushButton inherits behavior and resources from *Core*, *XmPrimitive* and *XmLabel*.

The class pointer is **xmPushButtonWidgetClass**.

The class name is *XmPushButton*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmPushButton **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
XmNdefaultButtonShadowThickness	**XmCDefaultButtonShadowThickness**	**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 called when PushButton is activated. PushButton is activated when the user presses and releases the active mouse button while the pointer is inside that widget. Activating the PushButton also disarms it. For this callback, the reason is XmCR_ACTIVATE.

XmNarmCallback

Specifies the list of callbacks called when PushButton is armed. PushButton is armed when the user presses the active mouse button while the pointer is inside that widget. For this callback, the reason is XmCR_ARM.

XmNarmColor

Specifies the color with which to fill the armed button. **XmNfillOnArm** must be set to True for this resource to have an effect. The default for a color display is a color between the background and the bottom shadow color. For a monochrome display, the default is set to the foreground color, and any text in the label appears in the background color when the button is armed.

XmNarmPixmap

Specifies the pixmap to be used as the button face if **XmNlabelType** is XmPIXMAP and PushButton is armed. This resource is disabled when the PushButton is in a menu.

XmNdefaultButtonShadowThickness

This resource specifies the width of the default button indicator shadow. If this resource is 0 (zero), the width of the shadow comes from the value of the **XmNshowAsDefault** resource. If this resource is greater than 0, the **XmNshowAsDefault** resource is only used to specify whether this button is the default. The default value is the initial value of **XmNshowAsDefault**.

XmNdisarmCallback

Specifies the list of callbacks that is called when PushButton is disarmed. PushButton is disarmed when the user presses and releases the active mouse button while the pointer is inside that widget. For this callback, the reason is XmCR_DISARM.

XmNfillOnArm

Forces the PushButton to fill the background of the button with the color specified by **XmNarmColor** when the button is armed and when this resource is set to True. If False, only the top and bottom shadow colors are switched. When the PushButton is in a menu, this resource is ignored and assumed to be False.

XmNmultiClick

If a button click is followed by another button click within the time span specified by the display's multiclick time, and this resource is set to XmMULTICLICK_DISCARD, do not process the second click. If this resource is set to XmMULTICLICK_KEEP, process the event and increment **click_count** in the callback structure. When the button is not in a menu, the default value is XmMULTICLICK_KEEP.

XmNshowAsDefault

If **XmNdefaultButtonShadowThickness** is greater than 0 (zero), a value greater than 0 in this resource specifies to mark this button as the default button. If **XmNdefaultButtonShadowThickness** is 0, a value greater than 0 in this resource specifies to mark this button as the default button with the shadow thickness specified by this resource. The space between the shadow and the default shadow is equal to the sum of both shadows. The default value is 0. When this value is not 0, the Label resources **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop** and **XmNmarginBottom** may be modified to accommodate the second shadow. This resource is disabled when the PushButton is in a menu.

Inherited Resources

PushButton inherits behavior and resources from the superclasses described 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
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	XmFONTLIST_ DEFAULT_TAG	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CSG

XmPrimitive **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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                        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.

Action Routines

The *XmPushButton* action routines are:

Activate( )

This action draws the shadow in the unarmed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is XmPIXMAP, **XmNlabelPixmap** is used for the button face. If the pointer is still within the button, this action calls the callbacks for **XmNactivateCallback**.

Arm( )

This action arms the PushButton. It draws the shadow in the armed state. If the button is not in a menu and if **XmNfillOnArm** is set to True, it fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is XmPIXMAP, the **XmNarmPixmap** is used for the button face. It calls the **XmNarmCallback** callbacks.

ArmAndActivate( )

In a menu, unposts all menus in the menu hierarchy and, unless the button is already armed, calls the **XmNarmCallback** callbacks. This action calls the **XmNactivateCallback** and **XmNdisarmCallback** callbacks.

Outside a menu, draws the shadow in the armed state and, if **XmNfillOnArm** is set to True, fills the button with the color specified by **XmNarmColor**. If **XmNlabelType** is XmPIXMAP, **XmNarmPixmap** is used for the button face. This action calls the **XmNarmCallback** callbacks.

Outside a menu, this action also arranges for the following to happen, either immediately or at a later time: the shadow is drawn in the unarmed state and, if **XmNfillOnArm** is set to True, the background color reverts to the unarmed color. If **XmNlabelType** is XmPIXMAP, **XmNlabelPixmap** is used for the button face. The **XmNactivateCallback** and **XmNdisarmCallback** callbacks are called.

BtnDown( )

This action unposts any menus posted by the PushButton's parent menu, disables keyboard traversal for the menu, and enables mouse traversal for the menu. It draws the shadow in the armed state and, unless the button is already armed, calls the **XmNarmCallback** callbacks.

BtnUp( )

This action unposts all menus in the menu hierarchy and activates the PushButton. It calls the **XmNactivateCallback** callbacks and then the **XmNdisarmCallback** callbacks.

Disarm( )

Calls the callbacks for **XmNdisarmCallback**.

Help( )

In a Pulldown or Popup MenuPane, unposts all menus in the menu hierarchy and, when the shell's keyboard focus policy is 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.

SEE ALSO

Core, *XmCreatePushButton*( ), *XmLabel*, *XmPrimitive* and *XmRowColumn*.

NAME

XmPushButtonGadget — the PushButtonGadget widget class

SYNOPSIS

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

DESCRIPTION

PushButtonGadget issues commands within an application. It consists of a text label or pixmap surrounded by a border shadow. When PushButtonGadget is selected, the shadow changes to give the appearance that the PushButtonGadget has been pressed in. When PushButtonGadget is unselected, the shadow changes to give the appearance that the PushButtonGadget is out.

The default behavior associated with a PushButtonGadget in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the PushButtonGadget. In addition, **BMenu** controls the behavior of the PushButtonGadget if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Thickness for a second shadow may be specified with the **XmNshowAsDefault** resource. If it has a non-zero value, the Label resources **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop** and **XmNmarginBottom** may be modified to accommodate the second shadow.

If an initial value is specified for **XmNarmPixmap** but not for **XmNlabelPixmap**, the **XmNarmPixmap** value is used for **XmNlabelPixmap**.

Classes

PushButtonGadget inherits behavior and resources from *Object*, *RectObj*, *XmGadget* and *XmLabelGadget*.

The class pointer is **xmPushButtonGadgetClass**.

The class name is *XmPushButtonGadget*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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.

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 PushButtonGadget is 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 this button is marked as the default button. If **XmNdefaultButtonShadowThickness** is 0, a value greater than 0 in this resource specifies this button is marked as the default button with the shadow thickness specified by this resource. The space between the shadow and the default shadow is equal to the sum of both shadows. The default value is 0. When this value is not 0, the Label resources **XmNmarginLeft**, **XmNmarginRight**, **XmNmarginTop** and **XmNmarginBottom** may be modified to accommodate the second shadow. This resource is disabled when the PushButton is in a menu.

Inherited Resources

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
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	dynamic	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CSG

XmGadget **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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.

SEE ALSO

Object, *RectObj*, *XmCreatePushButtonGadget*( ), *XmGadget*, *XmLabelGadget* and *XmRowColumn*.

NAME

XmRegisterSegmentEncoding — a compound string function that registers a compound text encoding format for a specified font list element tag

SYNOPSIS

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

char *XmRegisterSegmentEncoding(
      char                        *fontlist_tag,
      char                        *ct_encoding,
```

DESCRIPTION

XmRegisterSegmentEncoding( ) registers a compound text encoding format with the specified font list element tag. The *XmCvtXmStringToCT*( ) function uses this registry to map the font list tags of compound string segments to compound text encoding formats. Registering a font list tag that already exists in the registry overwrites the original entry. You can unregister a font list tag by passing a NULL value for the *ct_encoding* argument.

fontlist_tag Specifies the font list element tag to be registered. The tag must be a NULL-terminated ISO 8859-1 string.

ct_encoding Specifies the compound text character set to be used for segments with the font list tag. The value must be a NULL-terminated ISO 8859-1 string. A value of XmFONTLIST_DEFAULT_TAG maps the specified font list tag to the codeset of the locale.

RETURN VALUE

Returns NULL for a new font list tag or the old *ct_encoding* value for an already registered font list tag. The application is responsible for freeing the storage associated with the returned data (if any) by calling *XtFree*( ).

SEE ALSO

XmCvtXmStringToCT( ), *XmMapSegmentEncoding*( ) Section 4.2 on page 60 and Section 4.3 on page 61.

NAME

XmRemoveProtocolCallback — a VendorShell function that removes a callback from the internal list

SYNOPSIS

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

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

DESCRIPTION

XmRemoveProtocolCallback( ) removes a callback from the internal list.

shell Specifies the widget with which the protocol property is associated.

property Specifies the protocol property.

protocol Specifies the protocol atom (or an **int** cast to **Atom**).

callback Specifies the procedure to call when a protocol message is received.

closure Specifies the client data to be passed to the callback when it is invoked.

For a complete definition of VendorShell and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddProtocolCallback*( ), *XmInternAtom*( ) and *XmRemoveWMProtocolCallback*( ).

NAME

XmRemoveProtocols — a VendorShell function that removes the protocols from the protocol manager and deallocates the internal tables

SYNOPSIS

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

void XmRemoveProtocols(
      Widget                    shell,
      Atom                      property,
      Atom                     *protocols,
      Cardinal                  num_protocols);
```

DESCRIPTION

XmRemoveProtocols( ) removes the protocols from the protocol manager and deallocates the internal tables. If any of the protocols are active, it updates the handlers and updates the property if *shell* is realized.

shell Specifies the widget with which the protocol property is associated.

property Specifies the protocol property.

protocols Specifies the protocol atoms (or **int** cast to **Atom**).

num_protocols Specifies the number of elements in protocols.

For a complete definition of VendorShell and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddProtocols*( ), *XmInternAtom*( ) and *XmRemoveWMProtocols*( ).

NAME

XmRemoveTabGroup — a function that removes a tab group

SYNOPSIS

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

void XmRemoveTabGroup(
      Widget                    tab_group);
```

DESCRIPTION

This function is obsolete and its behavior is replaced by setting **XmNnavigationType** to XmNONE. *XmRemoveTabGroup*( ) removes a widget from the list of tab groups associated with a particular widget hierarchy and sets the widget's **XmNnavigationType** to XmNONE.

tab_group Specifies the widget ID.

SEE ALSO

XmAddTabGroup( ), *XmManager* and *XmPrimitive*.

NAME

XmRemoveWMProtocolCallback — a VendorShell convenience interface that removes a callback from the internal list

SYNOPSIS

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

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

DESCRIPTION

XmRemoveWMProtocolCallback( ) is a convenience interface. It calls *XmRemoveProtocolCallback*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

protocol Specifies the protocol atom (or an **int** type cast to **Atom**).

callback Specifies the procedure to call when a protocol message is received.

closure Specifies the client data to be passed to the callback when it is invoked.

For a complete definition of VendorShell and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddWMProtocolCallbak*( ), *XmInternAtom*( ) and *XmRemoveProtocolCallback*( ).

NAME

XmRemoveWMProtocols — a VendorShell convenience interface that removes the protocols from the protocol manager and deallocates the internal tables

SYNOPSIS

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

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

DESCRIPTION

XmRemoveWMProtocols( ) is a convenience interface. It calls *XmRemoveProtocols*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

protocols Specifies the protocol atoms (or an **int** type cast to **Atom**).

num_protocols
Specifies the number of elements in protocols.

For a complete definition of VendorShell and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmAddWMProtocols*( ), *XmInternAtom*( ) and *XmRemoveProtocols*( ).

NAME

XmRepTypeAddReverse — a representation type manager function that installs the reverse converter for a previously registered representation type

SYNOPSIS

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

void XmRepTypeAddReverse(
      XmRepTypeId               rep_type_id);
```

DESCRIPTION

XmRepTypeAddReverse( ) installs the reverse converter for a previously registered representation type. The reverse converter takes a numerical representation type value and returns its corresponding string value. Certain applications may require this capability to obtain a string value to display on a screen or to build a resource file.

The *values* argument of the *XmRepTypeRegister*( ) function can be used to register representation types with non-consecutive values or with duplicate names for the same value. If the list of numerical values for a representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value. For example, if a *value_names* array has **cancel**, **proceed** and **abort**, and the corresponding *values* array contains 0, 1 and 0, the reverse converter returns **cancel** instead of **abort** for an input value of 0.

rep_type_id Specifies the identification number of the representation type.

SEE ALSO

XmRepTypeGetId( ) and **XmRepTypeRegister**( ).

NAME

XmRepTypeGetId — a representation type manager function that retrieves the identification number of a representation type

SYNOPSIS

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

XmRepTypeId XmRepTypeGetId(
      String                    rep_type);
```

DESCRIPTION

XmRepTypeGetId( ) searches the registration list for the specified representation type and returns the associated identification number.

rep_type Specifies the representation type for which an identification number is requested.

RETURN VALUE

Returns the identification number of the specified representation type. If the representation type is not registered, the function returns [XmREP_TYPE_INVALID].

SEE ALSO

XmRepTypeGetRegistered( ) and *XmRepTypeRegister*( ).

NAME

XmRepTypeGetNameList — a representation type manager function that generates a list of values for a representation type

SYNOPSIS

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

String *XmRepTypeGetNameList(
      XmRepTypeId              rep_type_id,
      Boolean                  use_uppercase_format);
```

DESCRIPTION

XmRepTypeGetNameList( ) generates a NULL-terminated list of the value names associated with the specified representation type. Each value name is a NULL-terminated string. This routine allocates memory for the returned data. The application must free this memory using *XtFree*( ).

rep_type_id Specifies the identification number of the representation type.

use_uppercase_format Specifies a Boolean value that controls the format of the name list. If the value is True, each value name is in upper-case characters prefixed by Xm; if it is False, the names are in lowercase characters.

RETURN VALUE

Returns a pointer to an array of the value names.

SEE ALSO

XmRepTypeGetId( ), *XmRepTypeGetRegistered*( ) and *XmRepTypeRegister*( ).

NAME

XmRepTypeGetRecord — a representation type manager function that returns information about a representation type

SYNOPSIS

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

XmRepTypeEntry XmRepTypeGetRecord(
      XmRepTypeId               rep_type_id);
```

DESCRIPTION

XmRepTypeGetRecord( ) retrieves information about a particular representation type that is registered with the representation type manager. This routine allocates memory for the returned data. The application must free this memory using *XtFree*( ).

rep_type_id The identification number of the representation type.

The representation type entry structure contains the following information:

```
typedef struct
{
   String           rep_type_name,
   String          *value_names,
   unsigned char   *values,
   unsigned char    num_values,
   Boolean          reverse_installed,
   XmRepTypeId      rep_type_id,
} XmRepTypeEntryRec, *XmRepTypeEntry ;
```

rep_type_name The name of the representation type.

value_names An array of representation type value names.

values An array of representation type numerical values.

num_values The number of values associated with the representation type.

reverse_installed A flag that indicates whether or not the reverse converter is installed.

rep_type_id The identification number of the representation type.

RETURN VALUE

Returns a pointer to the representation type entry structure that describes the representation type.

SEE ALSO

XmRepTypeGetId( ), *XmRepTypeGetRegistered*( ) and *XmRepTypeRegister*( ).

NAME

XmRepTypeGetRegistered — a representation type manager function that returns a copy of the registration list

SYNOPSIS

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

XmRepTypeList XmRepTypeGetRegistered()
```

DESCRIPTION

XmRepTypeGetRegistered( ) retrieves information about all representation types that are registered with the representation type manager. The registration list is an array of structures, each of which contains information for a representation type entry. The end of the registration list is marked with a representation type entry whose **rep_type_name** field has a NULL pointer. This routine allocates memory for the returned data. The application must free this memory using *XtFree*( ).

The representation type entry structure contains the following information:

```
typedef struct
{
   String           rep_type_name,
   String          *value_names,
   unsigned char   *values,
   unsigned char    num_values,
   Boolean          reverse_installed,
   XmRepTypeId      rep_type_id,
} XmRepTypeEntryRec, *XmRepTypeList;
```

rep_type_name The name of the representation type.

value_names An array of representation type value names.

values An array of representation type numerical values.

num_values The number of values associated with the representation type.

reverse_installed A flag that indicates whether or not the reverse converter is installed.

rep_type_id The identification number of the representation type.

RETURN VALUE

Returns a pointer to the registration list of representation types.

SEE ALSO

XmRepTypeRegister( ) and *XmRepTypeGetRecord*( ).

NAME

XmRepTypeRegister — a representation type manager function that registers a representation type resource

SYNOPSIS

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

XmRepTypeId XmRepTypeRegister(
     String                     rep_type,
     String                    *value_names,
     unsigned char             *values,
     unsigned char              num_values);
```

DESCRIPTION

XmRepTypeRegister( ) registers a representation type resource with the representation type manager. All features of the representation type management facility become available for the specified representation type. The function installs a forward type converter to convert string values to numerical representation type values.

When the *values* argument is NULL, consecutive numerical values are assumed. The order of the strings in the *value_names* array determines the numerical values for the resource. For example, the first value name is 0 (zero); the second value name is 1; and so on.

If it is non-NULL, the *values* argument can be used to assign values to representation types that have non-consecutive values or have duplicate names for the same value. Representation types registered in this manner consume additional storage and are slightly slower than representation types with consecutive values.

A representation type can only be registered once; if the same representation type name is registered more than once, the behavior is undefined.

The function *XmRepTypeAddReverse*( ) installs a reverse converter for a registered representation type. The reverse converter takes a representation type numerical value and returns the corresponding string value. If the list of numerical values for a representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value. For example, if a *value_names* array has **cancel**, **proceed** and **abort**, and the corresponding *values* array contains 0, 1 and 0, the reverse converter returns **cancel** instead of **abort** for an input value of 0.

rep_type Specifies the representation type name.

value_names Specifies a pointer to an array of value names associated with the representation type. A value name is specified in lowercase characters without an Xm prefix. Words within a name are separated with underscores.

values Specifies a pointer to an array of values associated with the representation type. A value in this array is associated with the value name in the corresponding position of the *value_names* array.

num_values Specifies the number of entries in the *value_names* and *values* arrays.

RETURN VALUE

Returns the identification number for the specified representation type.

SEE ALSO

XmRepTypeAddReverse( ), *XmRepTypeGetId*( ), *XmRepTypeGetNameList*( ), *XmRepTypeGetRecord*( ), *XmRepTypeGetRegistered*( ) and *XmRepTypeValidValue*( ).

NAME

XmRepTypeValidValue — a representation type manager function that tests the validity of a numerical value of a representation type resource

SYNOPSIS

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

Boolean XmRepTypeValidValue(
      XmRepTypeId               rep_type_id,
      unsigned char             test_value,
      Widget                    enable_default_warning);
```

DESCRIPTION

XmRepTypeValidValue( ) tests the validity of a numerical value for a given representation type resource. The function generates a default warning message if the value is invalid and the *enable_default_warning* argument is non-NULL.

rep_type_id Specifies the identification number of the representation type.

test_value Specifies the numerical value to test.

enable_default_warning Specifies the ID of the widget that contains a default warning message. If this parameter is NULL, no default warning message is generated and the application must provide its own error handling.

RETURN VALUE

Returns True if the specified value is valid; otherwise, returns False.

SEE ALSO

XmRepTypeGetId( ) and *XmRepTypeRegister*( ).

NAME

XmResolveAllPartOffsets — a function that allows writing of upward-compatible applications and widgets

SYNOPSIS

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

void XmResolveAllPartOffsets (widget_class, offset, constraint_offset)
      WidgetClass     widget_class;
      XmOffsetPtr     offset;
      XmOffsetPtr     constraint_offset;
```

DESCRIPTION

The use of offset records requires two extra global variables per widget class. The variables consist of pointers to arrays of offsets into the widget record and constraint record for each part of the widget structure. The *XmResolveAllPartOffsets*( ) function allocates the offset records needed by an application to guarantee upward-compatible access to widget instance and constraint records by applications and widgets. These offset records are used by the widget to access all of the widget's variables. A widget needs to take the following steps:

- Instead of creating a resource list, the widget creates an offset resource list. To help you accomplish this, use the **XmPartResource** structure and the **XmPartOffset** macro. The **XmPartResource** data structure looks just like a resource list, but instead of having one integer for its offset, it has two shorts. This is put into the class record as if it were a normal resource list. Instead of using *XtOffset* for the offset, the widget uses *XmPartOffset*.

- If the widget is a subclass of the Constraint class and it defines additional constraint resources, create an offset resource list for the constraint part as well. Instead of using *XtOffset* for the offset, the widget uses *XmConstraintPartOffset* in the constraint resource list.

```
XmPartResource resources[] = {
    {   BarNxyz, BarCXyz, XmRBoolean, sizeof(Boolean),
        XmPartOffset(Bar,xyz), XmRImmediate, (XtPointer)False } };

XmPartResource constraints[] = {
    {   BarNmaxWidth, BarNMaxWidth,
          XmRDimension, sizeof(Dimension),
          XmConstraintPartOffset(Bar,max_width),
          XmRImmediate, (XtPointer)100 } };
```

- Instead of putting the widget size in the class record, the widget puts the widget part size in the same field. If the widget is a subclass of the Constraint class, instead of putting the widget constraint record size in the class record, the widget puts the widget constraint part size in the same field.

- Instead of putting *XtVersion* in the class record, the widget puts *XtVersionDontCheck* in the class record.

- Define a variable, of type **XmOffsetPtr**, to point to the offset record. If the widget is a subclass of the Constraint class, define a variable of type **XmOffsetPtr** to point to the constraint offset record. These can be part of the widget's class record or separate global variables.

- In class initialisation, the widget calls *XmResolveAllPartOffsets*( ), passing it pointers to the class record, the address of the offset record, and the address of the constraint offset record. If the widget not is a subclass of the Constraint class, it should pass NULL as the address of the constraint offset record. This does several things:

— adds the superclass (which, by definition, has already been initialized) size field to the part size field

— if the widget is a subclass of the Constraint class, adds the superclass constraint size field to the constraint size field

— allocates an array based upon the number of superclasses

— if the widget is a subclass of the constraint class, allocates an array for the constraint offset record

— fills in the offsets of all the widget parts and constraint parts with the appropriate values, determined by examining the size fields of all superclass records

— uses the part offset array to modify the offset entries in the resource list to be real offsets, in place.

- The widget defines a constant which will be the index to its part structure in the offsets array. The value should be 1 greater than the index of the widget's superclass. Constants defined for all Xm widgets can be found in **<XmP.h>**.

```
#define BarIndex (XmBulletinBIndex + 1)
```

- Instead of accessing fields directly, the widget must always go through the offset table. The **XmField** and **XmConstraintField** macros help you access these fields. Because the **XmPartOffset**, **XmConstraintPartOffset**, **XmField** and **XmConstraintField** macros concatenate things together, you must ensure that there is no space after the part argument. For example, the following macros do not work because of the space after the part (Label) argument:

```
XmField(w, offset, Label , text, char *)
XmPartOffset(Label , text).
```

Therefore, you must not have any spaces after the part (Label) argument, as illustrated here:

```
XmField(w, offset, Label, text, char *)
```

You can define macros for each field to make this easier.

— Assume an integer field *xyz*:

```
#define BarXyz(w) (*(int *)(((char *) w) + \
    offset[BarIndex] + XtOffset(BarPart,xyz)))
```

— For constraint field *max_width*:

```
#define BarMaxWidth(w) \
    XmConstraintField(w,constraint_offsets,Bar,max_width,Dimension)
```

The parameters for XmResolveAllPartOffsets are defined below:

widget_class Specifies the widget class pointer for the created widget

offset Returns the offset record

constraint_offset
Returns the constraint offset record

RELATED INFORMATION

XmResolvePartOffsets( )

NAME

XmResolvePartOffsets — a function that allows writing of upward-compatible applications and widgets

SYNOPSIS

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

void XmResolvePartOffsets(
      WidgetClass              widget_class,
      XmOffsetPtr             *offset);
```

DESCRIPTION

The use of offset records requires one extra global variable per widget class. The variable consists of a pointer to an array of offsets into the widget record for each part of the widget structure. The *XmResolvePartOffsets*( ) function allocates the offset records needed by an application to guarantee upward-compatible access to widget instance records by applications and widgets. These offset records are used by the widget to access all of the widget's variables. A widget needs to take the steps described in the following paragraphs.

Instead of creating a resource list, the widget creates an offset resource list. To accomplish this, use the **XmPartResource** structure and the *XmPartOffset*( ) macro. The **XmPartResource** data structure looks just like a resource list, but instead of having one integer for its offset, it has two short integers. This structure is put into the class record as if it were a normal resource list. Instead of using *XtOffset*( ) for the offset, the widget uses *XmPartOffset*( ):

```
XmPartResource resources[] = {
  { BarNxyz, BarCXyz, XmRBoolean,
    sizeof(Boolean), XmPartOffset(Bar,xyz),
    XmRImmediate, (XtPointer)False }
};
```

Instead of putting the widget size in the class record, the widget puts the widget part size in the same field.

Instead of putting XtVersion in the class record, the widget puts XtVersionDontCheck in the class record.

The widget defines a variable, of type **XmOffsetPtr**, to point to the offset record. This can be part of the widget's class record or a separate global variable.

In class initialization, the widget calls *XmResolvePartOffsets*( ), passing it a pointer to contain the address of the offset record and the class record. This does several things:

- Adds the superclass (which, by definition, has already been initialized) size field to the part size field.
- Allocates an array based upon the number of superclasses.
- Fills in the offsets of all the widget parts with the appropriate values, determined by examining the size fields of all superclass records.
- Uses the part offset array to modify the offset entries in the resource list to be real offsets.

The widget defines a constant that is the index to its part structure in the offsets array. The value should be 1 greater than the index of the widget's superclass. Constants defined for all **Xm** widgets can be found in **<Xm.h>** or the files it recursively includes.

```
#define BarIndex(XmBulletinBIndex + 1)
```

Instead of accessing fields directly, the widget must always go through the offset table. The XmField macro helps you access these fields. Because the XmPartOffset and XmField macros concatenate items, you must ensure that there is no space after the *part* argument. For example, the following macros do not work because of the space after the *part* (Label) argument:

```
XmField(w, offset, Label , text, char *)
XmPartOffset(Label , text)
```

The following example is correct:

```
XmField(w, offset, Label, text, char *)
```

You can define macros for each field to make this easier. Assume an integer field **xyz**:

```
#define BarXyz(w) (*(int *)(((char *) w) + \
   offset[BarIndex] + XtOffset(BarPart,xyz)))
```

The arguments for *XmResolvePartOffsets*( ) are:

widget_class Specifies the widget class pointer for the created widget.

offset Returns the offset record.

NAME

XmRowColumn — the RowColumn widget class

SYNOPSIS

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

DESCRIPTION

The RowColumn widget is a general purpose RowColumn manager capable of containing any widget type as a child. In general, it requires no special knowledge about how its children function and provides nothing beyond support for several different layout styles. However, it can be configured as a menu, in which case it expects only certain children, and it configures to a particular layout. The menus supported are MenuBar, Pulldown or Popup MenuPanes and OptionMenu.

The type of layout performed is controlled by how the application has set the various layout resources. It can be configured to lay out its children in either rows or columns. In addition, the application can specify how the children are laid out, as follows:

- The children are packed tightly together into either rows or columns.
- Each child is placed in an identically sized box (producing a symmetrical look).
- A 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.

In a MenuBar, Pulldown MenuPan or Popup MenuPane the default for the **XmNshadowThickness** resource is 2. In an OptionMenu or a WorkArea, (such as a RadioBox or CheckBox) this resource is not applicable and its use is undefined. If an application wishes to place a 3-D shadow around an OptionMenu or WorkArea, it can create the RowColumn as a child of a Frame widget.

In a MenuBar, Pulldown MenuPane or Popup MenuPane the **XmNnavigationType** resource is not applicable and its use is undefined. In a WorkArea, the default for **XmNnavigationType** is XmTAB_GROUP. In an OptionMenu the default for **XmNnavigationType** is XmNONE.

In a MenuBar, Pulldown MenuPane or Popup MenuPane the **XmNtraversalOn** resource is not applicable and its use is undefined. In an OptionMenu or WorkArea, the default for **XmNtraversalOn** is True.

If the parent of the RowColumn is a MenuShell, the **XmNmappedWhenManaged** resource is forced to False when the widget is realized.

Tear-off Menus

Pulldown and Popup MenuPanes support tear-off menus, which enable the user to retain a MenuPane on the display to facilitate subsequent menu selections. A MenuPane that can be torn-off is identified by a tear-off button that spans the width of the MenuPane and displays a dashed line. A torn-off MenuPane contains a window manager system menu icon and a title bar. The window title displays the label of the cascade button that initiated the action when the label type is XmSTRING. If the label contains a pixmap the window title is empty. A tear-off menu from a Popup MenuPane also displays an empty title.

The **XmNtearOffMenuDeactivateCallback**, **XmNtearOffMenuActivateCallback** and **XmNtearOffModel** are RowColumn resources that affect tear-off menu behavior.

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 and resources from *Core*, *Composite*, *Constraint* and *XmManager* classes.

The class pointer is **xmRowColumnWidgetClass**.

The class name is *XmRowColumn*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
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
XmNmnemonic	**XmCMnemonic**	**KeySym**	NULL	CSG
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	XmFONTLIST_DEFAULT_TAG	CSG
XmNnumColumns	**XmCNumColumns**	**short**	1	CSG
XmNorientation	**XmCOrientation**	**unsigned char**	dynamic	CSG
XmNpacking	**XmCPacking**	**unsigned char**	dynamic	CSG
XmNpopupEnabled	**XmCPopupEnabled**	**Boolean**	True	CSG
XmNradioAlwaysOne	**XmCRadioAlwaysOne**	**Boolean**	True	CSG
XmNradioBehavior	**XmCRadioBehavior**	**Boolean**	False	CSG
XmNresizeHeight	**XmCResizeHeight**	**Boolean**	True	CSG
XmNresizeWidth	**XmCResizeWidth**	**Boolean**	True	CSG
XmNrowColumnType	**XmCRowColumnType**	**unsigned char**	XmWORK_AREA	CG
XmNspacing	**XmCSpacing**	**Dimension**	dynamic	CSG
XmNsubMenuId	**XmCMenuWidget**	**Widget**	NULL	CSG
XmNtearOffMenuActivateCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNtearOffMenuDeactivateCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNtearOffModel	**XmCTearOffModel**	**unsigned char**	XmTEAR_OFF_DISABLED	CSG
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

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* 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**.

XmNisAligned

Specifies text alignment for each item within the RowColumn widget; this applies only to items that are subclasses of *XmLabel* or *XmLabelGadget*. However, if the item is a Label widget or gadget and its parent is either a Popup MenuPane or a Pulldown MenuPane, alignment is not performed; the Label is treated as the title within the MenuPane, and the alignment set by the application is not overridden. **XmNentryAlignment** controls the type of textual alignment.

XmNisHomogeneous

Indicates whether the RowColumn widget should enforce exact homogeneity among the items it contains; if this resource is set to True, only the widgets that are of the class indicated by **XmNentryClass** are allowed as children of the RowColumn widget. This is most often used when creating a MenuBar. Attempting to insert a child that is not a member of the specified class generates a warning message.

In a MenuBar, the value of **XmNisHomogeneous** is forced to True. In an OptionMenu, the value is forced to False. When *XmCreateRadioBox*( ) is called the default value is True. Otherwise, the default value is False.

XmNlabelString

Points to a text string that displays the label to one side of the selection area when **XmNrowColumnType** is set to XmMENU_OPTION. This resource is not meaningful for all other RowColumn types. If the application wishes to change the label after creation, it must get the LabelGadget ID (*XmOptionLabelGadget*( )) and call *XtSetValues*( ) on the LabelGadget directly. The default value is no label.

XmNmapCallback

Specifies a widget-specific callback function that is invoked when the window associated with the RowColumn widget is about to be mapped. The callback reason is XmCR_MAP.

XmNmarginHeight

Specifies the amount of blank space between the top edge of the RowColumn widget and the first item in each column, and the bottom edge of the RowColumn widget and the last item in each column. The default value is 0 (zero) for Pulldown and Popup MenuPanes, and 3 pixels for other RowColumn types.

XmNmarginWidth

Specifies the amount of blank space between the left edge of the RowColumn widget and the first item in each row, and the right edge of the RowColumn widget and the last item in each row. The default value is 0 (zero) for Pulldown and Popup MenuPanes, and 3 pixels for other RowColumn types.

XmNmenuAccelerator

This resource is useful only when the RowColumn widget has been configured to operate as a Popup MenuPane or a MenuBar. The format of this resource is similar to the left side specification of a translation string, with the limitation that it must specify a key event. For a Popup MenuPane, when the accelerator is typed by the user, the Popup MenuPane is posted. For a MenuBar, when the accelerator is typed by the user, the first item in the MenuBar is highlighted and traversal is enabled in the MenuBar. The default for a Popup MenuPane is **KMenu**. The default for a MenuBar is **KMenuBar**. Setting the **XmNpopupEnabled** resource to False disables the accelerator.

XmNmenuHelpWidget

Specifies the widget ID for the CascadeButton, which is treated as the Help widget if **XmNrowColumnType** is set to XmMENU_BAR. The MenuBar always places the Help widget at the bottom right corner (in a left to right environment) of the MenuBar. If the RowColumn widget is any type other than XmMENU_BAR, this resource is not meaningful.

XmNmenuHistory

Specifies the widget ID of the last menu entry to be activated. It is also useful for specifying the current selection for an OptionMenu. If **XmNrowColumnType** is set to XmMENU_OPTION, the specified menu item is positioned under the cursor when the menu is displayed.

If the RowColumn widget has the **XmNradioBehavior** resource set to True, the widget field associated with this resource contains the widget ID of the last ToggleButton or ToggleButtonGadget to change from unselected to selected. The default value is the widget ID of the first child in the widget.

XmNmnemonic

This resource is useful only when **XmNrowColumnType** is set to XmMENU_OPTION. It specifies a keysym for a key that, when pressed by the user along with the **MAlt** modifier, posts the associated Pulldown MenuPane. The first character in the OptionMenu label string that exactly matches the mnemonic in the character set specified in **XmNmnemonicCharSet** is underlined. The user can post the menu by pressing either the shifted or the unshifted mnemonic key. The default is no mnemonic.

XmNmnemonicCharSet

Specifies the character set of the mnemonic for an OptionMenu. The default is XmFONTLIST_DEFAULT_TAG. If the RowColumn widget is any type other than XmMENU_OPTION, this resource is not meaningful.

XmNnumColumns

Specifies the number of minor dimension extensions that are made to accommodate the entries; this attribute is meaningful only when **XmNpacking** is set to XmPACK_COLUMN.

For vertically oriented RowColumn widgets, this attribute indicates how many columns are built; the number of entries per column is adjusted to maintain this number of columns, if possible.

For horizontally oriented RowColumn widgets, this attribute indicates how many rows are built.

The default value is 1. In an OptionMenu the value is forced to 1. The value must be greater than 0 (zero).

XmNorientation

Determines whether RowColumn layouts are row-major or column-major. In a column-major layout, the children of the RowColumn are laid out in columns top to bottom within the widget. In a row-major layout the children of the RowColumn are laid out in rows. The XmVERTICAL resource value selects a column-major layout. XmHORIZONTAL selects a row-major layout.

When creating a MenuBar or an OptionMenu, the default is XmHORIZONTAL. Otherwise, the default value is XmVERTICAL. The results of specifying a value of XmVERTICAL for a MenuBar are undefined.

XmNpacking

Specifies how to pack the items contained within a RowColumn widget. This can be set to XmPACK_TIGHT, XmPACK_COLUMN or XmPACK_NONE. When a RowColumn widget packs the items it contains, it determines its major dimension using the value of the **XmNorientation** resource.

XmPACK_TIGHT indicates that given the current major dimension (for example, vertical if **XmNorientation** is XmVERTICAL), entries are placed one after the other until the RowColumn widget must wrap. RowColumn wraps when there is no room left for a complete child in that dimension. Wrapping occurs by beginning a new row or column in the next available space. Wrapping continues, as often as necessary, until all of the children are laid out. In the vertical dimension (columns), boxes are set to the same width; in the horizontal dimension (rows), boxes are set to the same depth. Each entry's position in the major dimension is left unaltered (for example, **XmNy** is left unchanged when **XmNorientation** is XmVERTICAL); its position in the minor dimension is set to the same value as the greatest entry in that particular row or column. The position in the minor dimension of any particular row or column is independent of all other rows or columns.

XmPACK_COLUMN indicates that all entries are placed in identically sized boxes. The boxes are based on the largest height and width values of all the children widgets. The value of the **XmNnumColumns** resource determines how many boxes are placed in the major dimension, before extending in the minor dimension.

XmPACK_NONE indicates that no packing is performed. The *x* and *y* attributes of each entry are left alone, and the RowColumn widget attempts to become large enough to enclose all entries.

When *XmCreateRadioBox*( ) is called or when **XmNrowColumnType** is set to XmWORK_AREA and **XmNradioBehavior** is True, the default value of **XmNpacking** is XmPACK_COLUMN. In an OptionMenu the value is initialized to XmPACK_TIGHT. Otherwise, the value defaults to XmPACK_TIGHT.

XmNpopupEnabled

Allows the menu system to enable keyboard input (accelerators and mnemonics) defined for the Popup MenuPane and any of its submenus. The Popup MenuPane needs to be informed whenever its accessibility to the user changes because posting of the Popup MenuPane is controlled by the application. The default value of this resource is True (keyboard input—accelerators and mnemonics—defined for the Popup MenuPane and any of its submenus is enabled).

XmNradioAlwaysOne

If True, forces the active ToggleButton or ToggleButtonGadget to be automatically selected after having been unselected (if no other toggle was activated). If False, the active toggle may be unselected. The default value is True. This resource is important only when **XmNradioBehavior** is True.

The application can always add and subtract toggles from RowColumn regardless of the selected or 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 non-standard 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 possible settings for this resource are:

XmWORK_AREA (default)
XmMENU_BAR
XmMENU_PULLDOWN
XmMENU_POPUP
XmMENU_OPTION.

This resource cannot be changed after the RowColumn widget is created. Any changes attempted through *XtSetValues*( ) are ignored.

The value of this resource is used to determine the value of a number of other resources. The descriptions of RowColumn resources explain this when it is the case. The resource **XmNnavigationType**, inherited from *XmManager*, is changed to XmNONE if **XmNrowColumnType** is XmMENU_OPTION.

XmNspacing

Specifies the horizontal and vertical spacing between items contained within the RowColumn widget. The default value is 3 pixels for XmOPTION_MENU and XmWORK_AREA and 0 (zero) for other RowColumn types.

XmNsubMenuId

Specifies the widget ID for the Pulldown MenuPane to be associated with an OptionMenu. This resource is useful only when **XmNrowColumnType** is set to XmMENU_OPTION. The default value is NULL.

XmNtearOffMenuActivateCallback

Specifies the callback list that notifies the application when the tear-off MenuPane is about to be activated. It precedes the tear-off menu's map callback.

This resource enables the application to track whether a menu item is sensitive or insensitive and to set the state to the original parent's menu item state when the torn-off menu is reposted. The application can use *XmGetPostedFromWidget*( ) to determine from which parent the menu was torn. The callback reason is XmCR_TEAR_OFF_ACTIVATE. The default is NULL.

XmNtearOffMenuDeactivateCallback

Specifies the callback list that notifies the application when the tear-off MenuPane is about to be deactivated. It follows the tear-off menu's unmap callback.

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 capability 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.

XmNunmapCallback

Specifies a list of callbacks called after the window associated with the RowColumn widget has been unmapped. The callback reason is XmCR_UNMAP. The default value is NULL.

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.

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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 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.

Action Routines

The *XmRowColumn* action routines are:

Help( )

Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

ManagerGadgetSelect( )

When a gadget child of the menu has the focus, invokes the gadget child's behavior associated with **KSelect**. This generally has the effect of unposting the menu hierarchy and arming and activating the gadget, except that, for a CascadeButtonGadget with a submenu, it posts the submenu.

MenuBtnDown( )

When a gadget child of the menu has focus, invokes the gadget child's behavior associated with **BSelect Press**. This generally has the effect of unposting any menus posted by the parent menu, enabling mouse traversal in the menu, and arming the gadget. For a CascadeButtonGadget with a submenu, it also posts the associated submenu.

MenuBtnUp( )

When a gadget child of the menu has focus, invokes the gadget child's behavior associated with **BSelect Release**. This generally has the effect of unposting the menu hierarchy and activating the gadget, except that for a CascadeButtonGadget with a submenu, it posts the submenu and enables keyboard traversal in the menu.

MenuGadgetEscape( )

In a top-level Pulldown MenuPane from a MenuBar, unposts the menu, disarms the MenuBar CascadeButton and the MenuBar, and, when the shell's keyboard focus policy is XmEXPLICIT, restores keyboard focus to the widget that had the focus before the MenuBar was entered. In other Pulldown MenuPanes, unposts the menu.

In a Popup MenuPane, unposts the menu and, when the shell's keyboard focus policy is XmEXPLICIT, restores keyboard focus to the widget from which the menu was posted.

MenuGadgetTraverseDown( )

If the current menu item has a submenu and is in a MenuBar, 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, this action disarms the current menu item and arms the item below it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's bottom edge, this action wraps to the topmost menu item in the column to the right, if one exists. When the current menu item is at the bottom, rightmost corner of the MenuPane, this action wraps to the tear-off control, if present, or to the top, leftmost menu item.

MenuGadgetTraverseLeft( )

When the current menu item is in a MenuBar, this action disarms the current item and arms the MenuBar item to the left. This action wraps within the MenuBar.

In MenuPanes, if the current menu item is not at the left edge of a MenuPane, this action disarms the current item and arms the item to its left. If the current menu item is at the left edge of a submenu attached to a MenuBar item, 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, this action unposts the current submenu only.

In Popup or Torn-off MenuPanes, when the current menu item is at the left edge, this action wraps within the MenuPane. If the current menu item is at the left edge of the MenuPane and not in the top row, this action wraps to the rightmost menu item in the row above. If the current menu item is in the upper, leftmost corner, this action wraps to the tear-off control, if present, or else it wraps to the bottom, rightmost menu item in the MenuPane.

MenuGadgetTraverseRight( )

If the current menu item is in a MenuBar, 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, 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, this action unposts all submenus and traverses to the MenuBar item to the right. If that MenuBar item has a submenu, it posts the submenu and arms the submenu's first traversable item.

In Popup or Torn-off menus, if the current menu item is not a CascadeButton and is at the right edge of a row (except the bottom row), this action wraps to the leftmost menu item in the row below. If the current menu item is not a CascadeButton and is in the bottom, rightmost corner of a Popup or Pulldown MenuPane, this action wraps to the tear-off control, if present, or else it wraps to the top, leftmost menu item of the MenuPane.

MenuGadgetTraverseUp( )

When the current menu item is in a MenuPane, this action disarms the current menu item and arms the item above it. This action wraps within the MenuPane. When the current menu item is at the MenuPane's top edge, this action wraps to the bottommost menu item in the column to the left, if one exists. When the current menu item is at the top, leftmost corner of the MenuPane, this action wraps to the tear-off control, if present, or to the bottom, rightmost menu item.

SEE ALSO

Composite, *Constraint*, *Core*, *XmCreateMenuBar*( ), *XmCreateOptionMenu*( ), *XmCreatePopupMenu*( ), *XmCreatePulldownMenu*( ), *XmCreateRadioBox*( ), *XmCreateRowColumn*( ), *XmCreateWorkArea*( ), *XmGetMenuCursor*( ), *Functions/XmGetP.inctedFromWidget*( ), *XmLabel*, *XmManager*, *XmMenuPosition*( ), *XmOptionButtonGadget*( ), *XmOptionLabelGadget*( ) and *XmUpdateDisplay*( ).

NAME

XmScale — the Scale widget class

SYNOPSIS

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

DESCRIPTION

Scale is used by an application to indicate a value from within a range of values, and it allows the user to input or modify a value from the same range.

A Scale has an elongated rectangular region similar to a ScrollBar. A slider inside this region indicates the current value along the Scale. The user can also modify the Scale's value by moving the slider within the rectangular region of the Scale. A Scale can also include a label set located outside the Scale region. These can indicate the relative value at various positions along the scale.

A Scale can be either input/output or output only. An input/output Scale's value can be set by the application and also modified by the user with the slider. An output-only Scale is used strictly as an indicator of the current value of something and cannot be modified interactively by the user. The *Core* resource **XmNsensitive** specifies whether the user can interactively modify the Scale's value.

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 and resources from *Core*, *Composite*, *Constraint* and *XmManager* classes.

The class pointer is **xmScaleWidgetClass**.

The class name is *XmScale*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmScale **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNdecimalPoints	**XmCDecimalPoints**	**short**	0	CSG
XmNdragCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNfontList	**XmCFontList**	**XmFontList**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	2	CSG
XmNmaximum	**XmCMaximum**	**int**	100	CSG
XmNminimum	**XmCMinimum**	**int**	0	CSG
XmNorientation	**XmCOrientation**	**unsigned char**	XmVERTICAL	CSG
XmNprocessingDirection	**XmCProcessingDirection**	**unsigned char**	dynamic	CSG
XmNscaleHeight	**XmCScaleHeight**	**Dimension**	0	CSG
XmNscaleMultiple	**XmCScaleMultiple**	**int**	dynamic	CSG
XmNscaleWidth	**XmCScaleWidth**	**Dimension**	0	CSG
XmNshowValue	**XmCShowValue**	**XtEnum**	XmNONE	CSG
XmNtitleString	**XmCTitleString**	**XmString**	NULL	CSG
XmNvalue	**XmCValue**	**int**	dynamic	CSG
XmNvalueChangedCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmNdecimalPoints

Specifies the number of decimal points to shift the slider value when displaying it. For example, a slider value of 2,350 and an **XmdecimalPoints** value of 2 results in a display value of 23.50. The value must not be negative.

XmNdragCallback

Specifies the list of callbacks called when the slider position changes as the slider is being dragged. The reason sent by the callback is XmCR_DRAG.

XmNfontList

Specifies the font list to use for the title text string specified by **XmNtitleString**, and the label displayed when **XmNshowValue** is True. If this value is NULL at initialization, the parent hierarchy is searched for an ancestor that is a subclass of the BulletinBoard, VendorShell, or MenuShell widget class. If such an ancestor is found, the font list is initialized to the **XmNlabelFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to **XmFontList** for more information on the creation and structure of a font list.

XmNhighlightOnEnter

Specifies whether the highlighting rectangle is drawn when the cursor moves into the widget. If the shell's focus policy is XmEXPLICIT, this resource is ignored, and the widget is highlighted when it has the focus. If the shell's focus policy is XmPOINTER and if this resource is True, the highlighting rectangle is drawn when the the cursor moves into the widget. If the shell's focus policy is XmPOINTER and if this resource is False, the highlighting rectangle is not drawn when the the cursor moves into the widget. The default is False.

XmNhighlightThickness

Specifies the size of the slider's border drawing rectangle used for enter window and traversal highlight drawing.

XmNmaximum

Specifies the slider's maximum value. **XmNmaximum** must be greater than **XmNminimum**.

XmNminimum

Specifies the slider's minimum value. **XmNmaximum** must be greater than **XmNminimum**.

XmNorientation
Displays Scale vertically or horizontally. This resource can have values of XmVERTICAL and XmHORIZONTAL.

XmNprocessingDirection
Specifies whether the value for **XmNmaximum** is on the right or left side of **XmNminimum** for horizontal Scales or above or below **XmNminimum** for vertical Scales. This resource can have values of XmMAX_ON_TOP, XmMAX_ON_BOTTOM, XmMAX_ON_LEFT and XmMAX_ON_RIGHT. If the Scale is oriented vertically, the default value is XmMAX_ON_TOP. If the XmScale is oriented horizontally, the default value may depend on the value of the **XmNstringDirection** resource.

XmNscaleHeight
Specifies the height of the slider area. The value should be in the specified unit type (the default is pixels). If no value is specified a default height is computed.

XmNscaleMultiple
Specifies the amount to move the slider when the user takes an action that moves the slider by a multiple increment. The default is (**XmNmaximum** − **XmNminimum**) divided by 10, with a minimum of 1.

XmNscaleWidth
Specifies the width of the slider area. The value should be in the specified unit type (the default is pixels). If no value is specified a default width is computed.

XmNshowValue
Specifies whether a label for the current slider value should be displayed next to the slider. If the value is XmNEAR_SLIDER, the current slider value is displayed. If the value is XmNONE, no slider value is displayed.

XmNtitleString
Specifies the title text string to appear in the Scale widget window.

XmNvalue
Specifies the slider's current position along the scale, between **XmNminimum** and **XmNmaximum**. The value is constrained to be within these inclusive bounds. The initial value of this resource is the larger of 0 (zero) and **XmNminimum**.

XmNvalueChangedCallback
Specifies the list of callbacks 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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 each callback:

```
typedef struct
{
     int                    reason;
     XEvent                *event;
     int                    value;
} XmScaleCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

value Is the new slider value.

Behavior

XmScale has the following behavior:

BSelect Press or **BTransfer Press**

In the region between an end of the Scale and the slider:
Moves the slider by one multiple increment in the direction of the end of the Scale and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_RIGHT or XmMAX_ON_BOTTOM, movement toward the right or bottom increments the Scale value, and movement toward the left or top decrements the Scale value. If **XmNprocessingDirection** is XmMAX_ON_LEFT or XmMAX_ON_TOP, movement toward the right or bottom decrements the Scale value, and movement toward the left or top increments the Scale value. If the button is held down longer than a delay period, the slider is moved again by the same increment and the same callbacks are called.

In slider:
Activates the interactive dragging of the slider.

BSelect Motion or **BTransfer Motion**

If the button press occurs within the slider, the subsequent motion events move the slider to the position of the pointer and call the callbacks for **XmNdragCallback**.

BSelect Release or **BTransfer Release**

If the button press occurs within the slider and the slider position is changed, the callbacks for **XmNvalueChangedCallback** are called.

MCtrl BSelect Press

In the region between an end of the Scale and the slider:
Moves the slider to that end of the Scale and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_RIGHT or XmMAX_ON_BOTTOM, movement toward the right or bottom increments the Scale value, and movement toward the left or top decrements the Scale value. If **XmNprocessingDirection** is XmMAX_ON_LEFT or XmMAX_ON_TOP, movement toward the right or bottom decrements the Scale value, and movement toward the left or top increments the Scale value.

KUp

For vertical Scales, moves the slider up one increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_TOP, movement toward the top increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_BOTTOM, movement toward the top decrements the Scale value.

KDown
For vertical Scales, moves the slider down one increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_BOTTOM, movement toward the bottom increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_TOP, movement toward the bottom decrements the Scale value.

KLeft
For horizontal Scales, moves the slider one increment to the left and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_LEFT, movement toward the left increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_RIGHT, movement toward the left decrements the Scale value.

KRight
For horizontal Scales, moves the slider one increment to the right and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_RIGHT, movement toward the right increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_LEFT, movement toward the right decrements the Scale value.

MCtrl KUp or **KPageUp**
For vertical Scales, moves the slider up one multiple increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_TOP, movement toward the top increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_BOTTOM, movement toward the top decrements the Scale value.

MCtrl KDown or **KPageDown**
For vertical Scales, moves the slider down one multiple increment and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_BOTTOM, movement toward the bottom increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_TOP, movement toward the bottom decrements the Scale value.

MCtrl KLeft or **KPageLeft**
For horizontal Scales, moves the slider one multiple increment to the left and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_LEFT, movement toward the left increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_RIGHT, movement toward the left decrements the Scale value.

MCtrl KRight or **KPageRight**
For horizontal Scales, moves the slider one multiple increment to the right and calls the **XmNvalueChangedCallback** callbacks. If **XmNprocessingDirection** is XmMAX_ON_RIGHT, movement toward the right increments the Scale value. If **XmNprocessingDirection** is XmMAX_ON_LEFT, movement toward the right decrements the Scale value.

KBeginLine or **KBeginData**
Moves the slider to the minimum value and calls the **XmNvalueChangedCallback** callbacks.

KEndLine or **KEndData**
Moves the slider to the maximum value and calls the **XmNvalueChangedCallback** callbacks.

KNextField
Traverses to the first item in the next tab group. If the current tab group is the last entry in the tab group list, it wraps to the beginning of the tab group list.

KPrevField
Traverses to the first item in the previous tab group. If the beginning of the tab group list is reached, it wraps to the end of the tab group list.

KHelp
Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

Virtual Bindings

The bindings for virtual keys are vendor specific.

SEE ALSO

Composite, *Constraint*, *Core*, *XmCreateScale*( ), *XmManager*, *XmScaleGetValue*( ) and *XmScaleSetValue*( ).

NAME

XmScaleGetValue — a Scale function that returns the current slider position.

SYNOPSIS

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

void XmScaleGetValue(
      Widget                    widget,
      int                      *value_return);
```

DESCRIPTION

XmScaleGetValue( ) returns the current slider position value displayed in the scale.

widget Specifies the Scale widget ID.

value_return Returns the current slider position value.

For a complete definition of Scale and its associated resources, see *XmScale*.

SEE ALSO

XmScale.

NAME

XmScaleSetValue — a Scale function that sets a slider value

SYNOPSIS

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

void XmScaleSetValue(
      Widget                    widget,
      int                       value);
```

DESCRIPTION

XmScaleSetValue( ) sets the slider *value* within the Scale widget.

widget Specifies the Scale widget ID.

value Specifies the slider position along the scale. This sets the **XmNvalue** resource.

For a complete definition of Scale and its associated resources, see *XmScale*.

SEE ALSO

XmScale.

NAME

XmScreen — the Screen widget class

SYNOPSIS

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

DESCRIPTION

The Screen 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 Screen object for each screen that it accesses.

A Screen 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 a Screen object by any other means. An application can use the function *XmGetXmScreen*( ) to obtain the widget ID of the Screen object for a given screen.

An application cannot supply initial values for Screen 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 Screen 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 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).

XmScreen **Resource Set**

Name	**Class**	**Type**	**Default**	**Access**
XmNmenuCursor	**XmCCursor**	**Cursor**	arrow	C
XmNunpostBehavior	**XmCUnpostBehavior**	**unsigned char**	XmUNPOST_AND_REPLAY	CSG
XmNhorizontalFontUnit	**XmCHorizontalFontUnit**	**int**	dynamic	CSG
XmNverticalFontUnit	**XmCVerticalFontUnit**	**int**	dynamic	CSG

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"
```

This resource uses the Xt String-to-Cursor converter, as documented in the **X Toolkit Intrinsics** specification.

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.

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 *XmGadget*, *XmManager* or *XmPrimitive* 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.

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 *XmGadget*, *XmManager* or *XmPrimitive* 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).

SEE ALSO

Core, *XmDisplay* and *XmGetXmScreen*( ).

NAME

XmScrollBar — the ScrollBar widget class

SYNOPSIS

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

DESCRIPTION

The ScrollBar widget allows the user to view data that is too large to be displayed all at once. ScrollBars are usually located inside a ScrolledWindow and adjacent to the widget that contains the data to be viewed. When the user interacts with the ScrollBar, the data within the other widget scrolls.

A ScrollBar consists of two arrows placed at each end of a rectangle. The rectangle is called the scroll region. A smaller rectangle, called the slider, is placed within the scroll region. The data is scrolled by clicking either arrow, selecting on the scroll region, or dragging the slider. When an arrow is selected, the slider within the scroll region is moved in the direction of the arrow by an amount supplied by the application. If the mouse button is held down, the slider continues to move at a constant rate.

The ratio of the slider size to the scroll region size typically corresponds to the relationship between the size of the visible data and the total size of the data. For example, if 10 percent of the data is visible, the slider typically occupies 10 percent of the scroll region. This provides the user with a visual clue to the size of the invisible data.

If the ScrollBar parent is an automatic ScrolledWindow, the **XmNtraversalOn** default is True. Otherwise, the default is False.

Classes

ScrollBar inherits behavior and resources from the *Core* and *XmPrimitive* classes.

The class pointer is **xmScrollBarWidgetClass**.

The class name is *XmScrollBar*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmScrollBar **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNdecrementCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNdragCallback	**XmCCallback**	**XtCallbackList**	NULL	C
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
XmNpageDecrementCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNpageIncrement	**XmCPageIncrement**	**int**	10	CSG
XmNpageIncrementCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNprocessingDirection	**XmCProcessingDirection**	**unsigned char**	dynamic	CSG
XmNrepeatDelay	**XmCRepeatDelay**	**int**	50 ms	CSG
XmNshowArrows	**XmCShowArrows**	**XtEnum**	XmEACH_SIDE	CSG
XmNsliderSize	**XmCSliderSize**	**int**	dynamic	CSG
XmNtoBottomCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNtoTopCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNtroughColor	**XmCTroughColor**	**Pixel**	dynamic	CSG
XmNvalue	**XmCValue**	**int**	dynamic	CSG
XmNvalueChangedCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmNdecrementCallback

Specifies the list of callbacks 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 called on each incremental change of position when the slider is being dragged. The reason sent by the callback is XmCR_DRAG.

XmNincrement

Specifies the amount by which the value increases or decreases when the user takes an action that moves the slider by one increment. The actual change in value is the lesser of **XmNincrement** and (previous **XmNvalue** − **XmNminimum**) when the slider moves to the end of the ScrollBar with the minimum value, and the lesser of **XmNincrement** and (**XmNmaximum** − **XmNsliderSize** − previous **XmNvalue**) when the slider moves to the end of the ScrollBar with the maximum value. The value of this resource must be greater than 0 (zero).

XmNincrementCallback

Specifies the list of callbacks 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 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 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 implementation-dependent. If the ScrollBar is oriented horizontally, the default value is implementation-dependent.

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.

XmNONE

Indicates that no arrows are displayed.

XmEACH_SIDE is the default value.

XmNsliderSize

Specifies the length of the slider between the values of 1 and (**XmNmaximum** – **XmNminimum**). The value is constrained to be within these inclusive bounds. The default value is (**XmNmaximum** – **XmNminimum**) divided by 10, with a minimum of 1.

XmNtoBottomCallback

Specifies the list of callbacks 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 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 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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED_PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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.

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( )

MCtrl BSelect Press in an arrow or in the scroll region between an arrow and the slider moves the slider as far as possible in the direction of the arrow. If **XmNprocessingDirection** is XmMAX_ON_RIGHT or XmMAX_ON_BOTTOM, movement toward the right or bottom calls the callbacks for **XmNtoBottomCallback**, and movement to the left or top calls the callbacks for **XmNtoTopCallback**. If **XmNprocessingDirection** is XmMAX_ON_LEFT or XmMAX_ON_TOP, movement toward the right or bottom calls the callbacks for **XmNtoTopCallback**, and movement to the left or top calls the callbacks for **XmNtoBottomCallback**. The **XmNvalueChangedCallback** is called if the **XmNtoTopCallback** or **XmNtoBottomCallback** is NULL. Pressing **KBeginLine** or **KBeginData** moves the slider to the minimum value and invokes the **XmNtoTopCallback**. Pressing **KEndLine** or **KEndData** moves the slider to the maximum value and invokes the **XmNtoBottomCallback**.

SEE ALSO

Core, *XmCreateScrollBar*( ), *XmPrimitive*, *XmScrollBarGetValues*( ) and *XmScrollBarSetValues*( ).

NAME

XmScrollBarGetValues — a ScrollBar function that returns the ScrollBar's increment values

SYNOPSIS

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

void XmScrollBarGetValues(
      Widget                     widget,
      int                       *value_return,
      int                       *slider_size_return,
      int                       *increment_return,
      int                       *page_increment_return);
```

DESCRIPTION

XmScrollBarGetValues( ) returns the the ScrollBar's increment values. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main ScrollBar or set slider function attributes.

widget Specifies the ScrollBar widget ID.

value_return Returns the ScrollBar's slider position between the **XmNminimum** and **XmNmaximum** resources.

slider_size_return
Returns the size of the slider as a value between 0 (zero) and the absolute value of **XmNmaximum** minus **XmNminimum**. The size of the slider varies, depending on how much of the slider scroll area it represents.

increment_return
Returns the amount of increment and decrement.

page_increment_return
Returns the amount of page increment and decrement.

For a complete definition of ScrollBar and its associated resources, see *XmScrollBar*.

RETURN VALUE

Returns the ScrollBar's increment values.

SEE ALSO

XmScrollBar.

NAME

XmScrollBarSetValues — a ScrollBar function that changes ScrollBar's increment values and the slider's size and position

SYNOPSIS

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

void XmScrollBarSetValues(
     Widget                    widget,
     int                       value,
     int                       slider_size,
     int                       increment,
     int                       page_increment,
     Boolean                   notify);
```

DESCRIPTION

XmScrollBarSetValues( ) changes the ScrollBar's increment values and the slider's size and position. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main ScrollBar or set slider function attributes.

widget Specifies the ScrollBar widget ID.

value Specifies the ScrollBar's slider position. Refer to the **XmNvalue** resource described in *XmScrollBar* on page 517.

slider_size Specifies the size of the slider. Refer to the **XmNsliderSize** resource described in *XmScrollBar* on page 517. This argument sets that resource.

increment Specifies the amount of button increment and decrement. Refer to the **XmNincrement** resource described in *XmScrollBar* on page 517. This argument sets that resource.

page_increment
Specifies the amount of page increment and decrement. Refer to the **XmNpageIncrement** resource described in *XmScrollBar* on page 517. This argument sets that resource.

notify Specifies a Boolean value that, when True, indicates a change in the ScrollBar value and also specifies that the ScrollBar widget automatically activates the **XmNvalueChangedCallback** with the recent change. If it is set to False, it specifies any change that has occurred in the ScrollBar's value, but does not activate **XmNvalueChangedCallback**.

For a complete definition of ScrollBar and its associated resources, see *XmScrollBar*.

SEE ALSO

XmScrollBar.

NAME

XmScrollVisible — a ScrolledWindow function that makes an invisible descendant of a ScrolledWindow work area visible

SYNOPSIS

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

void XmScrollVisible(
      Widget                    scrollw_widget,
      Widget                    widget,
      Dimension                 left_right_margin,
      Dimension                 top_bottom_margin);
```

DESCRIPTION

XmScrollVisible( ) makes an obscured or partially obscured widget or gadget descendant of a ScrolledWindow work area visible. The function repositions the work area and sets the specified margins between the widget and the nearest viewport boundary. The widget's location relative to the viewport determines whether one or both of the margins must be adjusted. This function requires that the **XmNscrollingPolicy** of the ScrolledWindow widget be set to XmAUTOMATIC.

scrollw_widget
Specifies the ID of the ScrolledWindow widget whose work area window contains an obscured descendant.

widget Specifies the ID of the widget to be made visible.

left_right_margin
Specifies the margin to establish between the left or right edge of the widget and the associated edge of the viewport. This margin is established only if the widget must be moved horizontally to make it visible.

top_bottom_margin
Specifies the margin to establish between the top or bottom edge of the widget and the associated edge of the viewport. This margin is established only if the widget must be moved vertically to make it visible.

For a complete definition of ScrolledWindow and its associated resources, see *XmScrolledWindow*.

SEE ALSO

XmScrolledWindow.

NAME

XmScrolledWindow — the ScrolledWindow widget class

SYNOPSIS

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

DESCRIPTION

The ScrolledWindow widget combines one or two ScrollBar widgets and a viewing area to implement a visible window onto some other (usually larger) data display. The visible part of the window can be scrolled through the larger display by the use of ScrollBars.

To use ScrolledWindow, an application first creates a ScrolledWindow widget, any needed ScrollBar widgets and a widget capable of displaying any desired data as the work area of ScrolledWindow. ScrolledWindow positions the work area widget and displays the ScrollBars if so requested. When the user performs some action on the ScrollBar, the application is notified through the normal ScrollBar callback interface.

ScrolledWindow can be configured to operate automatically so that it performs all scrolling and display actions with no need for application program involvement. It can also be configured to provide a minimal support framework in which the application is responsible for processing all user input and making all visual changes to the displayed data in response to that input.

When ScrolledWindow is performing automatic scrolling, it creates a clipping window. 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.

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

Classes

ScrolledWindow inherits behavior and resources from *Core*, *Composite*, *Constraint* and *XmManager*.

The class pointer is **xmScrolledWindowWidgetClass**.

The class name is *XmScrolledWindow*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmScrolledWindow **Resource Set**

Name	**Class**	**Type**	**Default**	**Access**
XmNclipWindow	**XmCClipWindow**	**Widget**	dynamic	G
XmNhorizontalScrollBar	**XmCHorizontalScrollBar**	**Widget**	dynamic	CSG
XmNscrollBarDisplayPolicy	**XmCScrollBarDisplayPolicy**	**unsigned char**	dynamic	CSG
XmNscrollBarPlacement	**XmCScrollBarPlacement**	**unsigned char**	XmBOTTOM_RIGHT	CSG
XmNscrolledWindowMarginHeight	**XmCScrolledWindowMarginHeight**	**Dimension**	0	N/A
XmNscrolledWindowMarginWidth	**XmCScrolledWindowMarginWidth**	**Dimension**	0	N/A
XmNscrollingPolicy	**XmCScrollingPolicy**	**unsigned char**	XmAPPLICATION_DEFINED	CG
XmNspacing	**XmCSpacing**	**Dimension**	4	CSG
XmNtraverseObscuredCallback	**XmCCallback**	**XtCallbackList**	NULL	CSG
XmNverticalScrollBar	**XmCVerticalScrollBar**	**Widget**	dynamic	CSG
XmNvisualPolicy	**XmCVisualPolicy**	**unsigned char**	dynamic	G
XmNworkWindow	**XmCWorkWindow**	**Widget**	NULL	CSG

XmNclipWindow

Specifies the widget ID of the clipping area. This is automatically created by ScrolledWindow when the **XmNvisualPolicy** resource is set to XmCONSTANT and can only be read by the application. Any attempt to set this resource to a new value causes a warning message to be printed by the scrolled window. If the **XmNvisualPolicy** resource is set to XmVARIABLE, this resource is set to NULL, and no clipping window is created.

XmNhorizontalScrollBar

Specifies the widget ID of the horizontal ScrollBar. This is automatically created by ScrolledWindow when the **XmNscrollingPolicy** is initialized to XmAUTOMATIC; otherwise, the default is NULL.

XmNscrollBarDisplayPolicy

Controls the automatic placement of the ScrollBars. If it is set to XmAS_NEEDED and if **XmNscrollingPolicy** is set to XmAUTOMATIC, ScrollBars are displayed only if the workspace exceeds the clip area in one or both dimensions. A resource value of XmSTATIC causes the ScrolledWindow to display the ScrollBars whenever they are managed, regardless of the relationship between the clip window and the work area. This resource must be XmSTATIC when **XmNscrollingPolicy** is XmAPPLICATION_DEFINED. The default is XmAS_NEEDED when **XmNscrollingPolicy** is XmAUTOMATIC, and XmSTATIC otherwise.

XmNscrollBarPlacement
Specifies the positioning of the ScrollBars in relation to the work window. The values are:

XmTOP_LEFT
The horizontal ScrollBar is placed above the work window; the vertical ScrollBar to is placed the left.

XmBOTTOM_LEFT
The horizontal ScrollBar is placed below the work window; the vertical ScrollBar to is placed the left.

XmTOP_RIGHT
The horizontal ScrollBar is placed above the work window; the vertical ScrollBar to is placed the right.

XmBOTTOM_RIGHT
The horizontal ScrollBar is placed below the work window; the vertical ScrollBar to is placed the right.

XmNscrolledWindowMarginHeight
Specifies the margin height on the top and bottom of the ScrolledWindow.

XmNscrolledWindowMarginWidth
Specifies the margin width on the right and left sides of the ScrolledWindow.

XmNscrollingPolicy
Performs automatic scrolling of the work area with no application interaction. If the value of this resource is XmAUTOMATIC, ScrolledWindow automatically creates the ScrollBars; attaches callbacks to the ScrollBars; sets the visual policy to XmCONSTANT; and automatically moves the work area through the clip window in response to any user interaction with the ScrollBars. An application can also add its own callbacks to the ScrollBars. This allows the application to be notified of a scroll event without having to perform any layout procedures.

Note: Since the ScrolledWindow adds callbacks to the ScrollBars, an application should not perform an *XtRemoveAllCallbacks*( ) on any of the ScrollBar widgets.

When **XmNscrollingPolicy** is set to XmAPPLICATION_DEFINED, the application is responsible for all aspects of scrolling. The ScrollBars must be created by the application, and it is responsible for performing any visual changes in the work area in response to user input.

This resource must be set to the desired policy at the time the ScrolledWindow is created. It cannot be changed through **SetValues**.

XmNspacing
Specifies the distance that separates the ScrollBars from the work window.

XmNtraverseObscuredCallback
Specifies a list of callbacks 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, it is not possible to traverse to an obscured widget. The callback reason is XmCR_OBSCURED_TRAVERSAL.

XmNverticalScrollBar
Specifies the widget ID of the vertical ScrollBar. This is automatically created by ScrolledWindow when the **XmNscrollingPolicy** is initialized to XmAUTOMATIC; otherwise, the default is NULL.

XmNvisualPolicy
Enlarges the ScrolledWindow to match the size of the work area. It can also be used as a static viewport onto a larger data space. If the visual policy is XmVARIABLE, the ScrolledWindow forces the ScrollBar display policy to XmSTATIC and allows the work area to grow or shrink at any time and adjusts its layout to accommodate the new size. When the policy is XmCONSTANT, the work area grows or shrinks as requested, but a clipping window forces the size of the visible portion to remain constant. The only time the viewing area can grow is in response to a resize from the ScrolledWindow's parent. The default is XmCONSTANT when **XmNscrollingPolicy** is XmAUTOMATIC, and XmVARIABLE otherwise.

Note: This resource must be set to the desired policy at the time the ScrolledWindow is created. It cannot be changed through *SetValues*( ).

XmNworkWindow Specifies the widget ID of the viewing area.

Inherited Resources

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

XmManager **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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

ScrolledWindow defines no new callback structures. The application must use the ScrollBar callbacks to be notified of user input.

SEE ALSO

Composite, *Constraint*, *Core*, *XmCreateScrolledWindow*( ), *XmManager* and *XmScrolledWindowSetAreas*( ).

NAME

XmScrolledWindowSetAreas — a ScrolledWindow function that adds or changes a window work region and a horizontal or vertical ScrollBar widget to the ScrolledWindow widget

SYNOPSIS

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

void XmScrolledWindowSetAreas(
      Widget                    widget,
      Widget                    horizontal_scrollbar,
      Widget                    vertical_scrollbar,
      Widget                    work_region);
```

DESCRIPTION

XmScrolledWindowSetAreas( ) adds or changes a window work region and a horizontal or vertical ScrollBar widget to the ScrolledWindow widget for the application. Each widget is optional and may be passed as NULL.

widget Specifies the ScrolledWindow widget ID.

horizontal_scrollbar
Specifies the ScrollBar widget ID for the horizontal ScrollBar to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The resource name associated with this argument is **XmNhorizontalScrollBar**.

vertical_scrollbar
Specifies the ScrollBar widget ID for the vertical ScrollBar to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The resource name associated with this argument is **XmNverticalScrollBar**.

work_region Specifies the widget ID for the work window to be associated with the ScrolledWindow widget. Set this ID only after creating an instance of the ScrolledWindow widget. The attribute name associated with this argument is **XmNworkWindow**.

For a complete definition of ScrolledWindow and its associated resources, see *XmScrolledWindow*.

SEE ALSO

XmScrolledWindow.

NAME

XmSelectionBox — the SelectionBox widget class

SYNOPSIS

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

DESCRIPTION

SelectionBox is a general dialog widget that allows the user to select one item from a list. By default, a SelectionBox includes the following:

- a scrolling list of alternatives
- an editable text field for the selected alternative
- labels for the list and text field
- three or four buttons.

The default button labels are **OK**, **Cancel** and **Help**. By default an **Apply** button is also created; if the parent of the SelectionBox is a DialogShell, it is managed; otherwise it is unmanaged. Additional children may be added to the SelectionBox after creation.

The 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.

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 selects
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	*Text* or *XmTextField*	selection box containing text of selected item

Classes

SelectionBox inherits behavior and resources from *Core*, *Composite*, *Constraint*, *XmManager* and *XmBulletinBoard*.

The class pointer is **xmSelectionBoxWidgetClass**.

The class name is *XmSelectionBox*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmSelectionBox **Resource Set**

Name	Class	Type	Default	Access
XmNapplyCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNapplyLabelString	**XmCApplyLabelString**	**XmString**	dynamic	N/A
XmNcancelCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNcancelLabelString	**XmCCancelLabelString**	**XmString**	dynamic	N/A
XmNchildPlacement	**XmCChildPlacement**	**unsigned char**	XmPLACE _ABOVE_ SELECTION	CSG
XmNdialogType	**XmCDialogType**	**unsigned char**	XmDIALOG _COMMAND	G
XmNhelpLabelString	**XmCHelpLabelString**	**XmString**	dynamic	N/A
XmNlistItemCount	**XmCItemCount**	**int**	0	CSG
XmNlistItems	**XmCItems**	**XmStringTable**	NULL	CSG
XmNlistLabelString	**XmCListLabelString**	**XmString**	NULL	N/A
XmNlistVisibleItemCount	**XmCVisibleItemCount**	**int**	dynamic	CSG
XmNminimizeButtons	**XmCMinimizeButtons**	**Boolean**	False	N/A
XmNmustMatch	**XmCMustMatch**	**Boolean**	False	N/A
XmNnoMatchCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNokCallback	**XmCCallback**	**XtCallbackList**	NULL	N/A
XmNokLabelString	**XmCOkLabelString**	**XmString**	dynamic	N/A
XmNselectionLabelString	**XmCSelectionLabelString**	**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**.

XmNcancelCallback

Specifies the list of callbacks called when the user activates the **Cancel** button. The callback reason is XmCR_CANCEL.

XmNcancelLabelString

Specifies the string label for the **Cancel** button. The default for this resource depends on the locale. In the C locale the default is **Cancel**.

XmNchildPlacement

Specifies the placement of the work area child. The possible values are:

XmPLACE_ABOVE_SELECTION

Places the work area child above the Text area

XmPLACE_BELOW_SELECTION

Places the work area child below the Text area

XmPLACE_TOP

Places the work area child above the List area, and below a MenuBar, if one is present

XmNdialogType

Determines the set of SelectionBox children widgets that are created and managed at initialization. The possible values are:

XmDIALOG_PROMPT

All standard children except the list and list label are created, and all except the **Apply** button are managed.

XmDIALOG_COMMAND

Only the list, the selection label and the text field are created and managed.

XmDIALOG_SELECTION

All standard children are created and managed.

XmDIALOG_FILE_SELECTION

All standard children are created and managed.

XmDIALOG_WORK_AREA

All standard children are created, and all except the **Apply** button are managed.

If the parent of the SelectionBox is a DialogShell, the default is XmDIALOG_SELECTION; otherwise, the default is XmDIALOG_WORK_AREA. *XmCreatePromptDialog*( ) and *XmCreateSelectionDialog*( ) set and append this resource to the creation *arglist* supplied by the application. This resource cannot be modified after creation.

XmNhelpLabelString

Specifies the string label for the **Help** button. The default for this resource depends on the locale. In the C locale the default is **Help**.

XmNlistItems

Specifies the items in the SelectionBox list.

XmNlistItemCount

Specifies the number of items in the SelectionBox list. The value must not be negative.

XmNlistLabelString

Specifies the string label to appear above the SelectionBox list containing the selection items. The default for this resource depends on the locale. In the C locale the default is **Items** unless **XmNdialogType** is XmDIALOG_PROMPT; in this case the default is NULL.

XmNlistVisibleItemCount

Specifies the number of items displayed in the SelectionBox list. The value must be greater than 0 (zero) unless **XmNdialogType** is XmDIALOG_PROMPT; in this case, the value is always 0. The default is dynamic based on the height of the list.

XmNminimizeButtons
Sets the buttons to the width of the widest button and height of the tallest button if False. If True, button width and height are not modified.

XmNmustMatch
Specifies whether the selection widget should check if the user's selection in the text edit field has an exact match in the SelectionBox list when the **OK** button is activated. If the selection does not have an exact match, and **XmNmustMatch** is True, the **XmNnoMatchCallback** callbacks are called. If the selection does have an exact match or if **XmNmustMatch** is False, **XmNokCallback** callbacks are called.

XmNnoMatchCallback
Specifies the list of callbacks called when the user makes a selection from the text edit field that does not have an exact match with any of the items in the list box. The callback reason is XmCR_NO_MATCH. Callbacks in this list are called only if **XmNmustMatch** is true.

XmNokCallback
Specifies the list of callbacks called when the user activates the **OK** button. The callback reason is XmCR_OK. If the selection text does not match a list item, and **XmNmustMatch** is True, the **XmNnoMatchCallback** callbacks are called instead.

XmNokLabelString
Specifies the string label for the **OK** button. The default for this resource depends on the locale. In the C locale the default is **OK**.

XmNselectionLabelString
Specifies the string label for the selection text edit field. The default for this resource depends on the locale. In the C locale the default is **Selection**.

XmNtextAccelerators
Specifies translations added to the Text widget child of the SelectionBox. The default includes bindings for the up and down keys for auto selection of list items. This resource is ignored if **XmNaccelerators** is initialized to a non-default 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**	False	N/A
XmNbuttonFontList	**XmCButtonFontList**	**XmFontList**	dynamic	N/A
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
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
XmNunmapCallback	**XmCCallback**	**XtCallbackList**	NULL	C

XmManager **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**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	SG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmTAB_GROUP	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	dynamic	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 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.

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.

SEE ALSO

Composite, *Constraint*, *Core*, *XmBulletinBoard*, *XmCreateSelectionBox*( ), *XmCreateSelectionDialog*( ), *XmCreatePromptDialog*( ), *XmManager* and *XmSelectionBoxGetChild*( ).

NAME

XmSelectionBoxGetChild — a SelectionBox function that is used to access a component

SYNOPSIS

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

Widget XmSelectionBoxGetChild(
      Widget                    widget,
      unsigned char             child);
```

DESCRIPTION

XmSelectionBoxGetChild( ) is used to access a component within a SelectionBox. The parameters given to the function are the SelectionBox widget and a value indicating which component to access.

widget Specifies the SelectionBox widget ID.

child Specifies a component within the SelectionBox. The following values are legal for this parameter:

XmDIALOG_APPLY_BUTTON
XmDIALOG_CANCEL_BUTTON
XmDIALOG_DEFAULT_BUTTON
XmDIALOG_HELP_BUTTON
XmDIALOG_LIST
XmDIALOG_LIST_LABEL
XmDIALOG_OK_BUTTON
XmDIALOG_SELECTION_LABEL
XmDIALOG_SEPARATOR
XmDIALOG_TEXT
XmDIALOG_WORK_AREA.

For a complete definition of SelectionBox and its associated resources, see *XmSelectionBox*.

RETURN VALUE

Returns the widget ID of the specified SelectionBox component. An application should not assume that the returned widget is of any particular class.

SEE ALSO

XmSelectionBox.

NAME

XmSeparator — the Separator widget class

SYNOPSIS

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

DESCRIPTION

Separator is a primitive widget that separates items in a display. Several different line drawing styles are provided, as well as horizontal or vertical orientation.

The Separator line drawing is automatically centered within the height of the widget for a horizontal orientation and centered within the width of the widget for a vertical orientation. An *XtSetValues* with a new **XmNseparatorType** resizes the widget to its minimal height (for horizontal orientation) or its minimal width (for vertical orientation) unless height or width is explicitly set in the *XtSetValues*( ) call.

Separator does not draw shadows around the separator. The Primitive resource **XmNshadowThickness** is used for the Separator's thickness when **XmNseparatorType** is XmSHADOW_ETCHED_IN or XmSHADOW_ETCHED_OUT.

Separator does not highlight and allows no traversing. The primitive resource **XmNtraversalOn** is forced to False.

The **XmNseparatorType** of XmNO_LINE provides an escape to the application programmer who needs a different style of drawing. A pixmap the height of the widget can be created and used as the background pixmap by building an argument list using the **XmNbackgroundPixmap** argument type as defined by *Core*. Whenever the widget is redrawn, its background is displayed containing the desired separator drawing.

Classes

Separator inherits behavior and resources from *Core* and *XmPrimitive*.

The class pointer is **xmSeparatorWidgetClass**.

The class name is *XmSeparator*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix 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).

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
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**. For horizontal orientation, the top line is drawn in **XmNtopShadowColor** and the bottom line is drawn in **XmNbottomShadowColor**. For vertical orientation, the left line is drawn in **XmNtopShadowColor** and the right line is drawn in **XmNbottomShadowColor**.

XmSHADOW_ETCHED_OUT
Double line giving the effect of an etched line coming out from the window. The thickness of the double line is equal to the value of **XmNshadowThickness**. For horizontal orientation, the top line is drawn in **XmNbottomShadowColor** and the bottom line is drawn in **XmNtopShadowColor**. For vertical orientation, the left line is drawn in **XmNbottomShadowColor** and the right line is drawn in **XmNtopShadowColor**.

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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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

SEE ALSO

Core, *XmCreateSeparator*( ) and *XmPrimitive*.

NAME

XmSeparatorGadget — the SeparatorGadget widget class

SYNOPSIS

```
#include <Xm/SeparatorG.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 or XmSHADOW_ETCHED_OUT.

SeparatorGadget does not highlight and allows no traversing. The Gadget resource **XmNtraversalOn** is forced to False.

Classes

SeparatorGadget inherits behavior and resources from *Object*, *RectObj* and *XmGadget*.

The class pointer is **xmSeparatorGadgetClass**.

The class name is *XmSeparatorGadget*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

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
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**. For horizontal orientation, the top line is drawn in **XmNtopShadowColor** and the bottom line is drawn in **XmNbottomShadowColor**. For vertical orientation, the left line is drawn in **XmNtopShadowColor** and the right line is drawn in **XmNbottomShadowColor**.

XmSHADOW_ETCHED_OUT
Double line giving the effect of an etched line coming out from the window. The thickness of the double line is equal to the value of **XmNshadowThickness**. For horizontal orientation, the top line is drawn in **XmNbottomShadowColor** and the bottom line is drawn in **XmNtopShadowColor**. For vertical orientation, the left line is drawn in **XmNbottomShadowColor** and the right line is drawn in **XmNtopShadowColor**.

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**
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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

SEE ALSO

Object, *RectObject*, *XmCreateSeparatorGadget*( ) and *XmGadget*.

NAME

XmSetMenuCursor — a RowColumn function that modifies the menu cursor for a client

SYNOPSIS

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

void XmSetMenuCursor(
      Display                   *display,
      Cursor                     cursorId);
```

DESCRIPTION

XmSetMenuCursor( ) programmatically modifies the menu cursor for a client; after the cursor has been created by the client, this function registers the cursor with the menu system. After calling this function, the specified cursor is displayed whenever this client displays a Motif menu on the indicated display. The client can then specify different cursors on different displays.

display Specifies the display to which the cursor is to be associated.

cursorId Specifies the X cursor ID.

For a complete definition of the menu cursor resource, see *XmRowColumn*.

SEE ALSO

XmRowColumn.

NAME

XmSetProtocolHooks — a *VendorShell* function that allows pre-actions and post-actions to be executed when a protocol message is received from MWM

SYNOPSIS

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

void XmSetProtocolHooks(
      Widget                     shell,
      Atom                       property,
      Atom                       protocol,
      XtCallbackProc             prehook,
      XtPointer                  pre_closure,
      XtCallbackProc             posthook,
      XtPointer                  post_closure);
```

DESCRIPTION

XmSetProtocolHooks( ) is used by shells that want to have pre-actions and post-actions executed when a protocol message is received from MWM. Since there is no guaranteed ordering in execution of event handlers or callback lists, this allows the shell to control the flow while leaving the protocol manager structures opaque.

shell Specifies the widget with which the protocol property is associated.

property Specifies the protocol property.

protocol Specifies the protocol atom (or an **int** cast to **Atom**).

prehook Specifies the procedure to call before calling entries on the client callback list.

pre_closure Specifies the client data to be passed to the pre-hook when it is invoked.

posthook Specifies the procedure to call after calling entries on the client callback list.

post_closure Specifies the client data to be passed to the post-hook when it is invoked.

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmInternAtom*( ) and *XmSetWMProtocolHooks*( ).

NAME

XmSetWMProtocolHooks — a *VendorShell* convenience interface that allows pre-actions and post-actions to be executed when a protocol message is received from the window manager

SYNOPSIS

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

void XmSetWMProtocolHooks(
      Widget                    shell,
      Atom                      protocol,
      XtCallbackProc            prehook,
      XtPointer                 pre_closure,
      XtCallbackProc            posthook,
      XtPointer                 post_closure);
```

DESCRIPTION

XmSetWMProtocolHooks( ) is a convenience interface. It calls *XmSetProtocolHooks*( ) with the property value set to the atom returned by interning WM_PROTOCOLS.

shell Specifies the widget with which the protocol property is associated.

protocol Specifies the protocol atom (or an **int** cast to **Atom**).

prehook Specifies the procedure to call before calling entries on the client callback list.

pre_closure Specifies the client data to be passed to the pre-hook when it is invoked.

posthook Specifies the procedure to call after calling entries on the client callback list.

post_closure Specifies the client data to be passed to the post-hook when it is invoked.

For a complete definition of *VendorShell* and its associated resources, see *VendorShell*.

SEE ALSO

VendorShell, *XmInternAtom*( ) and *XmSetProtocolHooks*( ).

NAME

XmStringBaseline — a compound string function that returns the number of pixels between the top of the character box and the baseline of the first line of text

SYNOPSIS

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

Dimension XmStringBaseline(
      XmFontList                  fontlist,
      XmString                    string);
```

DESCRIPTION

XmStringBaseline( ) returns the number of pixels between the top of the character box and the baseline of the first line of text in the provided compound string.

When *string* has been created with *XmStringCreateSimple*( ), the font associated with the character set derived from the current language environment must appear at the front of *fontlist*. Otherwise, the result of the function is undefined.

fontlist Specifies the font list.

string Specifies the string.

RETURN VALUE

Returns the number of pixels between the top of the character box and the baseline of the first line of text.

SEE ALSO

XmStringCreate( ) and *XmStringCreateSimple*( ).

NAME

XmStringByteCompare — a compound string function that indicates the results of a byte-by-byte comparison

SYNOPSIS

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

Boolean XmStringByteCompare(
      XmString                  s1,
      XmString                  s2);
```

DESCRIPTION

XmStringByteCompare( ) returns a **Boolean** type indicating the results of a byte-by-byte comparison of two compound strings.

In general, if two compound strings are created with the same (**char ***) string using *XmStringCreateLocalized*( ) in the same language environment, the compound strings compare as equal. If two compound strings are created with the same (**char ***) string and the same font list element tag set other than XmFONTLIST_DEFAULT_TAG using *XmStringCreate*( ), the strings compare as equal.

In some cases, once a compound string is put into a widget, that string is converted into an internal form to allow faster processing. Part of the conversion process strips out unnecessary or redundant information. If an application then does an *XtGetValues*( ) to retrieve a compound string from a widget (specifically, Label and all of its subclasses), it is not guaranteed that the compound string returned is byte-for-byte the same as the string given to the widget originally.

s1 Specifies a compound string to be compared with *s2*.

s2 Specifies a compound string to be compared with *s1*.

RETURN VALUE

Returns True if two compound strings are identical byte-by-byte.

SEE ALSO

XmStringCreate( ) and *XmStringCreateLocalized*( ).

NAME

XmStringCompare — a compound string function that compares two strings

SYNOPSIS

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

Boolean XmStringCompare(
      XmString                  s1,
      XmString                  s2);
```

DESCRIPTION

XmStringCompare( ) returns a **Boolean** type indicating the results of a semantically equivalent comparison of two compound strings.

Semantically equivalent means that the strings have the same text components, font list element tags, directions and separators. In general, if two compound strings are created with the same (**char ***) string using *XmStringCreateLocalized*( ) in the same language environment, the compound strings compare as equal. If two compound strings are created with the same (**char ***) string and the same font list element tag other than XmFONTLIST_DEFAULT_TAG using *XmStringCreate*( ), the strings compare as equal.

s1 Specifies a compound string to be compared with *s2*.

s2 Specifies a compound string to be compared with *s1*.

RETURN VALUE

Returns True if two compound strings are equivalent.

SEE ALSO

XmStringCreate( ) and *XmStringCreateLocalized*( ).

NAME

XmStringConcat — a compound string function that appends one string to another

SYNOPSIS

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

XmString XmStringConcat(
      XmString                 s1,
      XmString                 s2);
```

DESCRIPTION

XmStringConcat( ) copies *s2* to the end of *s1* and returns a copy of the resulting compound string. The original strings are preserved. The function allocates space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFree*( ).

s1 Specifies the compound string to which a copy of *s2* is appended.

s2 Specifies the compound string that is appended to the end of *s1*.

RETURN VALUE

Returns a new compound string.

SEE ALSO

XmStringCreate( ) and *XmStringFree*( ).

NAME

XmStringCopy — a compound string function that makes a copy of a string

SYNOPSIS

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

XmString XmStringCopy(
      XmString                  s1);
```

DESCRIPTION

XmStringCopy( ) makes a copy of a compound string. The function allocates space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFree*( ).

s1 Specifies the compound string to be copied.

RETURN VALUE

Returns a new compound string.

SEE ALSO

XmStringCreate( ) and *XmStringFree*( ).

NAME

XmStringCreate — a compound string function that creates a compound string

SYNOPSIS

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

XmString XmStringCreate(
      char                        *text,
      char                        *tag);
```

DESCRIPTION

XmStringCreate( ) creates a compound string with two components: text and a font list element tag. The function allocates space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFree*( ).

text Specifies a NULL-terminated string to be used as the text component of the compound string.

tag Specifies the font list element tag to be associated with the given text. The value XmFONTLIST_DEFAULT_TAG identifies a locale text segment.

RETURN VALUE

Returns a new compound string.

SEE ALSO

Section 4.3 on page 61, Section 4.4 on page 62, Section 4.5 on page 62, Section 4.2 on page 60, *XmFontListAppendEntry*( ), *XmFontListCopy*( ), *XmFontListEntryCreate*( ), *XmFontListEntryFree*( ), *XmFontListEntryGetFont*( ), *XmFontListEntryGetTag*( ), *XmFontListEntryLoad*( ), *XmFontListFree*( ), *XmFontListFreeFontContext*( ), *XmFontListInitFontContext*( ), *XmFontListNextEntry*( ), *XmFontListRemoveEntry*( ), *XmStringBaseline*( ), *XmStringByteCompare*( ), *XmStringCompare*( ), *XmStringConcat*( ), *XmStringCopy*( ), *XmStringCreateLocalized*( ), *XmStringCreateSimple*( ), *XmStringDraw*( ), *XmStringDrawImage*( ), *XmStringDrawUnderline*( ), *XmStringEmpty*( ), *XmStringExtent*( ), *XmStringFree*( ), *XmStringFreeContext*( ), *XmStringGetNextSegment*( ), *XmStringHasSubstring*( ), *XmStringHeight*( ), *XmStringInitContext*( ), *XmStringLength*( ), *XmStringLineCount*( ), *XmStringSegmentCreate*( ), *XmStringSeparatorCreate*( ) and *XmStringWidth*( ).

NAME

XmStringCreateLocalized — a compound string function that creates a compound string in the current locale

SYNOPSIS

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

XmString XmStringCreateLocalized(
      char                     *text);
```

DESCRIPTION

XmStringCreateLocalized( ) creates a compound string containing the specified text and assigns XmFONTLIST_DEFAULT_TAG as the font list entry tag. An identical compound string would result from the function *XmStringCreate*( ) called with XmFONTLIST_DEFAULT_TAG explicitly as the font list entry tag. The function allocates space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFree*( ).

text Specifies a NULL-terminated string of text encoded in the current locale to be used as the text component of the compound string.

RETURN VALUE

Returns a new compound string.

SEE ALSO

XmStringCreate( ).

NAME

XmStringCreateSimple — a compound string function that creates a compound string in the language environment of a widget

SYNOPSIS

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

XmString XmStringCreateSimple(
      char                     *text);
```

DESCRIPTION

XmStringCreateSimple( ) creates a compound string with two components: text and a character set. It derives the character set from the current language environment.

Note: This routine is obsolete and exists for compatibility with previous releases. It has been replaced by *XmStringCreateLocalized*( ).

text Specifies a NULL-terminated string to be used as the text component of the compound string.

RETURN VALUE

Returns a new compound string.

SEE ALSO

XmStringCreate( ) and *XmStringCreateLocalized*( ).

NAME

XmStringDraw — a compound string function that draws a compound string in an X window

SYNOPSIS

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

void XmStringDraw(
      Display               *d,
      Window                 w,
      XmFontList             fontlist,
      XmString               string,
      GC                     gc,
      Position               x,
      Position               y,
      Dimension              width,
      unsigned char          alignment,
      unsigned char          layout_direction,
      XRectangle            *clip);
```

DESCRIPTION

XmStringDraw( ) draws a compound string in an X Window.

When *string* has been created with *XmStringCreateSimple*( ), the font associated with the character set derived from the current language environment must appear at the front of *fontlist*. Otherwise, the result of the function is undefined.

d Specifies the display.

w Specifies the window.

fontlist Specifies the font list.

string Specifies the string.

gc Specifies the graphics context to use.

x Specifies a coordinate of the rectangle that contains the displayed compound string.

y Specifies a coordinate of the rectangle that contains the displayed compound string.

width Specifies the width of the rectangle that contains the displayed compound string.

alignment Specifies how the string is aligned within the specified rectangle. It is one of:

XmALIGNMENT_BEGINNING
XmALIGNMENT_CENTER
XmALIGNMENT_END.

layout_direction
Controls the direction in which the segments of the compound string are laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string is drawn. If the value is NULL, no clipping is done.

SEE ALSO

XmStringCreate( ) and *XmStringCreateSimple*( ).

NAME

XmStringDrawImage — a compound string function that draws a compound string in an X Window and creates an image

SYNOPSIS

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

void XmStringDrawImage(
      Display                 *d,
      Window                   w,
      XmFontList               fontlist,
      XmString                 string,
      GC                       gc,
      Position                 x,
      Position                 y,
      Dimension                width,
      unsigned char            alignment,
      unsigned char            layout_direction,
      XRectangle              *clip);
```

DESCRIPTION

XmStringDrawImage( ) draws a compound string in an X Window and paints both the foreground and background bits of each character.

When *string* has been created with *XmStringCreateSimple*( ), the font associated with the character set derived from the current language environment must appear at the front of *fontlist*. Otherwise, the result of the function is undefined.

d Specifies the display.

w Specifies the window.

fontlist Specifies the font list.

string Specifies the string.

gc Specifies the graphics context to use.

x Specifies a coordinate of the rectangle that contains the displayed compound string.

y Specifies a coordinate of the rectangle that contains the displayed compound string.

width Specifies the width of the rectangle that contains the displayed compound string.

alignment Specifies how the string is aligned within the specified rectangle. It is one of:

XmALIGNMENT_BEGINNING
XmALIGNMENT_CENTER
XmALIGNMENT_END.

layout_direction
Controls the direction in which the segments of the compound string will be laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string is drawn. If NULL, no clipping is done.

SEE ALSO

XmStringCreate( ) and *XmStringCreateSimple*( ).

NAME

XmStringDrawUnderline — a compound string function that underlines a string drawn in an X Window

SYNOPSIS

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

void XmStringDrawUnderline(
      Display                 *d,
      Window                   w,
      XmFontList               fontlist,
      XmString                 string,
      GC                       gc,
      Position                 x,
      Position                 y,
      Dimension                width,
      unsigned char            alignment,
      unsigned char            layout_direction,
      XRectangle              *clip,
      XmString                 underline);
```

DESCRIPTION

XmStringDrawUnderline( ) draws a compound string in an X Window. If the substring identified by *underline* can be matched in *string*, the substring is underlined. Once a match has occurred, no further matches or underlining are done.

When *string* has been created with *XmStringCreateSimple*( ), the font associated with the character set derived from the current language environment must appear at the front of *fontlist*. Otherwise, the result of the function is undefined.

d Specifies the display.

w Specifies the window.

fontlist Specifies the font list.

string Specifies the string.

gc Specifies the graphics context to use.

x Specifies a coordinate of the rectangle that contains the displayed compound string.

y Specifies a coordinate of the rectangle that contains the displayed compound string.

width Specifies the width of the rectangle that contains the displayed compound string.

alignment Specifies how the string is aligned within the specified rectangle. It is one of:

XmALIGNMENT_BEGINNING
XmALIGNMENT_CENTER
XmALIGNMENT_END.

layout_direction
Controls the direction in which the segments of the compound string are laid out. It also determines the meaning of the *alignment* parameter.

clip Allows the application to restrict the area into which the compound string is drawn. If it is NULL, no clipping is done.

underline Specifies the substring to be underlined.

SEE ALSO

XmStringCreate( ) and *XmStringCreateSimple*( ).

NAME

XmStringEmpty — a compound string function that provides information on the existence of non-zero-length text components

SYNOPSIS

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

Boolean XmStringEmpty(
      XmString                  s1);
```

DESCRIPTION

XmStringEmpty( ) returns a Boolean value indicating whether any non-zero-length text components exist in the provided compound string. It returns True if there are no text segments in the string. If this routine is passed NULL as the string, it returns True.

s1 Specifies the compound string.

RETURN VALUE

Returns True if there are no text segments in the string. If this routine is passed NULL as the string, it returns True.

SEE ALSO

XmStringCreate( ).

NAME

XmStringExtent — a compound string function that determines the size of the smallest rectangle that encloses the compound string

SYNOPSIS

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

void XmStringExtent(
      XmFontList                fontlist,
      XmString                  string,
      Dimension                *width,
      Dimension                *height);
```

DESCRIPTION

XmStringExtent( ) determines the width and height, in pixels, of the smallest rectangle that encloses the compound string provided.

When *string* has been created with *XmStringCreateSimple*( ), the font associated with the character set derived from the current language environment must appear at the front of *fontlist*. Otherwise, the result of the function is undefined.

fontlist Specifies the font list.

string Specifies the string.

width Specifies a pointer to the width of the rectangle.

height Specifies a pointer to the height of the rectangle.

SEE ALSO

XmStringCreate( ) and *XmStringCreateSimple*( ).

NAME

XmStringFree — a compound string function that recovers memory

SYNOPSIS

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

void XmStringFree(
      XmString                    string);
```

DESCRIPTION

XmStringFree( ) recovers memory used by a compound string.

string Specifies the compound string to be freed.

SEE ALSO

XmStringCreate( ).

NAME

XmStringFreeContext — a compound string function that instructs the toolkit that the context is no longer needed

SYNOPSIS

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

void XmStringFreeContext(
      XmStringContext           context);
```

DESCRIPTION

XmStringFreeContext( ) instructs the toolkit that the context is no longer needed and is not used without reinitialization.

context Specifies the string context structure allocated by the *XmStringInitContext*( ) function.

SEE ALSO

XmStringCreate( ) and *XmStringInitContext*( ).

NAME

XmStringGetNextSegment — a compound string function that fetches the octets in the next segment of a compound string

SYNOPSIS

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

Boolean XmStringGetNextSegment(
      XmStringContext          context,
      char                   **text,
      XmStringCharSet         *tag,
      XmStringDirection       *direction,
      Boolean                 *separator);
```

DESCRIPTION

XmStringGetNextSegment( ) fetches the octets in the next segment; repeated calls fetch sequential segments. The *text*, *tag* and *direction* of the fetched segment are returned each time. A **Boolean** type is returned to indicate whether a valid segment was successfully parsed. The function allocates space to hold the returned compound string in *text*. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFree*( ).

context Specifies the string context structure allocated by the *XmStringInitContext*( ) function.

text Specifies a pointer to a NULL-terminated string.

tag Specifies a pointer to the font list element tag associated with the text.

direction Specifies a pointer to the direction of the text.

separator Specifies whether the next component of the compound string is a separator.

RETURN VALUE

Returns True if a valid segment is found.

SEE ALSO

XmStringCreate( ) and *XmStringInitContext*( ).

NAME

XmStringHasSubstring — a compound string function that indicates whether one compound string is contained within another

SYNOPSIS

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

Boolean XmStringHasSubstring(
      XmString                  string,
      XmString                  substring);
```

DESCRIPTION

XmStringHasSubstring( ) indicates whether or not one compound string is contained within another.

string Specifies the compound string to be searched.

substring Specifies the compound string to be searched for.

RETURN VALUE

Returns True if *substring* has a single segment and if its text is completely contained within any single segment of *string*; otherwise, it returns False. If two compound strings created using *XmStringCreateLocalized*( ) in the same language environment satisfy this condition, the function returns True. If two compound strings created with the same character set using *XmStringCreate*( ) satisfy this condition, the function returns True.

SEE ALSO

XmStringCreate( ) *XmStringCreateSimple*( ) and *XmStringCreateLocalized*( ).

NAME

XmStringHeight — a compound string function that returns the line height of the given compound string

SYNOPSIS

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

Dimension XmStringHeight(
      XmFontList                fontlist,
      XmString                  string);
```

DESCRIPTION

XmStringHeight( ) returns the height, in pixels, of the sum of all the line heights of the given compound string. Separator components delimit lines.

When *string* has been created with *XmStringCreateSimple*( ), the font associated with the character set derived from the current language environment must appear at the front of *fontlist*. Otherwise, the result of the function is undefined.

fontlist Specifies the font list.

string Specifies the string.

RETURN VALUE

Returns the height of the specified string.

SEE ALSO

XmStringCreate( ) and *XmStringCreateSimple*( ).

NAME

XmStringInitContext — a compound string function that allows applications to read out the content segment by segment

SYNOPSIS

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

Boolean XmStringInitContext(
      XmStringContext          *context,
      XmString                  string);
```

DESCRIPTION

XmStringInitContext( ) maintains a context to allow applications to read out the contents of a compound string segment by segment. This function establishes the context for this read out. This context is used when reading subsequent segments out of the string. A **Boolean** type is returned to indicate if the input string could be parsed. The function allocates space to hold the returned context. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFreeContext*( ).

context Specifies a pointer to the allocated context.

string Specifies the string.

RETURN VALUE

Returns True if the context was allocated.

SEE ALSO

XmStringCreate( ).

NAME

XmStringLength — a compound string function that obtains the length of a compound string

SYNOPSIS

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

int XmStringLength(
      XmString                  s1);
```

DESCRIPTION

XmStringLength( ) obtains the length of a compound string. It returns the number of bytes in *s1* including all tags, direction indicators and separators. If the compound string has an invalid structure, 0 (zero) is returned.

s1   Specifies the compound string.

RETURN VALUE

Returns the length of the compound string.

SEE ALSO

XmStringCreate( ).

NAME

XmStringLineCount — a compound string function that returns the number of separators plus 1 in the provided compound string

SYNOPSIS

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

int XmStringLineCount(
      XmString                  string);
```

DESCRIPTION

XmStringLineCount( ) returns the number of separators plus one in the provided compound string. In effect, it counts the lines of text.

string Specifies the string.

RETURN VALUE

Returns the number of lines in the compound string.

SEE ALSO

XmStringCreate( ).

NAME

XmStringSegmentCreate — a compound string function that creates a compound string

SYNOPSIS

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

XmString XmStringSegmentCreate(
      char                   *text,
      char                   *tag,
      XmStringDirection       direction,
      Boolean                 separator);
```

DESCRIPTION

XmStringSegmentCreate( ) is a high-level function that assembles a compound string consisting of a font list element tag, a direction component, a text component, and an optional separator component. The function allocates space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFree*( ).

text Specifies a NULL-terminated string to be used as the text component of the compound string.

tag Specifies the font list element tag to be associated with the text. The value XmFONTLIST_DEFAULT_TAG identifies a segment encoded in the current locale.

direction Specifies the direction of the text.

separator Specifies separator addition. A value of False means the compound string does not have a separator at the end. A value of True means a separator immediately follows the text component.

RETURN VALUE

Returns a new compound string.

SEE ALSO

XmStringCreate( ).

NAME

XmStringSeparatorCreate — a compound string function that creates a compound string

SYNOPSIS

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

XmString XmStringSeparatorCreate();
```

DESCRIPTION

XmStringSeparatorCreate( ) creates a compound string with a single component, a separator. The function allocates space to hold the returned compound string. The application is responsible for managing the allocated space. The application can recover the allocated space by calling *XmStringFree*( ).

RETURN VALUE

Returns a new compound string.

SEE ALSO

XmStringCreate( ).

NAME

XmStringWidth — a compound string function that returns the width of the longest sequence of text components in a compound string

SYNOPSIS

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

Dimension XmStringWidth(
      XmFontList                  fontlist,
      XmString                    string);
```

DESCRIPTION

XmStringWidth( ) returns the width, in pixels, of the longest sequence of text components in the provided compound string. Separator components are used to delimit sequences of text components.

When *string* has been created with *XmStringCreateSimple*( ), the font associated with the character set derived from the current language environment must appear at the front of *fontlist*. Otherwise, the result of the function is undefined.

fontlist Specifies the font list.

string Specifies the string.

RETURN VALUE

Returns the width of the compound string.

SEE ALSO

XmStringCreate( ) and *XmStringCreateSimple*( ).

NAME

XmTargetsAreCompatible — a function that tests whether the target types match between a drop site and source object

SYNOPSIS

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

Boolean XmTargetsAreCompatible(
      Display                  *display,
      Atom                     *export_targets,
      Cardinal                  num_export_targets,
      Atom                     *import_targets,
      Cardinal                  num_import_targets);
```

DESCRIPTION

XmTargetsAreCompatible( ) determines whether the import targets of the destination match any of the export targets of a source. If there is at least one target in common, the function returns True.

display Specifies the display connection.

export_targets
Specifies the list of target atoms associated with the source object. This resource identifies the selection targets to which the source can convert.

num_export_targets
Specifies the number of entries in the list of export targets.

import_targets
Specifies the list of targets to be checked against the **XmNexportTargets** of the source associated with the specified DragContext.

num_import_targets
Specifies the number of entries in the *import_targets* list.

RETURN VALUE

Returns a **Boolean** type that indicates whether the destination targets are compatible with the source targets. If there is at least one target in common, the routine returns True; otherwise, it returns False.

SEE ALSO

XmDragContext and *XmDropSite*.

NAME

XmText — the Text widget class

SYNOPSIS

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

DESCRIPTION

Text provides a single-line and multi-line 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 **ICCCM** specification. Text supports primary and secondary selection.

Mouse Selection

The Text widget allows text to be edited, inserted and selected. The user can cut, copy and paste text using the clipboard, primary transfer or secondary transfer.

The insertion cursor, displayed as an I-beam, shows where input is inserted. Input is inserted just before the insertion cursor.

Classes

Text inherits behavior and resources from *Core* and *XmPrimitive*.

The class pointer is **xmTextWidgetClass**.

The class name is *XmText*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmText **Resource Set**

Name	**Class**	**Type**	**Default**	**Access**
XmNactivateCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNautoShowCursorPosition	**XmCAutoShowCursorPosition**	**Boolean**	True	CSG
XmNcursorPosition	**XmCCursorPosition**	**XmTextPosition**	0	CSG
XmNeditable	**XmCEditable**	**Boolean**	True	CSG
XmNeditMode	**XmCEditMode**	**int**	XmSINGLE_LINE_EDIT	CSG
XmNfocusCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNgainPrimaryCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNlosePrimaryCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNlosingFocusCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmarginHeight	**XmCMarginHeight**	**Dimension**	5	CSG
XmNmarginWidth	**XmCMarginWidth**	**DimensiOBon**	5	CSG
XmNmaxLength	**XmCMaxLength**	**int**	largest integer	CSG
XmNmodifyVerifyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmotionVerifyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
Headers/Xm.incource	**XmCSource**	**XmTextSource**	Default source	CSG
XmNtopCharacter	**XmCTextPosition**	**XmTextPosition**	0	CSG
XmNvalue	**XmCValue**	**String**	""	CSG
XmNvalueChangedCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNverifyBell	**XmCVerifyBell**	**Boolean**	dynamic	CSG

XmNactivateCallback

Specifies the list of callbacks called when the user invokes an event that calls the *Activate*( ) function. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is XmCR_ACTIVATE.

XmNautoShowCursorPosition

Ensures that the visible text contains the insert cursor when set to True. If the insert cursor changes, the contents of Text may scroll to bring the insertion point into the window.

XmNcursorPosition

Indicates the position in the text where the current insert cursor is to be located. Position is determined by the number of characters from the beginning of the text. The first character position is 0 (zero).

XmNeditable

When set to True, indicates that the user can edit the text string. Prohibits the user from editing the text when set to False.

XmNeditMode

Specifies the set of keyboard bindings used in Text. The default, XmSINGLE_LINE_EDIT, provides the set of key bindings to be used in editing single-line text. XmMULTI_LINE_EDIT provides the set of key bindings to be used in editing multiline text.

The results of placing a Text widget inside a ScrolledWindow when the Text's **XmNeditMode** is XmSINGLE_LINE_EDIT are undefined.

XmNfocusCallback

Specifies the list of callbacks called when Text accepts input focus. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is XmCR_FOCUS.

XmNgainPrimaryCallback

Specifies the list of callbacks called when an event causes the Text widget to gain ownership of the primary selection. The reason sent by the callback is XmCR_GAIN_PRIMARY.

XmNlosePrimaryCallback
Specifies the list of callbacks called when an event causes the Text widget to lose ownership of the primary selection. The reason sent by the callback is XmCR_LOSE_PRIMARY.

XmNlosingFocusCallback
Specifies the list of callbacks called before Text loses input focus. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is XmCR_LOSING_FOCUS.

XmNmarginHeight
Specifies the distance between the top edge of the widget window and the text, and between the bottom edge of the widget window and the text.

XmNmarginWidth
Specifies the distance between the left edge of the widget window and the text, and between the right edge of the widget window and the text.

XmNmaxLength
Specifies the maximum length of the text string that can be entered into text from the keyboard. This value must be non-negative. Strings that are entered using the **XmNvalue** resource or the *XmTextSetString*( ) function ignore this resource.

XmNmodifyVerifyCallback
Specifies the list of callbacks called before text is deleted from or inserted into Text. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is XmCR_MODIFYING_TEXT_VALUE. When multiple Text widgets share the same source, only the widget that initiates the source change generates the **XmNmodifyVerifyCallback**.

XmNmotionVerifyCallback
Specifies the list of callbacks called before the insert cursor is moved to a new position. The type of the structure whose address is passed to this callback is **XmTextVerifyCallbackStruct**. The reason sent by the callback is XmCR_MOVING_INSERT_CURSOR. It is possible for more than one **XmNmotionVerifyCallback** to be generated from a single action.

XmNsource
Specifies the source with which the widget displays text. If no source is specified, the widget creates a default string source. This resource can be used to share text sources between Text widgets.

XmNtopCharacter
Displays the position of text at the top of the window. Position is determined by the number of characters from the beginning of the text. The first character position is 0 (zero).

If the **XmNeditMode** is XmMULTI_LINE_EDIT, the line of text that contains the top character is displayed at the top of the widget without shifting the text left or right. *XtGetValues*( ) for **XmNtopCharacter** returns the position of the first character in the line that is displayed at the top of the widget.

XmNvalue
Displays the string value. *XtGetValues*( ) returns 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 generates 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.

XmNverifyBell

Specifies whether the bell should sound when the verification returns without continuing the action.

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 type indicating possible actions. Each mouse click performed within half a second of the previous mouse click increments the index into this array and performs the defined action for that index. The possible actions in the order they occur in the default array are

XmSELECT_POSITION

Resets the insert cursor position.

XmSELECT_WORD

Selects a word.

XmSELECT_LINE

Selects a line of text.

XmSELECT_ALL

Selects all of the text.

XmNselectionArrayCount

Indicates the number of elements in the **XmNselectionArray** resource. The value must not be negative.

XmNselectThreshold

Specifies the number of pixels of motion that is required to select the next character when selection is performed using the click-drag mode of selection. The value must not be negative.

XmText **Output Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNblinkRate	**XmCBlinkRate**	**int**	500	CSG
XmNcolumns	**XmCColumns**	**short**	dynamic	CSG
XmNcursorPositionVisible	**XmCCursorPositionVisible**	**Boolean**	True	CSG
XmNfontList	**XmCFontList**	**XmFontList**	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 measured in character spaces. The value must be greater than 0 (zero). The default value depends on the value of the **XmNwidth** resource. If no width is specified the default is 20.

XmNcursorPositionVisible

Indicates that the insert cursor position is marked by a blinking text cursor when the **Boolean** value is True.

XmNfontList

Specifies the font list to be used for Text. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is subclass of the *XmBulletinBoard* or *VendorShell* widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent.

Text searches the font list for the first occurrence of a font set that has an XmFONTLIST_DEFAULT_TAG. If a default element is not found, the first font set in the font list is used. If the list contains no font sets, the first font in the font list is used. Refer to Section 4.2 on page 60 for more information on a font list structure.

XmNresizeHeight

Indicates that Text attempts 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 widget whose parent is a ScrolledWindow, and when **XmNscrollVertical** is True.

XmNresizeWidth

Indicates that Text attempts to resize its width to accommodate all the text contained in the widget when the **Boolean** value is True. This attribute is ignored if **XmNwordWrap** is True.

XmNrows

Specifies the initial height of the text window measured in character heights. This attribute is ignored if the text widget resource **XmNeditMode** is XmSINGLE_LINE_EDIT. The value must be greater than 0 (zero). The default value depends on the value of the **XmNheight** resource. If no height is specified the default is 1.

XmNwordWrap

Indicates that lines are to be broken at word breaks (that is, the text does not go off the right edge of the window) when the **Boolean** value is True. Words are defined as a sequence of characters separated by white space. White space is defined as a space, tab or newline. This attribute is ignored if the text widget resource **XmNeditMode** is XmSINGLE_LINE_EDIT.

The following resources are used only when text is created in a ScrolledWindow. See the reference page for *XmCreateScrolledText*( ).

XmText **Scrolling Resource Set**

Name	Class	Type	Default	Access
XmNscrollHorizontal	**XmCScroll**	**Boolean**	True	CG
XmNscrollLeftSide	**XmCScrollSide**	**Boolean**	dynamic	CG
XmNscrollTopSide	**XmCScrollSide**	**Boolean**	False	CG
XmNscrollVertical	**XmCScroll**	**Boolean**	True	CG

XmNscrollHorizontal

Adds a ScrollBar that allows the user to scroll horizontally through text when the Boolean value is True. This resource is forced to False when the Text widget is placed in a ScrolledWindow with **XmNscrollingPolicy** set to XmAUTOMATIC.

XmNscrollLeftSide

Indicates that the vertical ScrollBar should be placed on the left side of the scrolled text window when the Boolean value is True. This attribute is ignored if **XmNscrollVertical** is False or the Text resource **XmNeditMode** is XmSINGLE_LINE_EDIT. The default value may depend on the value of the **XmNstringDirection** resource.

XmNscrollTopSide

Indicates that the horizontal ScrollBar should be placed on the top side of the scrolled text window when the Boolean value is True.

XmNscrollVertical

Adds a ScrollBar that allows the user to scroll vertically through text when the Boolean value is True. This attribute is ignored if the Text resource **XmNeditMode** is XmSINGLE_LINE_EDIT. This resource is forced to False when the Text widget is placed in a ScrolledWindow with **XmNscrollingPolicy** set to XmAUTOMATIC.

Inherited Resources

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

XmPrimitive **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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 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; these changes are 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.

doit Indicates whether the action that invoked the callback is performed. Setting **doit** to False negates the action.

currInsert Indicates the current position of the insert cursor.

newInsert Indicates the position at which the user attempts to position the insert cursor.

startPos Indicates the starting position of the text to modify. If the callback is not a modify verification callback, this value is the same as **currInsert**.

endPos Indicates the ending position of the text to modify. If no text is replaced or deleted, the value is the same as **startPos**. If the callback is not a modify verification callback, this value is the same as **currInsert**.

text Points to a structure of type **XmTextBlockRec**. This structure holds the textual information to be inserted.

```
typedef struct
{
      char                      *ptr;
      int                        length;
      XmTextFormat               format;
} XmTextBlockRec, *XmTextBlock;
```

ptr Points to the text to be inserted.

length Specifies the length of the text to be inserted.

format Specifies the format of the text, either XmFMT_8_BIT or XmFMT_16_BIT.

The following table lists the reasons for the individual verification callback structure fields being valid.

Reason	**Valid Fields**
XmCR_LOSING_FOCUS	**reason, event, doit, currInsert, newInsert, startPos, endPos**
XmCR_MODIFYING_TEXT_VALUE	**reason, event, doit, currInsert, newInsert, startPos, endPos, text**
XmCR_MOVING_INSERT_CURSOR	**reason, event, doit, currInsert, newInsert**

Action Routines

The *XmText* action routines are:

activate( )

Calls the callbacks for **XmNactivateCallback**. If the parent is a manager, passes the event to the parent.

backward-character( )

Moves the insertion cursor one character to the left. This action may have different behavior in a right-to-left language environment.

backward-paragraph(extend)

If **XmNeditMode** is XmMULTI_LINE_EDIT and this action is called with no argument, moves the insertion cursor to the first character that is not white space 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.

backward-word(extend)

If this action is called with no argument, moves the insertion cursor to the first character that is not white space after the first white-space 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.

beep( )

Causes the terminal to beep.

beginning-of-file(extend)

If this action is called with no argument, moves the insertion cursor to the beginning of the text.

If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

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.

clear-selection( )
Clears the current selection by replacing each character except with a space character.

copy-clipboard( )
Copies the current selection to the clipboard.

copy-primary( )
Copies the primary selection to just before the insertion cursor.

copy-to( )
If a secondary selection exists, copies the secondary selection to just before the insertion cursor. If no secondary selection exists, copies the primary selection to the pointer location.

cut-clipboard( )
Cuts the current selection to the clipboard.

cut-primary( )
Cuts the primary selection and pastes it to just before the insertion cursor.

delete-next-character( )
In normal mode, if there is a non-null selection, deletes the selection; otherwise, deletes the character following the insertion cursor. In add mode, if there is a non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character following the insertion cursor. This may impact the selection.

delete-next-word( )
In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next space, tab or end-of-line character. This may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-previous-character( )
In normal mode, if there is a non-null selection, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. In add mode, if there is a non-null selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. This may impact the selection.

delete-previous-word( )
In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the next space, tab or beginning-of-line character. This may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-selection( )
: Deletes the current selection.

delete-to-end-of-line( )
: In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next end of line character. This may impact the selection.

delete-to-start-of-line( )
: In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the previous beginning-of-line character. This may impact the selection.

deselect-all( )
: Deselects the current selection.

do-quick-action( )
: Marks the end of a secondary selection. Performs the quick action initiated by the *quick-copy-set*( ) or *quick-cut-set*( ) action.

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.

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.

extend-adjust( )
: Selects text from the anchor to the pointer position and deselects text outside that range. Moving the pointer over several lines selects text from the anchor to the end of each line the pointer moves over and up to the pointer position on the current line.

extend-end( )
: Moves the insertion cursor to the position of the pointer.

extend-start( )
: Adjusts the anchor using the balance-beam method. Selects text from the anchor to the pointer position and deselects text outside that range.

forward-character( )
: Moves the insertion cursor one character to the right. This action may have different behavior in a right-to-left language environment.

forward-paragraph(extend)
: If **XmNeditMode** is XmMULTI_LINE_EDIT, and this action is called with no argument, moves the insertion cursor to the first character that is not white space 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.

forward-word(extend)

If this action is called with no argument, moves the insertion cursor to the first white-space character or end-of-line following the next character that is not white space. 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.

grab-focus( )

This key binding performs the action defined in the **XmNselectionArray**, depending on the number of multiple mouse clicks. The default selection array ordering is one click to move the insertion cursor to the pointer position, two clicks to select a word, three clicks to select a line of text and four clicks to select all text. A single click also deselects any selected text and sets the anchor at the pointer position. This action may have different behavior in a locale other than the C locale.

Help( )

Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

insert-string(*string*)

If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts *string* before the insertion cursor.

key-select(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.

kill-next-character( )

In normal mode, if there is a non-null selection, deletes the selection; otherwise, kills the character following the insertion cursor and stores the character in the cut buffer. In add mode, if there is a non-null selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the character following the insertion cursor and stores the character in the cut buffer. This may impact the selection.

kill-next-word( )

In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters following the insertion cursor to the next space, tab or end-of-line character, and stores the characters in the cut buffer. This may impact the selection. This action may have different behavior in a locale other than the C locale.

kill-previous-character( )

In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the character immediately preceding the insertion cursor and stores the character in the cut buffer. This may impact the selection.

kill-previous-word( )

In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters preceding the insertion cursor up to the next space, tab or beginning-of-line character, and stores the characters in the cut buffer. This may impact the selection. This action may have different behavior in a locale other than the C locale.

kill-selection( )

Kills the currently selected text and stores the text in the cut buffer.

kill-to-end-of-line( )

In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters following the insertion cursor to the next end of line character and stores the characters in the cut buffer. This may impact the selection.

kill-to-start-of-line( )

In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, kills the characters preceding the insertion cursor to the next beginning-of-line character and stores the characters in the cut buffer. This may impact the selection.

move-destination( )

Moves the insertion cursor to the pointer position without changing any existing current selection. If there is no current selection, sets the widget as the destination widget.

move-to( )

If a secondary selection exists, cuts the secondary selection to the insertion cursor. If no secondary selection exists, cuts the primary selection to the pointer location.

newline( )

If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts a newline before the insertion cursor.

newline-and-backup( )

If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts a newline just before the insertion cursor and repositions the insertion cursor to the end of the line before the newline.

newline-and-indent( )

If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts a newline and then the same number of white-space characters as at the beginning of the previous line.

next-line( )

Moves the insertion cursor to the next line.

next-page(extend)

If this action is called with no argument, moves the insertion cursor forward one page.

If 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.

next-tab-group( )

Traverses to the next tab group.

page-left( )

Scrolls the viewing window left one page of text.

page-right( )

Scrolls the viewing window right one page of text.

paste-clipboard( )

Pastes the contents of the clipboard before the insertion cursor.

prev-tab-group( )

Traverses to the previous tab group.

process-cancel( )

Cancels the current *extend-adjust*( ) or *secondary-adjust*( ) operation and leaves the selection state as it was before the operation; otherwise, and if the parent is a manager, passes the event to the parent.

process-down( )

If **XmNeditMode** is XmSINGLE_LINE_EDIT, and **XmNnavigationType** is XmNONE, traverses to the widget below the current one in the tab group.

If **XmNeditMode** is XmMULTI_LINE_EDIT, moves the insertion cursor down one line.

process-home( )

Moves the insertion cursor to the beginning of the line.

process-return( )

If **XmNeditMode** is XmSINGLE_LINE_EDIT, calls the callbacks for **XmNactivateCallback**, and if the parent is a manager, passes the event to the parent. If **XmNeditMode** is XmMULTI_LINE_EDIT, inserts a newline.

process-shift-down( )

If **XmNeditMode** is XmMULTI_LINE_EDIT, moves the insertion cursor down one line.

process-shift-up( )

If **XmNeditMode** is XmMULTI_LINE_EDIT, moves the insertion cursor up one line.

process-tab( )

If **XmNeditMode** is XmSINGLE_LINE_EDIT, traverses to the next tab group. If **XmNeditMode** is XmMULTI_LINE_EDIT, inserts a tab.

process-up( )

If **XmNeditMode** is XmSINGLE_LINE_EDIT and **XmNnavigationType** is **XmNONE**, traverses to the widget above the current one in the tab group.

If **XmNeditMode** is XmMULTI_LINE_EDIT, moves the insertion cursor up one line.

quick-copy-set( )

Marks the beginning of a secondary selection for use in quick copy.

quick-cut-set( )

Marks the beginning of a secondary selection for use in quick cut.

redraw-display( )
Redraws the contents of the text window.

scroll-one-line-down( )
Scrolls the text area down one line.

scroll-one-line-up( )
Scrolls the text area up one line.

secondary-adjust( )
Extends the secondary selection to the pointer position.

secondary-notify( )
Copies the secondary selection to the insertion cursor of the destination widget.

secondary-start( )
Marks the beginning of a secondary selection.

select-adjust( )
Extends the current selection. The amount of text selected depends on the number of mouse clicks, as specified by the **XmNselectionArray** resource.

select-all( )
Selects all text.

select-end( )
Extends the current selection. The amount of text selected depends on the number of mouse clicks, as specified by the **XmNselectionArray** resource.

select-start( )
Marks the beginning of a new selection region.

self-insert( )
If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts the character associated with the key pressed at the insertion cursor.

set-anchor( )
Resets the anchor point for extended selections. Resets the destination of secondary selection actions.

set-insertion-point( )
Sets the insertion position.

set-selection-hint( )
Sets the text source and location of the current selection.

toggle-add-mode( )
Toggles the state of Add Mode.

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.

SEE ALSO

Section 4.2 on page 60, Section 4.6 on page 62, *Core*, *XmCreateScrolledText*( ), *XmCreateText*( ), *XmFontListAppendEntry*( ), *XmPrimitive*, *XmTextClearSelection*( ), *XmTextCopy*( ), *XmTextCut*( ), *XmTextField*, *XmTextGetBaseline*( ), *XmTextGetEditable*( ), *XmTextGetInsertionPosition*( ), *XmTextGetLastPosition*( ), *XmTextGetMaxLength*( ), *XmTextGetSelection*( ), *XmTextGetSelectionPosition*( ), *XmTextGetSource*( ), *XmTextGetString*( ), *XmTextGetTopCharacter*( ), *XmTextInsert*( ), *XmTextPaste*( ), *XmTextPosToXY*( ), *XmTextRemove*( ), *XmTextReplace*( ), *XmTextScroll*( ), *XmTextSetAddMode*( ), *XmTextSetEditable*( ), *XmTextSetHighlight*( ), *XmTextSetInsertionPosition*( ), *XmTextSetMaxLength*( ), *XmTextSetSelection*( ), *XmTextSetSource*( ), *XmTextSetString*( ), *XmTextSetTopCharacter*( ), *XmTextShowPosition*( ) and *XmTextXYToPos*( ).

NAME

XmTextClearSelection — a Text function that clears the primary selection

SYNOPSIS

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

void XmTextClearSelection(
      Widget                    widget,
      Time                      time);
```

DESCRIPTION

XmTextClearSelection( ) clears the primary selection in the Text widget.

widget Specifies the Text widget ID.

time Specifies the server time at which the selection value is desired. This should be the time of the event that triggered this request.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextCopy — a Text function that copies the primary selection to the clipboard

SYNOPSIS

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

Boolean XmTextCopy(
      Widget                      widget,
      Time                        time);
```

DESCRIPTION

XmTextCopy( ) copies the primary selected text to the clipboard.

widget Specifies the Text widget ID.

time Specifies the server time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

SEE ALSO

XmText.

NAME

XmTextCut — a Text function that copies the primary selection to the clipboard and deletes the selected text

SYNOPSIS

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

Boolean XmTextCut(
      Widget                    widget,
      Time                      time);
```

DESCRIPTION

XmTextCut( ) copies the primary selected text to the clipboard and then deletes the primary selected text. This routine also calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks.

widget Specifies the Text widget ID.

time Specifies the server time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection, or if no data is placed on the clipboard. Otherwise, it returns True.

SEE ALSO

XmText.

NAME

XmTextDisableRedisplay — a Text function that temporarily prevents visual update of the Text widget

SYNOPSIS

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

void XmTextDisableRedisplay(
      Widget                    widget);
```

DESCRIPTION

XmTextDisableRedisplay( ) prevents redisplay of the specified Text widget even though its visual attributes have been modified. The visual appearance of the widget remains unchanged until *XmTextEnableRedisplay*( ) is called. This allows an application to make multiple changes to the widget without causing intermediate visual updates.

widget Specifies the Text widget ID.

SEE ALSO

XmText and XmTextEnableRedisplay( ).

NAME

XmTextEnableRedisplay — a Text function that forces the visual update of a Text widget

SYNOPSIS

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

void XmTextEnableRedisplay(
      Widget                    widget);
```

DESCRIPTION

XmTextEnableRedisplay( ) is used in conjunction with *XmTextDisableRedisplay*( ), which suppresses visual update of the Text widget. When *XmTextEnableRedisplay*( ) is called, it determines if any visual attributes have been set or modified for the specified widget since *XmTextDisableRedisplay*( ) was called. If so, it forces the widget to update its visual display for all of the intervening changes. Any subsequent changes that affect visual appearance cause the widget to update its visual display.

widget Specifies the Text widget ID

SEE ALSO

XmText and *XmTextDisableRedisplay*( ).

NAME

XmTextField — the TextField class

SYNOPSIS

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

DESCRIPTION

The TextField widget provides a single line text editor for customizing both user and programmatic interfaces. It is used for single-line string entry, and forms entry with verification procedures. It provides an application with a consistent editing system for textual data.

TextField provides separate callback lists to verify movement of the insert cursor, modification of the text, and changes in input focus. Each of these callbacks provides the verification function with the widget instance, the event that caused the callback, and a data structure specific to the verification type. From this information, the function can verify if the application considers this to be a legitimate state change and can signal the widget whether to continue with the action.

The user interface tailors a new set of actions. The key bindings have been added for insert cursor movement, deletion, insertion and selection of text.

TextField allows the user to select regions of text. Selection is based on the model specified in the **ICCCM** specification. TextField supports primary and secondary selection.

Classes

TextField widget inherits behavior and resources from *Core* and *XmPrimitive*.

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 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).

XmTextField **Resource 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
XmNcursorPositionVisible	**XmCCursorPositionVisible**	**Boolean**	True	CSG
XmNeditable	**XmCEditable**	**Boolean**	True	CSG
XmNfocusCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNfontList	**XmCFontList**	**XmFontList**	dynamic	CSG
XmNgainPrimaryCallback	**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
XmNmodifyVerifyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNmotionVerifyCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNpendingDelete	**XmCPendingDelete**	**Boolean**	True	CSG
XmNresizeWidth	**XmCResizeWidth**	**Boolean**	False	CSG
XmNselectionArray	**XmCSelectionArray**	**XtPointer**	default array	CSG
XmNselectionArrayCount	**XmCSelectionArrayCount**	**int**	3	CSG
XmNselectThreshold	**XmCSelectThreshold**	**int**	5	CSG
XmNvalue	**XmCValue**	**String**	""	CSG
XmNvalueChangedCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNverifyBell	**XmCVerifyBell**	**Boolean**	dynamic	CSG

XmNactivateCallback

Specifies the list of callbacks called when the user invokes an event that calls the *Activate*( ) function. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is XmCR_ACTIVATE.

XmNblinkRate

Specifies the blink rate of the text cursor in milliseconds. The time indicated in the blink rate relates to the length of time the cursor is visible and the time the cursor is invisible (that is, the time it takes to blink the insertion cursor on and off is twice the blink rate). The cursor does not blink when the blink rate is set to 0 (zero). The value must not be negative.

XmNcolumns

Specifies the initial width of the text window as an integer number of characters. The width equals the number of characters specified by this resource multiplied by the maximum character width of the associated font. For proportionate fonts, the actual number of characters that fit on a given line may be greater than the value specified. The value must be greater than 0 (zero). The default value depends on the value of the **XmNwidth** resource. If no width is specified the default is 20.

XmNcursorPosition

Indicates the position in the text where the current insert cursor is to be located. Position is determined by the number of characters from the beginning of the text.

XmNcursorPositionVisible

Indicates that the insert cursor position is marked by a blinking text cursor when the **Boolean** is True.

XmNeditable

When set to True, indicates that the user can edit the text string. A false value prohibits the user from editing the text.

XmNfocusCallback

Specifies the list of callbacks called when TextField accepts input focus. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is XmCR_FOCUS.

XmNfontList

Specifies the font list to be used for TextField. If this value is NULL at initialization, the parent hierarchy of the widget is searched for an ancestor that is a subclass of the BulletinBoard or VendorShell widget class. If such an ancestor is found, the font list is initialized to the **XmNtextFontList** of the ancestor widget. If no such ancestor is found, the default is implementation dependent. Refer to Section 4.2 on page 60 for more information on a font list structure.

TextField searches the font list for the first occurrence of a font set that has an XmFONTLIST_DEFAULT_TAG. If a default element is not found, the first font set in the font list is used. If the list contains no font sets, the first font in the font list is used.

XmNgainPrimaryCallback

Specifies the list of callbacks 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 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 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 non-negative. Strings that are entered using the **XmNvalue** resource or the *XmTextFieldSetString*( ) function ignore this resource.

XmNmodifyVerifyCallback

Specifies the list of callbacks 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 generates the **XmNmodifyVerifyCallback**.

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**s to be generated from a single action.

XmNpendingDelete

Indicates that pending delete mode is on when the **Boolean** is True. Pending deletion is defined as deletion of the selected text when an insertion is made.

XmNresizeWidth

Indicates that the TextField widget attempts to resize its width to accommodate all the text contained in the widget when Boolean is True.

XmNselectionArray

Defines the actions for multiple mouse clicks. Each mouse click performed within a half of a second of the previous mouse click increments the index into this array and perform the defined action for that index. The possible actions are

XmSELECT_POSITION

Resets the insert cursor position.

XmSELECT_WORD

Selects a word.

XmSELECT_LINE

Selects a line of text.

XmNselectionArrayCount

Specifies the number of actions defined in the **XmNselectionArray** resource. The value must not be negative.

XmNselectThreshold

Specifies the number of pixels of motion that is required to select the next character when selection is performed using the click-drag mode of selection. The value must not be negative.

XmNvalue

Displays the string value. *XtGetValues*( ) returns 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 TextField. The type of the structure whose address is passed to this callback is **XmAnyCallbackStruct**. The reason sent by the callback is XmCR_VALUE_CHANGED. When multiple TextField widgets share the same source, only the widget that initiates the source change generates the **XmNvalueChangedCallback**. This callback represents a change in the source in the TextField, not in the TextField widget. The **XmNvalueChangedCallback** should occur only in pairs with a **XmNmodifyVerifyCallback**, assuming that the **doit** flag in the callback structure of the **XmNmodifyVerifyCallback** is not set to False.

XmNverifyBell

Specifies whether a bell sounds when an action is reversed during a verification callback.

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**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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 of **startPos**, **endPos** and **text** in the callback structure **XmTextVerifyCallbackStruct** may be modified upon receiving the callback, and these changes are 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 is performed. Setting **doit** to False negates the action.

currInsert Indicates the current position of the insert cursor.

newInsert Indicates the position at which the user attempts to position the insert cursor.

startPos Indicates the starting position of the text to modify. If the callback is not a modify verification callback, this value is the same as **currInsert**.

endPos Indicates the ending position of the text to modify. If no text is replaced or deleted, the value is the same as **startPos**. If the callback is not a modify verification callback, this value is the same as **currInsert**.

text Points to the following structure of type **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.

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

Reason	**Valid Fields**
XmCR_LOSING_FOCUS	**reason, event, doit**
XmCR_MODIFYING_TEXT_VALUE	**reason, event, doit, currInsert, newInsert, startPos, endPos, text**
XmCR_MOVING_INSERT_CURSOR	**reason, event, doit, currInsert, newInsert**

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( )

Moves the insertion cursor one character to the left. This action may have different behavior in a right-to-left language environment.

backward-word(extend)

If this action is called with no argument, moves the insertion cursor to the first character that is not white space after the first white-space 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.

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.

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.

clear-selection( )
Clears the current selection by replacing each character except with a space character.

copy-clipboard( )
Copies the current selection to the clipboard.

copy-primary( )
Copies the primary selection to just before the insertion cursor.

copy-to( )
If a secondary selection exists, copies the secondary selection to just before the insertion cursor. If no secondary selection exists, copies the primary selection to the pointer location.

cut-clipboard( )
Cuts the current selection to the clipboard.

cut-primary( )
Cuts the primary selection and pastes it to just before the insertion cursor.

delete-next-character( )
In normal mode, if there is a non-null selection, deletes the selection; otherwise, deletes the character following the insertion cursor. In add mode, if there is a non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character following the insertion cursor. This may impact the selection.

delete-next-word( )
In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next space, tab or end-of-line character. This may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-previous-character( )
In normal mode, if there is a non-null selection, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. In add mode, if there is a non-null selection, the cursor is not disjoint from the selection and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the character of text immediately preceding the insertion cursor. This may impact the selection.

delete-previous-word( )
In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the next space, tab or beginning-of-line character. This may impact the selection. This action may have different behavior in a locale other than the C locale.

delete-selection( )

Deletes the current selection.

delete-to-end-of-line( )

In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters following the insertion cursor to the next end of line character. This may impact the selection.

delete-to-start-of-line( )

In normal mode, if there is a non-null 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 non-null selection, the cursor is not disjoint from the selection, and **XmNpendingDelete** is set to True, deletes the selection; otherwise, deletes the characters preceding the insertion cursor to the previous beginning-of-line character. This may impact the selection.

deselect-all( )

Deselects the current selection.

end-of-line(*extend*)

If this action is called with no argument, moves the insertion cursor to the end of the line. If called with an argument of *extend*, moves the insertion cursor as in the case of no argument and extends the current selection.

extend-adjust( )

Selects text from the anchor to the pointer position and deselects text outside that range. Moving the pointer over several lines selects text from the anchor to the end of each line the pointer moves over and up to the pointer position on the current line.

extend-end( )

Moves the insertion cursor to the position of the pointer.

extend-start( )

Adjusts the anchor using the balance-beam method. Selects text from the anchor to the pointer position and deselects text outside that range.

forward-character( )

Moves the insertion cursor one character to the right. This action may have different behavior in a right-to-left language environment.

forward-word(extend)

If this action is called with no argument, moves the insertion cursor to the first white-space character or end-of-line following the next character that is not white space. 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.

grab-focus( )

This key binding performs the action defined in the **XmNselectionArray**, depending on the number of multiple mouse clicks. The default selection array ordering is one click to move the insertion cursor to the pointer position, two clicks to select a word, three clicks to select a line of text and four clicks to select all text. A single click also deselects any selected text and sets the anchor at the pointer position. This action may have different behavior in a locale other than the C locale.

Help( )

Calls the callbacks for **XmNhelpCallback** if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest ancestor that has them.

insert-string(*string*)

If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts *string* before the insertion cursor.

key-select(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.

move-destination( )

Moves the insertion cursor to the pointer position without changing any existing current selection. If there is no current selection, sets the widget as the destination widget.

move-to( )

If a secondary selection exists, cuts the secondary selection to the insertion cursor. If no secondary selection exists, cuts the primary selection to the pointer location.

next-line( )

Moves the insertion cursor to the next line.

next-page(extend)

If this action is called with no argument, moves the insertion cursor forward one page.

If 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.

next-tab-group( )

Traverses to the next tab group.

page-left( )

Scrolls the viewing window left one page of text.

page-right( )

Scrolls the viewing window right one page of text.

paste-clipboard( )

Pastes the contents of the clipboard before the insertion cursor.

prev-tab-group( )

Traverses to the previous tab group.

process-cancel( )

Cancels the current *extend-adjust*( ) or *secondary-adjust*( ) 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.

secondary-adjust( )
Extends the secondary selection to the pointer position.

secondary-start( )
Marks the beginning of a secondary selection.

select-all( )
Selects all text.

self-insert( )
If **XmNpendingDelete** is True and the cursor is not disjoint from the current selection, deletes the entire selection. Inserts the character associated with the key pressed at the insertion cursor.

set-anchor( )
Resets the anchor point for extended selections. Resets the destination of secondary selection actions.

toggle-add-mode( )
Toggles the state of Add Mode.

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.

SEE ALSO

Section 4.2 on page 60, *Core*, *XmCreateTextField*( ), *XmFontListAppendEntry*( ), *XmPrimitive*, *XmTextFieldClearSelection*( ), *XmTextFieldCopy*( ), *XmTextFieldCut*( ), *XmTextFieldGetBaseline*( ), *XmTextFieldGetEditable*( ), *XmTextFieldGetInsertionPosition*( ), *XmTextFieldGetLastPosition*( ), *XmTextFieldGetMaxLength*( ), *XmTextFieldGetSelection*( ), *XmTextFieldGetSelectionPosition*( ), *XmTextFieldGetString*( ), *XmTextFieldGetSubstring*( ), *XmTextFieldInsert*( ), *XmTextFieldPaste*( ), *XmTextFieldPosToXY*( ), *XmTextFieldRemove*( ), *XmTextFieldReplace*( ), *XmTextFieldSetAddMode*( ), *XmTextFieldSetEditable*( ), *XmTextFieldSetHighlight*( ), *XmTextFieldSetInsertionPosition*( ), *XmTextFieldSetMaxLength*( ), *XmTextFieldSetSelection*( ), *XmTextFieldSetString*( ), *XmTextFieldShowPosition*( ) and *XmTextFieldXYToPos*( ).

NAME

XmTextFieldClearSelection — a TextField function that clears the primary selection

SYNOPSIS

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

void XmTextFieldClearSelection(
      Widget                     widget,
      Time                       time);
```

DESCRIPTION

XmTextFieldClearSelection( ) clears the primary selection in the TextField widget.

widget Specifies the TextField widget ID.

time Specifies the time at which the selection value is desired. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldCopy — a TextField function that copies the primary selection to the clipboard

SYNOPSIS

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

Boolean XmTextFieldCopy(
      Widget                     widget,
      Time                       time);
```

DESCRIPTION

XmTextFieldCopy( ) copies the primary selected text to the clipboard.

widget Specifies the TextField widget ID.

time Specifies the time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection or if no data is placed on the clipboard. Otherwise, it returns True.

SEE ALSO

XmTextField.

NAME

XmTextFieldCut — a TextField function that copies the primary selection to the clipboard and deletes the selected text

SYNOPSIS

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

Boolean XmTextFieldCut(
      Widget                    widget,
      Time                      time);
```

DESCRIPTION

XmTextFieldCut( ) copies the primary selected text to the clipboard and then deletes the primary selected text. This routine also calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks.

widget Specifies the TextField widget ID.

time Specifies the time at which the selection value is to be modified. This should be the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

This function returns False if the primary selection is NULL, if the *widget* does not own the primary selection, if the function is unable to gain ownership of the clipboard selection or if no data is placed on the clipboard. Otherwise, it returns True.

SEE ALSO

XmTextField.

NAME

XmTextFieldGetBaseline — a TextField function that accesses the x position of the first baseline

SYNOPSIS

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

int XmTextFieldGetBaseline(
      Widget                    widget);
```

DESCRIPTION

XmTextFieldGetBaseline( ) accesses the x position of the first baseline in the TextField widget, relative to the x position of the top of the widget.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns an integer value that indicates the x position of the first baseline in the TextField widget. The calculation takes into account the margin height, shadow thickness, highlight thickness and ascent of the first font in the fontlist. In this calculation, the x position of the top of the widget is 0 (zero).

SEE ALSO

XmTextField.

NAME

XmTextFieldGetEditable — a TextField function that accesses the edit permission state

SYNOPSIS

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

Boolean XmTextFieldGetEditable(
      Widget                    widget);
```

DESCRIPTION

XmTextFieldGetEditable( ) accesses the edit permission state of the TextField widget.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns a **Boolean** type that indicates the state of the **XmNeditable** resource.

SEE ALSO

XmTextField.

NAME

XmTextFieldGetInsertionPosition — a TextField function that accesses the position of the insertion cursor

SYNOPSIS

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

XmTextPosition XmTextFieldGetInsertionPosition(
      Widget                    widget);
```

DESCRIPTION

XmTextFieldGetInsertionPosition( ) accesses the insertion cursor position of the TextField widget.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns an **XmTextPosition** value that indicates the state of the **XmNcursorPosition** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

SEE ALSO

XmTextField.

NAME

XmTextFieldGetLastPosition — a TextField function that accesses the position of the last text character

SYNOPSIS

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

XmTextPosition XmTextFieldGetLastPosition(
      Widget                    widget);
```

DESCRIPTION

XmTextFieldGetLastPosition( ) accesses the position of the last character in the text buffer of the TextField widget.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns an **XmTextPosition** value that indicates the position of the last character in the text buffer. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero). The last character position is equal to the number of characters in the text buffer.

SEE ALSO

XmTextField.

NAME

XmTextFieldGetMaxLength — a TextField function that accesses the value of the current maximum allowable length of a text string entered from the keyboard

SYNOPSIS

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

int XmTextFieldGetMaxLength(
      Widget                    widget);
```

DESCRIPTION

XmTextFieldGetMaxLength( ) accesses the value of the current maximum allowable length of the text string in the TextField widget entered from the keyboard. The maximum allowable length prevents the user from entering a text string larger than this limit.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns the integer value that indicates the string's maximum allowable length that can be entered from the keyboard.

SEE ALSO

XmTextField.

NAME

XmTextFieldGetSelection — a TextField function that retrieves the value of the primary selection

SYNOPSIS

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

char *XmTextFieldGetSelection(
      Widget                       widget);
```

DESCRIPTION

XmTextFieldGetSelection( ) retrieves the value of the primary selection. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the string by calling *XtFree*( ).

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns a character pointer to the string that is associated with the primary selection.

SEE ALSO

XmTextField.

NAME

XmTextFieldGetSelectionPosition — a TextField function that accesses the position of the primary selection

SYNOPSIS

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

Boolean XmTextFieldGetSelectionPosition(
      Widget                    widget,
      XmTextPosition           *left,
      XmTextPosition           *right);
```

DESCRIPTION

XmTextFieldGetSelectionPosition( ) accesses the left and right position of the primary selection in the text buffer of the TextField widget.

widget Specifies the TextField widget ID.

left Specifies the pointer in which the position of the left boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

right Specifies the pointer in which the position of the right boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

This function returns True if the widget owns the primary selection; otherwise, it returns False.

SEE ALSO

XmTextField.

NAME

XmTextFieldGetString — a TextField function that accesses the string value

SYNOPSIS

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

char *XmTextFieldGetString(
      Widget                      widget);
```

DESCRIPTION

XmTextFieldGetString( ) accesses the string value of the TextField widget. The application is responsible for freeing the storage associated with the string by calling *XtFree*( ).

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns a character pointer to the string value of the TextField widget. This returned value is a copy of th **XmNvalue** resource. Returns an empty string if the length of the TextField widget's string is 0 (zero).

SEE ALSO

XmTextField.

NAME

XmTextFieldGetSubstring — a TextField function that retrieves a copy of a portion of the internal text buffer

SYNOPSIS

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

int XmTextFieldGetSubstring(
      Widget                     widget,
      XmTextPosition             start,
      int                        num_chars,
      int                        buffer_size,
      char                      *buffer);
```

DESCRIPTION

XmTextFieldGetSubstring( ) retrieves a copy of a portion of the internal text buffer of a TextField widget. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

The size of the required buffer depends on the maximum number of bytes per character (MB_CUR_MAX) for the current locale. MB_CUR_MAX is a macro defined in **stdlib.h**. The buffer should be large enough to contain the substring to be copied and a NULL terminator. Use the following equation to calculate the size of buffer the application should provide:

```
buffer_size = (num_chars * MB_CUR_MAX) + 1
```

widget Specifies the TextField widget ID.

start Specifies the beginning character position from which the data is retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

num_chars Specifies the number of characters to be copied into the provided buffer.

buffer_size Specifies the size of the supplied buffer in bytes. This size should account for a NULL terminator.

buffer Specifies the character buffer into which the internal text buffer is copied.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

[XmCOPY_SUCCEEDED]
The function was successful.

[XmCOPY_FAILED]
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

[XmCOPY_TRUNCATED]
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget's buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

SEE ALSO

XmTextField.

NAME

XmTextFieldInsert — a TextField function that inserts a character string into a text string

SYNOPSIS

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

void XmTextFieldInsert(
      Widget                    widget,
      XmTextPosition            position,
      char                     *value);
```

DESCRIPTION

XmTextFieldInsert( ) inserts a character string into the text string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the *position* parameter must be 4.

This routine also calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks.

widget Specifies the TextField widget ID.

position Specifies the position in the text string where the character string is to be inserted.

value Specifies the character string value to be added to the text widget.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldPaste — a TextField function that inserts the clipboard selection

SYNOPSIS

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

Boolean XmTextFieldPaste(
      Widget                    widget);
```

DESCRIPTION

XmTextFieldPaste( ) inserts the clipboard selection at the insertion cursor of the destination widget. If **XmNpendingDelete** is True and the insertion cursor is inside the current selection, the clipboard selection replaces the selected text. This routine calls the widget's **XmNvalueChangedCallback** and **XmNmodifyVerifyCallback**.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

This function returns False if no transfers take place. Otherwise, it returns True.

SEE ALSO

XmTextField.

NAME

XmTextFieldPosToXY — a TextField function that accesses the x and y position of a character position

SYNOPSIS

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

Boolean XmTextFieldPosToXY(
      Widget                     widget,
      XmTextPosition             position,
      Position                  *x,
      Position                  *y);
```

DESCRIPTION

XmTextFieldPosToXY( ) accesses the x and y position, relative to the upper-left corner of the TextField widget, of a given character position in the text buffer.

widget Specifies the TextField widget ID.

position Specifies the character position in the text for which the x and y position is accessed. This is an integer number of characters from the beginning of the buffer. The first character position is 0.

x Specifies the pointer in which the x position, relative to the upper-left corner of the widget, is returned. This value is meaningful only if the function returns True.

y Specifies the pointer in which the y position, relative to the upper left corner of the widget, is returned. This value is meaningful only if the function returns True.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

This function returns True if the character position is displayed in the TextField widget; otherwise, it returns False, and no x or y value is returned.

SEE ALSO

XmTextField.

NAME

XmTextFieldRemove — a TextField function that deletes the primary selection

SYNOPSIS

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

Boolean XmTextFieldRemove(
      Widget                    widget);
```

DESCRIPTION

XmTextFieldRemove( ) deletes the primary selected text. This routine also calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks if there is a selection.

widget Specifies the TextField widget ID.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

This function returns False if the primary selection is NULL or if the *widget* does not own the primary selection. Otherwise, it returns True.

SEE ALSO

XmTextField.

NAME

XmTextFieldReplace — a TextField function that replaces part of a text string

SYNOPSIS

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

void XmTextFieldReplace(
      Widget                    widget,
      XmTextPosition            from_pos,
      XmTextPosition            to_pos,
      char                     *value);
```

DESCRIPTION

XmTextFieldReplace( ) replaces part of the text string in the TextField widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the parameter *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both arguments, *from_pos* and *to_pos*, must be 4.

This routine also calls the widget's callbacks **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback**.

widget Specifies the TextField widget ID.

from_pos Specifies the start position of the text to be replaced.

to_pos Specifies the end position of the text to be replaced.

value Specifies the character string value to be added to the text widget.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldSetAddMode — a TextField function that sets the state of Add mode

SYNOPSIS

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

void XmTextFieldSetAddMode(
      Widget                      widget,
      Boolean                     state);
```

DESCRIPTION

XmTextFieldSetAddMode( ) controls whether or not the TextField widget is in Add mode. When the widget is in Add mode, the insert cursor can be moved without disturbing the primary selection.

widget Specifies the TextField widget ID.

state Specifies whether or not the widget is in Add mode. A value of True turns on Add mode; a value of False turns off Add mode.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldSetEditable — a TextField function that sets the edit permission

SYNOPSIS

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

void XmTextFieldSetEditable(
      Widget                    widget,
      Boolean                   editable);
```

DESCRIPTION

XmTextFieldSetEditable( ) sets the edit permission state of the TextField widget. When set to True, the text string can be edited.

widget Specifies the TextField widget ID.

editable Specifies a Boolean value that when True allows text string edits.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldSetHighlight — a TextField function that highlights text

SYNOPSIS

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

void XmTextFieldSetHighlight(
      Widget                    widget,
      XmTextPosition            left,
      XmTextPosition            right,
      XmHighlightMode           mode);
```

DESCRIPTION

XmTextFieldSetHighlight( ) highlights text between the two specified character positions. The *mode* parameter determines the type of highlighting. Highlighting text merely changes the visual appearance of the text; it does not set the selection.

widget Specifies the TextField widget ID.

left Specifies the position of the left boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

right Specifies the position of the right boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

mode Specifies the type of highlighting to be performed. A value of XmHIGHLIGHT_NORMAL removes highlighting. A value of XmHIGHLIGHT_SELECTED highlights the test using reverse video. A value of XmHIGHLIGHT_SECONDARY_SELECTED highlights the text using underlining.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldSetInsertionPosition — a TextField function that sets the position of the insertion cursor

SYNOPSIS

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

void XmTextFieldSetInsertionPosition(
      Widget                    widget,
      XmTextPosition            position);
```

DESCRIPTION

XmTextFieldSetInsertionPosition( ) sets the insertion cursor position of the TextField widget. This routine also calls the widget's **XmNmotionVerifyCallback** callbacks if the insertion cursor position changes.

widget Specifies the TextField widget ID.

position Specifies the position of the insert cursor. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldSetMaxLength — a TextField function that sets the value of the current maximum allowable length of a text string entered from the keyboard

SYNOPSIS

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

void XmTextFieldSetMaxLength(
      Widget                      widget,
      int                         max_length);
```

DESCRIPTION

XmTextFieldSetMaxLength( ) sets the value of the current maximum allowable length of the text string in the TextField widget. The maximum allowable length prevents the user from entering a text string from the keyboard that is larger than this limit. Strings that are entered using the **XmNvalue** resource or the *XmTextSetString*( ) function ignore this resource.

widget Specifies the TextField widget ID.

max_length Specifies the maximum allowable length of the text string.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmText and *XmTextFieldSetString*( ).

NAME

XmTextFieldSetSelection — a TextField function that sets the primary selection of the text

SYNOPSIS

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

void XmTextFieldSetSelection(
      Widget                     widget,
      XmTextPosition             first,
      XmTextPosition             last,
      Time                       time);
```

DESCRIPTION

XmTextFieldSetSelection( ) sets the primary selection of the text in the widget. It also sets the insertion cursor position to the last position of the selection and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the TextField widget ID.

first Marks the first character position of the text to be selected.

last Marks the last position of the text to be selected.

time Specifies the time at which the selection value is desired. This should be the same as the time of the event that triggered this request.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldSetString — a TextField function that sets the string value

SYNOPSIS

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

void XmTextFieldSetString(
      Widget                     widget,
      char                      *value);
```

DESCRIPTION

XmTextFieldSetString( ) sets the string value of the TextField widget. This routine calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks. It also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the TextField widget ID.

value Specifies the character pointer to the string value and places the string into the text edit window.

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldShowPosition — a TextField function that forces text at a given position to be displayed

SYNOPSIS

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

void XmTextFieldShowPosition(
      Widget                     widget,
      XmTextPosition             position);
```

DESCRIPTION

XmTextFieldShowPosition( ) forces text at the specified position to be displayed. If the **XmNautoShowCursorPosition** resource is True, the application should also set the insert cursor to this position.

widget Specifies the TextField widget ID.

position Specifies the character position to be displayed. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of TextField and its associated resources, see *XmTextField*.

SEE ALSO

XmTextField.

NAME

XmTextFieldXYToPos — a TextField function that accesses the character position nearest an x and y position

SYNOPSIS

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

XmTextPosition XmTextFieldXYToPos(
      Widget                     widget,
      Position                   x,
      Position                   y);
```

DESCRIPTION

XmTextFieldXYToPos( ) accesses the character position nearest to the specified x and y position, relative to the upper-left corner of the TextField widget.

widget Specifies the TextField widget ID.

x Specifies the x position, relative to the upper-left corner of the widget.

y Specifies the y position, relative to the upper-left corner of the widget.

For a complete definition of TextField and its associated resources, see *XmTextField*.

RETURN VALUE

Returns the character position in the text nearest the *x* and *y* position specified. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

SEE ALSO

XmTextField.

NAME

XmTextFindString — a Text function that finds the start position of a text string

SYNOPSIS

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

Boolean XmTextFindString(
      Widget                     widget,
      XmTextPosition             start,
      char                      *string,
      XmTextDirection            direction,
      XmTextPosition            *position);
```

DESCRIPTION

XmTextFindString( ) locates the start position of a specified text string. This routine searches forward or backward for the first occurrence of the string starting from the given start position. If it finds a match, the function returns the position of the first character of the string in *position*.

widget Specifies the Text widget ID.

start Specifies the character position from which the search proceeds. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

string Specifies the search string.

direction Indicates the search direction. It is relative to the primary direction of the text. The possible values are:

XmTEXT_FORWARD
The search proceeds toward the end of the text buffer.

XmTEXT_BACKWARD
The search proceeds toward the beginning of the text buffer.

position Specifies the pointer in which the first character position of the string match is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero). If the function returns False, this value is undefined.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns True if a string match is found; otherwise, returns False.

SEE ALSO

XmText.

NAME

XmTextGetBaseline — a Text function that accesses the x position of the first baseline

SYNOPSIS

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

int XmTextGetBaseline(
      Widget                    widget);
```

DESCRIPTION

XmTextGetBaseline( ) accesses the x position of the first baseline in the Text widget, relative to the x position of the top of the widget.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns an integer value that indicates the x position of the first baseline in the Text widget. The calculation takes into account the margin height, shadow thickness, highlight thickness, and font ascent of the first font in the fontlist. In this calculation the x position of the top of the widget is 0 (zero).

SEE ALSO

XmText.

NAME

XmTextGetEditable — a Text function that accesses the edit permission state

SYNOPSIS

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

Boolean XmTextGetEditable(
      Widget                    widget);
```

DESCRIPTION

XmTextGetEditable( ) accesses the edit permission state of the Text widget.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns a **Boolean** value that indicates the state of the **XmNeditable** resource.

SEE ALSO

XmText.

NAME

XmTextGetInsertionPosition — a Text function that accesses the position of the insert cursor

SYNOPSIS

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

XmTextPosition XmTextGetInsertionPosition(
      Widget                    widget);
```

DESCRIPTION

XmTextGetInsertionPosition( ) accesses the insertion cursor position of the Text widget.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns an **XmTextPosition** value that indicates the state of the **XmNcursorPosition** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

SEE ALSO

XmText.

NAME

XmTextGetLastPosition — a Text function that accesses the last position in the text

SYNOPSIS

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

XmTextPosition XmTextGetLastPosition(
      Widget                    widget);
```

DESCRIPTION

XmTextGetLastPosition( ) accesses the last position in the text buffer of the Text widget. This is an integer number of characters from the beginning of the buffer, and represents the position that text added to the end of the buffer is placed after. The first character position is 0 (zero). The last character position is equal to the number of characters in the text buffer.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns an **XmTextPosition** value that indicates the last position in the text buffer.

SEE ALSO

XmText.

NAME

XmTextGetMaxLength — a Text function that accesses the value of the current maximum allowable length of a text string entered from the keyboard

SYNOPSIS

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

int XmTextGetMaxLength(
      Widget                    widget);
```

DESCRIPTION

XmTextGetMaxLength( ) accesses the value of the current maximum allowable length of the text string in the Text widget entered from the keyboard. The maximum allowable length prevents the user from entering a text string larger than this limit.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns the integer value that indicates the string's maximum allowable length that can be entered from the keyboard.

SEE ALSO

XmText.

NAME

XmTextGetSelection — a Text function that retrieves the value of the primary selection

SYNOPSIS

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

char *XmTextGetSelection(
      Widget                    widget);
```

DESCRIPTION

XmTextGetSelection( ) retrieves the value of the primary selection. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the string by calling *XtFree*( ).

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns a character pointer to the string that is associated with the primary selection.

SEE ALSO

XmText.

NAME

XmTextGetSelectionPosition — a Text function that accesses the position of the primary selection

SYNOPSIS

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

Boolean XmTextGetSelectionPosition(
      Widget                    widget,
      XmTextPosition           *left,
      XmTextPosition           *right);
```

DESCRIPTION

XmTextGetSelectionPosition( ) accesses the left and right position of the primary selection in the text buffer of the Text widget.

widget Specifies the Text widget ID.

left Specifies the pointer in which the position of the left boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

right Specifies the pointer in which the position of the right boundary of the primary selection is returned. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

This function returns True if the widget owns the primary selection; otherwise, it returns False.

SEE ALSO

XmText.

NAME

XmTextGetSource — a Text function that accesses the source of the widget

SYNOPSIS

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

XmTextSource XmTextGetSource(
      Widget                    widget);
```

DESCRIPTION

XmTextGetSource( ) accesses the source of the Text widget. Text widgets can share sources of text so that editing in one widget is reflected in another. This function accesses the source of one widget so that it can be made the source of another widget, using the function *XmTextSetSource*( ).

Setting a new text source destroys the old text source if no other Text widgets are using that source. To replace a text source but keep it for later use, create an unmanaged Text widget and set its source to the text source you want to keep.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns an **XmTextSource** value that represents the source of the Text widget.

SEE ALSO

XmText.

NAME

XmTextGetString — a Text function that accesses the string value

SYNOPSIS

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

char *XmTextGetString(
      Widget                    widget);
```

DESCRIPTION

XmTextGetString( ) accesses the string value of the Text widget. The application is responsible for freeing the storage associated with the string by calling *XtFree*( ).

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns a character pointer to the string value of the text widget. This returned value is a copy of the **XmNvalue** resource. Returns an empty string if the length of the Text widget's string is 0 (zero).

SEE ALSO

XmText.

NAME

XmTextGetSubstring — a Text function that retrieves a copy of a portion of the internal text buffer

SYNOPSIS

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

int XmTextGetSubstring(
      Widget                    widget,
      XmTextPosition            start,
      int                       num_chars,
      int                       buffer_size,
      char                     *buffer);
```

DESCRIPTION

XmTextGetSubstring( ) retrieves a copy of a portion of the internal text buffer of a Text widget. The function copies a specified number of characters from a given start position in the internal text buffer into a buffer provided by the application. A NULL terminator is placed at the end of the copied data.

The size of the required buffer depends on the maximum number of bytes per character (MB_CUR_MAX) for the current locale. MB_CUR_MAX is a macro defined in **<stdlib.h>**. The buffer should be large enough to contain the substring to be copied and a NULL terminator. Use the following equation to calculate the size of buffer the application should provide:

```
   buffer_size = (num_chars * MB_CUR_MAX) + 1
```

widget Specifies the Text widget ID.

start Specifies the beginning character position from which the data is retrieved. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

num_chars Specifies the number of characters to be copied into the buffer provided.

buffer_size Specifies the size of the supplied buffer in bytes. This size should account for a NULL terminator.

buffer Specifies the character buffer into which the internal text buffer is copied.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

[XmCOPY_SUCCEEDED]
The function was successful.

[XmCOPY_FAILED]
The function failed because it was unable to copy the specified number of characters into the buffer provided. The buffer size may be insufficient. The contents of *buffer* are undefined.

[XmCOPY_TRUNCATED]
The requested number of characters extended beyond the internal buffer. The function copied characters between *start* and the end of the widget's buffer and terminated the string with a NULL terminator; fewer than *num_chars* characters were copied.

SEE ALSO

XmText.

NAME

XmTextGetTopCharacter — a Text function that accesses the position of the first character displayed

SYNOPSIS

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

XmTextPosition XmTextGetTopCharacter(
      Widget                    widget);
```

DESCRIPTION

XmTextGetTopCharacter( ) accesses the position of the text at the top of the Text widget.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns an **XmTextPosition** value that indicates the state of the **XmNtopCharacter** resource. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

SEE ALSO

XmText.

NAME

XmTextInsert — a Text function that inserts a character string into a text string

SYNOPSIS

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

void XmTextInsert(
      Widget                    widget,
      XmTextPosition            position,
      char                     *value);
```

DESCRIPTION

XmTextInsert( ) inserts a character string into the text string in the Text widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text. For example, to insert a string after the fourth character, the parameter *position* must be 4.

This routine also calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks.

widget Specifies the Text widget ID.

position Specifies the position in the text string where the character string is to be inserted.

value Specifies the character string value to be added to the text widget.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextPaste — a Text function that inserts the clipboard selection

SYNOPSIS

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

Boolean XmTextPaste(
      Widget                    widget);
```

DESCRIPTION

XmTextPaste( ) inserts the clipboard selection at the insertion cursor of the destination widget. If **XmNpendingDelete** is True and the insertion cursor is inside the current selection, the clipboard selection replaces the selected text. This routine calls the widget's **XmNvalueChangedCallback** and **XmNmodifyVerifyCallback**.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

This function returns False no transfers take place. Otherwise, it returns True.

SEE ALSO

XmText.

NAME

XmTextPosToXY — a Text function that accesses the x and y position of a character position

SYNOPSIS

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

Boolean XmTextPosToXY(
      Widget                     widget,
      XmTextPosition             position,
      Position                  *x,
      Position                  *y);
```

DESCRIPTION

XmTextPosToXY( ) accesses the x and y position, relative to the upper-left corner of the Text widget, of a given character position in the text buffer.

widget Specifies the Text widget ID.

position Specifies the character position in the text for which the x and y position is accessed. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

x Specifies the pointer in which the x position, relative to the upper-left corner of the widget, is returned. This value is meaningful only if the function returns True.

y Specifies the pointer in which the y position, relative to the upper left corner of the widget, is returned. This value is meaningful only if the function returns True.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

This function returns True if the character position is displayed in the Text widget; otherwise, it returns False, and no *x* or *y* value is returned.

SEE ALSO

XmText.

NAME

XmTextRemove — a Text function that deletes the primary selection

SYNOPSIS

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

Boolean XmTextRemove(
      Widget                    widget);
```

DESCRIPTION

XmTextRemove( ) deletes the primary selected text. This routine also calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks if there is a selection.

widget Specifies the Text widget ID.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

This function returns False if the primary selection is NULL or if the *widget* does not own the primary selection. Otherwise, it returns True.

SEE ALSO

XmText.

NAME

XmTextReplace — a Text function that replaces part of a text string

SYNOPSIS

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

void XmTextReplace(
      Widget                   widget,
      XmTextPosition           from_pos,
      XmTextPosition           to_pos,
      char                    *value);
```

DESCRIPTION

XmTextReplace( ) replaces part of the text string in the Text widget. The character positions begin at 0 (zero) and are numbered sequentially from the beginning of the text.

An example text replacement would be to replace the second and third characters in the text string. To accomplish this, the argument *from_pos* must be 1 and *to_pos* must be 3. To insert a string after the fourth character, both arguments, *from_pos* and *to_pos*, must be 4.

This routine also calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks.

widget Specifies the Text widget ID.

from_pos Specifies the start position of the text to be replaced.

to_pos Specifies the end position of the text to be replaced.

value Specifies the character string value to be added to the text widget.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextScroll — a Text function that scrolls text

SYNOPSIS

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

void XmTextScroll(
      Widget                    widget,
      int                       lines);
```

DESCRIPTION

XmTextScroll( ) scrolls text in a Text widget.

widget Specifies the Text widget ID.

lines Specifies the number of lines of text to scroll. A positive value causes text to scroll upward; a negative value causes text to scroll downward.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetAddMode — a Text function that sets the state of Add mode

SYNOPSIS

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

void XmTextSetAddMode(
      Widget                     widget,
      Boolean                    state);
```

DESCRIPTION

XmTextSetAddMode( ) controls whether or not the Text widget is in Add mode. When the widget is in Add mode, the insert cursor can be moved without disturbing the primary selection.

widget Specifies the Text widget ID.

state Specifies whether or not the widget is in Add mode. A value of True turns on Add mode; a value of False turns off Add mode.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetEditable — a Text function that sets the edit permission

SYNOPSIS

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

void XmTextSetEditable(
      Widget                      widget,
      Boolean                     editable);
```

DESCRIPTION

XmTextSetEditable( ) sets the edit permission state of the Text widget. When set to True, the text string can be edited.

widget Specifies the Text widget ID.

editable Specifies a Boolean value that when True allows text string edits.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetHighlight — a Text function that highlights text

SYNOPSIS

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

void XmTextSetHighlight(
      Widget                      widget,
      XmTextPosition              left,
      XmTextPosition              right,
      XmHighlightMode             mode);
```

DESCRIPTION

XmTextSetHighlight( ) highlights text between the two specified character positions. The *mode* argument determines the type of highlighting. Highlighting text merely changes the visual appearance of the text; it does not set the selection.

widget Specifies the Text widget ID.

left Specifies the position of the left boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

right Specifies the position of the right boundary of text to be highlighted. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

mode Specifies the type of highlighting to be done. A value of XmHIGHLIGHT_NORMAL removes highlighting. A value of XmHIGHLIGHT_SELECTED highlights the text using reverse video. A value of XmHIGHLIGHT_SECONDARY_SELECTED highlights the text using underlining.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetInsertionPosition — a Text function that sets the position of the insert cursor

SYNOPSIS

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

void XmTextSetInsertionPosition(
      Widget                    widget,
      XmTextPosition            position);
```

DESCRIPTION

XmTextSetInsertionPosition( ) sets the insertion cursor position of the Text widget. This routine also calls the widget's **XmNmotionVerifyCallback** callbacks if the insertion cursor position changes.

widget Specifies the Text widget ID.

position Specifies the position of the insertion cursor. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetMaxLength — a Text function that sets the value of the current maximum allowable length of a text string entered from the keyboard

SYNOPSIS

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

void XmTextSetMaxLength(
      Widget                     widget,
      int                        max_length);
```

DESCRIPTION

XmTextSetMaxLength( ) sets the value of the current maximum allowable length of the text string in the Text widget. The maximum allowable length prevents the user from entering a text string from the keyboard that is larger than this limit. Strings that are entered using the **XmNvalue** resource or the *XmTextSetString*( ) function ignore this resource.

widget Specifies the Text widget ID.

max_length Specifies the maximum allowable length of the text string.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText and *XmTextSetString*( ).

NAME

XmTextSetSelection — a Text function that sets the primary selection of the text

SYNOPSIS

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

void XmTextSetSelection(
      Widget                    widget,
      XmTextPosition            first,
      XmTextPosition            last,
      Time                      time);
```

DESCRIPTION

XmTextSetSelection( ) sets the primary selection of the text in the widget. It also sets the insertion cursor position to the last position of the selection and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the Text widget ID.

first Marks the first character position of the text to be selected.

last Marks the last position of the text to be selected.

time Specifies the time at which the selection value is desired. This should be the same as the time of the event that triggered this request.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetSource — a Text function that sets the source of the widget

SYNOPSIS

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

void XmTextSetSource(
      Widget                     widget,
      XmTextSource               source,
      XmTextPosition             top_character,
      XmTextPosition             cursor_position);
```

DESCRIPTION

XmTextSetSource( ) sets the source of the Text widget. Text widgets can share sources of text so that editing in one widget is reflected in another. This function sets the source of one widget so that it can share the source of another widget.

Setting a new text source destroys the old text source if no other Text widgets are using that source. To replace a text source but keep it for later use, create an unmanaged Text widget and set its source to the text source you want to keep.

widget Specifies the Text widget ID.

source Specifies the source with which the widget displays text. This can be a value returned by the *XmTextGetSource*( ) function. If no source is specified, the widget creates a default string source.

top_character Specifies the position in the text to display at the top of the widget. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

cursor_position
Specifies the position in the text at which the insert cursor is located. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetString — a Text function that sets the string value

SYNOPSIS

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

void XmTextSetString(
      Widget                     widget,
      char                      *value);
```

DESCRIPTION

XmTextSetString( ) sets the string value of the Text widget. This routine calls the widget's **XmNmodifyVerifyCallback** and **XmNvalueChangedCallback** callbacks. This function also sets the insertion cursor position to the beginning of the string and calls the widget's **XmNmotionVerifyCallback** callbacks.

widget Specifies the Text widget ID.

value Specifies the character pointer to the string value and places the string into the text edit window.

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextSetTopCharacter — a Text function that sets the position of the first character displayed

SYNOPSIS

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

void XmTextSetTopCharacter(
      Widget                    widget,
      XmTextPosition            top_character);
```

DESCRIPTION

XmTextSetTopCharacter( ) sets the position of the text at the top of the Text widget. If the **XmNeditMode** is XmMULTI_LINE_EDIT, the line of text that contains *top_character* is displayed at the top of the widget without the text shifting left or right.

widget Specifies the Text widget ID.

top_character Specifies the position in text of the first character to be displayed at the top of the widget. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextShowPosition — a Text function that forces text at a given position to be displayed

SYNOPSIS

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

void XmTextShowPosition(
      Widget                    widget,
      XmTextPosition            position);
```

DESCRIPTION

XmTextShowPosition( ) forces text at the specified position to be displayed. If the **XmNautoShowCursorPosition** resource is True, the application should also set the insert cursor to this position.

widget Specifies the Text widget ID.

position Specifies the character position to be displayed. This is an integer number of characters from the beginning of the text buffer. The first character position is 0 (zero).

For a complete definition of Text and its associated resources, see *XmText*.

SEE ALSO

XmText.

NAME

XmTextXYToPos — a Text function that accesses the character position nearest an x and y position

SYNOPSIS

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

XmTextPosition XmTextXYToPos(
      Widget                    widget,
      Position                  x,
      Position                  y);
```

DESCRIPTION

XmTextXYToPos( ) accesses the character position nearest to the specified x and y position, relative to the upper-left corner of the Text widget.

widget Specifies the Text widget ID.

x Specifies the x position, relative to the upper-left corner of the widget.

y Specifies the y position, relative to the upper-left corner of the widget.

For a complete definition of Text and its associated resources, see *XmText*.

RETURN VALUE

Returns the character position in the text nearest the x and y position specified. This is an integer number of characters from the beginning of the buffer. The first character position is 0 (zero).

SEE ALSO

XmText.

NAME

XmToggleButton — the ToggleButton widget class

SYNOPSIS

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

DESCRIPTION

ToggleButton sets non-transitory state data within an application. Usually this widget consists of an indicator (for example, a square or diamond) with either text or a pixmap on one side of it. However, it can also consist of just text or a pixmap without the indicator.

The toggle graphics display a 1-of-many or N-of-many selection state. When a toggle indicator is displayed, a square indicator shows an N-of-many selection state and a diamond or circular indicator shows a 1-of-many selection state.

ToggleButton implies a selected or unselected state. In the case of a label and an indicator, an empty indicator (square or diamond shaped) indicates that ToggleButton is unselected, and a filled indicator shows that it is selected. In the case of a pixmap toggle, different pixmaps are used to display the selected or unselected states.

The default behavior associated with a ToggleButton in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the ToggleButton. In addition, **BMenu** controls the behavior of the ToggleButton if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Label's resource **XmNmarginLeft** may be increased to accommodate the toggle indicator when it is created.

Classes

ToggleButton inherits behavior and resources from *Core*, *XmPrimitive* and *XmLabel*.

The class pointer is **xmToggleButtonWidgetClass**.

The class name is *XmToggleButton*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmToggleButton **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNarmCallback	**XmCArmCallback**	**XtCallbackList**	NULL	C
XmNdisarmCallback	**XmCDisarmCallback**	**XtCallbackList**	NULL	C
XmNfillOnSelect	**XmCFillOnSelect**	**Boolean**	dynamic	CSG
XmNindicatorOn	**XmCIndicatorOn**	**Boolean**	True	CSG
XmNindicatorSize	**XmCIndicatorSize**	**Dimension**	dynamic	CSG
XmNindicatorType	**XmCIndicatorType**	**unsigned char**	dynamic	CSG
XmNselectColor	**XmCSelectColor**	**Pixel**	dynamic	CSG
XmNselectInsensitivePixmap	**XmCSelectInsensitivePixmap**	**Pixmap**	XmUNSPECIFIED_PIXMAP	CSG
XmNselectPixmap	**XmCSelectPixmap**	**Pixmap**	XmUNSPECIFIED_PIXMAP	CSG
XmNset	**XmCSet**	**Boolean**	False	CSG
XmNspacing	**XmCSpacing**	**Dimension**	4	CSG
XmNvalueChangedCallback	**XmCValueChangedCallback**	**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.

XmNdisarmCallback

Specifies the list of callbacks called when ToggleButton is disarmed. To disarm this widget, press and release the active mouse button while the pointer is inside the ToggleButton. This widget is also disarmed when the user moves out of the widget and releases the mouse button when the pointer is outside the widget. For this callback, the reason is XmCR_DISARM.

XmNfillOnSelect

Fills the indicator with the color specified in **XmNselectColor** and switches the top and bottom shadow colors when set to True. Otherwise, it switches only the top and bottom shadow colors. The default is set to the value of **XmNindicatorOn**. When **XmNindicatorOn** is False, and **XmNfillOnSelect** is set explicitly to True, the background is filled with the color specified by **XmNselectColor**.

XmNindicatorOn

Specifies that a toggle indicator is drawn to one side of the toggle text or pixmap when set to True. When set to False, no space is allocated for the indicator, and it is not displayed. If **XmNindicatorOn** is True, the indicator shadows are switched when the button is selected or unselected, but any shadows around the entire widget are not switched. However, if **XmNindicatorOn** is False, any shadows around the entire widget are switched when the toggle is selected or unselected.

XmNindicatorSize

Sets the size of the indicator. If no value is specified, the size of the indicator is based on the size of the label string or pixmap. If the label string or pixmap changes, the size of the indicator is recomputed based on the size of the label string or pixmap. Once a value has been specified for **XmNindicatorSize**, the indicator has that size, regardless of the size of the label string or pixmap, until a new value is specified. The size of indicators inside menus may differ from those outside menus.

XmNindicatorType

Specifies if the indicator is a 1-of or N-of indicator. For the 1-of indicator, the value is XmONE_OF_MANY. For the N-of indicator, the value is XmN_OF_MANY. The N-of-many indicator is square. The 1-of-many indicator is diamond shaped. This resource specifies only the visual symbols and does not enforce the behavior. When the ToggleButton is in a RadioBox, the default is XmONE_OF_MANY; otherwise, the default is XmN_OF_MANY.

XmNselectColor

Allows the application to specify what color fills the center of the square or diamond-shaped indicator when it is set. If this color is the same as either the top or the bottom shadow color of the indicator, a one-pixel-wide margin is left between the shadows and the fill; otherwise, it is filled completely. This resource's default for a color display is a color between the background and the bottom shadow color. For a monochrome display, the default is set to the foreground color. To set the background of the button to **XmNselectColor** when **XmNindicatorOn** is False, the value of **XmNfillOnSelect** must be explicitly set to True.

XmNselectInsensitivePixmap

Specifies a pixmap used as the button face when the ToggleButton is selected, the button is insensitive, and the Label resource **XmNlabelType** is set to XmPIXMAP. If the ToggleButton is unselected and the button is insensitive, the pixmap in **XmNlabelInsensitivePixmap** is used as the button face. If no value is specified for **XmNlabelInsensitivePixmap**, that resource is set to the value specified for **XmNselectInsensitivePixmap**.

XmNselectPixmap

Specifies the pixmap to be used as the button face when **XmNlabelType** is XmPIXMAP and the ToggleButton is selected. When the ToggleButton is unselected, the pixmap specified in the Label's **XmNlabelPixmap** is used. If no value is specified for **XmNlabelPixmap**, that resource is set to the value specified for **XmNselectPixmap**.

XmNset

Represents the state of the ToggleButton. A value of False indicates that the ToggleButton is not set. A value of True indicates that the ToggleButton is set. Setting this resource sets the state of the ToggleButton. Setting this resource sets the state of the ToggleButton.

XmNspacing

Specifies the amount of spacing between the toggle indicator and the toggle label (text or pixmap).

XmNvalueChangedCallback

Specifies the list of callbacks called when the ToggleButton value is changed. To change the value, press and release the active mouse button while the pointer is inside the ToggleButton. This action also causes this widget to be disarmed. For this callback, the reason is XmCR_VALUE_CHANGED.

XmNvisibleWhenOff

Indicates that the toggle indicator is visible in the unselected state when the Boolean value is True. When the ToggleButton is in a menu, the default value is False. When the ToggleButton is in a RadioBox, the default value is True.

Inherited Resources

ToggleButton inherits behavior and resources from the superclasses 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
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	XmFONTLIST_ DEFAULT_TAG	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CSG

XmPrimitive **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNbottomShadowColor	**XmCBottomShadowColor**	**Pixel**	dynamic	CSG
XmNbottomShadowPixmap	**XmCBottomShadowPixmap**	**Pixmap**	XmUNSPECIFIED _PIXMAP	CSG
XmNforeground	**XmCForeground**	**Pixel**	dynamic	CSG
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightColor	**XmCHighlightColor**	**Pixel**	dynamic	CSG
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightPixmap	**XmCHighlightPixmap**	**Pixmap**	dynamic	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtopShadowColor	**XmCTopShadowColor**	**Pixel**	dynamic	CSG
XmNtopShadowPixmap	**XmCTopShadowPixmap**	**Pixmap**	dynamic	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	dynamic	G
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                        set;
} XmToggleButtonCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback.

set Reflects the ToggleButton's current state when the callback occurred, either True (selected) or False (unselected).

Action Routines

The *XmToggleButton* action routines are:

Arm( )

If the button was previously unset, this action does the following: if **XmNindicatorOn** is True, it draws the indicator shadow so that the indicator looks pressed; if **XmNfillOnSelect** is True, it fills the indicator with the color specified by **XmNselectColor**. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks pressed. If **XmNlabelType** is XmPIXMAP, the **XmNselectPixmap** is used as the button face. This action calls the **XmNarmCallback** callbacks.

If the button was previously set, this action does the following: if both **XmNindicatorOn** and **XmNvisibleWhenOff** are True, it draws the indicator shadow so that the indicator looks raised; if **XmNfillOnSelect** is True, it fills the indicator with the background color. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks raised. If **XmNlabelType** is XmPIXMAP, the **XmNlabelPixmap** is used as the button face. This action calls the **XmNarmCallback** callbacks.

ArmAndActivate( )

If the ToggleButton was previously set, unsets it; if the ToggleButton was previously unset, sets it.

In a menu, this action unposts all menus in the menu hierarchy. Unless the button is already armed, it calls the **XmNarmCallback** callbacks. This action calls the **XmNvalueChangedCallback** and **XmNdisarmCallback** callbacks.

Outside a menu, if the button was previously unset, this action does the following: if **XmNindicatorOn** is True, it draws the indicator shadow so that the indicator looks pressed; if **XmNfillOnSelect** is True, it fills the indicator with the color specified by **XmNselectColor**. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks pressed. If **XmNlabelType** is XmPIXMAP, the **XmNselectPixmap** is used as the button face. This action calls the **XmNarmCallback**, **XmNvalueChangedCallback** and **XmNdisarmCallback** callbacks.

Outside a menu, if the button was previously set, this action does the following: if both **XmNindicatorOn** and **XmNvisibleWhenOff** are True, it draws the indicator shadow so that the indicator looks raised; if **XmNfillOnSelect** is True, it fills the indicator with the background color. If **XmNindicatorOn** is False, it draws the button shadow so that the button looks raised. If **XmNlabelType** is XmPIXMAP, the **XmNlabelPixmap** is used as the button face. This action calls the **XmNarmCallback**, **XmNvalueChangedCallback** and **XmNdisarmCallback** callbacks.

BtnDown( )

This action unposts any menus posted by the ToggleButton's parent menu, disables keyboard traversal for the menu, and enables mouse traversal for the menu. It draws the shadow in the armed state and, unless the button is already armed, calls the **XmNarmCallback** callbacks.

BtnUp( )

This action unposts all menus in the menu hierarchy. If the ToggleButton was previously set, unsets it; if the ToggleButton was previously unset, sets it. It calls the **XmNvalueChangedCallback** callbacks and then the **XmNdisarmCallback** callbacks.

Disarm( )

Calls the callbacks for **XmNdisarmCallback**.

Help( )

In a Pulldown or Popup MenuPane, unposts all menus in the menu hierarchy and restores keyboard focus to the tab group 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 tab group 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.

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.

SEE ALSO

Core, *XmCreateRadioBox*( ), *XmCreateToggleButton*( ), *XmLabel*, *XmPrimitive*, *XmRowColumn*, *XmToggleButtonGetState*( ) and *XmToggleButtonSetState*( ).

NAME

XmToggleButtonGadget — the ToggleButtonGadget widget class

SYNOPSIS

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

DESCRIPTION

ToggleButtonGadget sets non-transitory state data within an application. Usually this gadget consists of an indicator (square or diamond-shaped) with either text or a pixmap on one side of it. However, it can also consist of just text or a pixmap without the indicator.

The toggle graphics display a 1-of-many or N-of-many selection state. When a toggle indicator is displayed, a square indicator shows an N-of-many selection state and a diamond-shaped indicator shows a 1-of-many selection state.

ToggleButtonGadget implies a selected or unselected state. In the case of a label and an indicator, an empty indicator (square or diamond-shaped) indicates that ToggleButtonGadget is unselected, and a filled indicator shows that it is selected. In the case of a pixmap toggle, different pixmaps are used to display the selected or unselected states.

The default behavior associated with a ToggleButtonGadget in a menu depends on the type of menu system in which it resides. By default, **BSelect** controls the behavior of the ToggleButtonGadget. In addition, **BMenu** controls the behavior of the ToggleButtonGadget if it resides in a PopupMenu system. The actual mouse button used is determined by its RowColumn parent.

Label's resource **XmNmarginLeft** may be increased to accommodate the toggle indicator when it is created.

Classes

ToggleButtonGadget inherits behavior and resources from *Object*, *RectObj*, *XmGadget* and *XmLabelGadget*.

The class pointer is **xmToggleButtonGadgetClass**.

The class name is *XmToggleButtonGadget*.

New Resources

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a resource by name or by class in a **.Xdefaults** file, remove the **XmN** or **XmC** prefix and use the remaining letters. To specify one of the defined values for a resource in a **.Xdefaults** file, remove the **Xm** prefix and use the remaining letters (in either lowercase or uppercase, but include any underscores between words). The codes in the access column indicate 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).

XmToggleButtonGadget **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNarmCallback	**XmCArmCallback**	**XtCallbackList**	NULL	C
XmNdisarmCallback	**XmCDisarmCallback**	**XtCallbackList**	NULL	C
XmNfillOnSelect	**XmCFillOnSelect**	**Boolean**	dynamic	CSG
XmNindicatorOn	**XmCIndicatorOn**	**Boolean**	True	CSG
XmNindicatorSize	**XmCIndicatorSize**	**Dimension**	dynamic	CSG
XmNindicatorType	**XmCIndicatorType**	**unsigned char**	dynamic	CSG
XmNselectColor	**XmCSelectColor**	**Pixel**	dynamic	CSG
XmNselectInsensitivePixmap	**XmCSelectInsensitivePixmap**	**Pixmap**	XmUNSPECIFIED_PIXMAP	CSG
XmNselectPixmap	**XmCSelectPixmap**	**Pixmap**	XmUNSPECIFIED_PIXMAP	CSG
XmNset	**XmCSet**	**Boolean**	False	CSG
XmNspacing	**XmCSpacing**	**Dimension**	4	CSG
XmNvalueChangedCallback	**XmCValueChangedCallback**	**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.

XmNdisarmCallback

Specifies a list of callbacks called when ToggleButtonGadget is disarmed. To disarm this gadget, press and release the active mouse button while the pointer is inside the ToggleButtonGadget. The gadget is also disarmed when the user moves out of the gadget and releases the mouse button when the pointer is outside the gadget. For this callback, the reason is XmCR_DISARM.

XmNfillOnSelect

Fills the indicator with the color specified in **XmNselectColor** and switches the top and bottom shadow colors when set to True. Otherwise, it switches only the top and bottom shadow colors. The default is set to the value of **XmNindicatorOn**. When **XmNindicatorOn** is False, and **XmNfillOnSelect** is set explicitly to True, the background is filled with the color specified by **XmNselectColor**.

XmNindicatorOn

Specifies that a toggle indicator is drawn to one side of the toggle text or pixmap when set to True. When set to False, no space is allocated for the indicator, and it is not displayed. If **XmNindicatorOn** is True, the indicator shadows are switched when the button is selected or unselected, but any shadows around the entire widget are not switched. However, if **XmNindicatorOn** is False, any shadows around the entire widget are switched when the toggle is selected or unselected.

XmNindicatorSize

Sets the size of the indicator. If no value is specified, the size of the indicator is based on the size of the label string or pixmap. If the label string or pixmap changes, the size of the indicator is recomputed based on the size of the label string or pixmap. Once a value has been specified for **XmNindicatorSize**, the indicator has that size, regardless of the size of the label string or pixmap, until a new value is specified. The size of indicators inside menus may differ from those outside menus.

XmNindicatorType

Specifies if the indicator is a 1-of or an N-of indicator. For the 1-of indicator, the value is XmONE_OF_MANY. For the N-of indicator, the value is XmN_OF_MANY. The N-of-many indicator is square. The 1-of-many indicator is diamond-shaped. This resource specifies only the visual symbols and does not enforce the behavior. When the ToggleButtonGadget is in a RadioBox, the default is XmONE_OF_MANY; otherwise, the default is XmN_OF_MANY.

XmNselectColor

Allows the application to specify what color fills the center of the square or diamond-shaped indicator when it is set. If this color is the same as either the top or the bottom shadow color of the indicator, a one-pixel-wide margin is left between the shadows and the fill; otherwise, it is filled completely. This resource's default for a color display is a color between the background and the bottom shadow color. For a monochrome display, the default is set to the foreground color. The meaning of this resource is undefined when **XmNindicatorOn** is False.

XmNselectInsensitivePixmap

Specifies a pixmap used as the button face when the ToggleButtonGadget is selected, the button is insensitive, and the LabelGadget resource **XmNlabelType** is XmPIXMAP. If the ToggleButtonGadget is unselected and the button is insensitive, the pixmap in **XmNlabelInsensitivePixmap** is used as the button face. If no value is specified for **XmNlabelInsensitivePixmap**, that resource is set to the value specified for **XmNselectInsensitivePixmap**.

XmNselectPixmap

Specifies the pixmap to be used as the button face if **XmNlabelType** is XmPIXMAP and the ToggleButtonGadget is selected. When the ToggleButtonGadget is unselected, the pixmap specified in LabelGadget's **XmNlabelPixmap** is used. If no value is specified for **XmNlabelPixmap**, that resource is set to the value specified for **XmNselectPixmap**.

XmNset

Represents the state of the ToggleButton. A value of false indicates that the ToggleButton is not set. A value of true indicates that the ToggleButton is set. Setting this resource sets the state of the ToggleButton. Setting this resource sets the state of the ToggleButton.

XmNspacing

Specifies the amount of spacing between the toggle indicator and the toggle label (text or pixmap).

XmNvalueChangedCallback

Specifies a list of callbacks called when the ToggleButtonGadget value is changed. To change the value, press and release the active mouse button while the pointer is inside the ToggleButtonGadget. This action also causes the gadget to be disarmed. For this callback, the reason is XmCR_VALUE_CHANGED.

XmNvisibleWhenOff

Indicates that the toggle indicator is visible in the unselected state when the Boolean value is True. When the ToggleButtonGadget is in a menu, the default value is False. When the ToggleButtonGadget is in a RadioBox, the default value is True.

Inherited Resources

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

XmLabelGadget **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNaccelerator	**XmCAccelerator**	**String**	NULL	CSG
XmNacceleratorText	**XmCAcceleratorText**	**XmString**	NULL	CSG
XmNalignment	**XmCAlignment**	**unsigned char**	dynamic	CSG
XmNfontList	**XmCFontList**	**XmFontList**	dynamic	CSG
XmNlabelInsensitivePixmap	**XmCLabelInsensitivePixmap**	**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
XmNmnemonicCharSet	**XmCMnemonicCharSet**	**String**	dynamic	CSG
XmNrecomputeSize	**XmCRecomputeSize**	**Boolean**	True	CSG
XmNstringDirection	**XmCStringDirection**	**XmStringDirection**	dynamic	CSG

XmGadget **Resource Set**				
Name	**Class**	**Type**	**Default**	**Access**
XmNhelpCallback	**XmCCallback**	**XtCallbackList**	NULL	C
XmNhighlightOnEnter	**XmCHighlightOnEnter**	**Boolean**	False	CSG
XmNhighlightThickness	**XmCHighlightThickness**	**Dimension**	0	CSG
XmNnavigationType	**XmCNavigationType**	**XmNavigationType**	XmNONE	CSG
XmNshadowThickness	**XmCShadowThickness**	**Dimension**	2	CSG
XmNtraversalOn	**XmCTraversalOn**	**Boolean**	True	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 current state when the callback occurred, either True (selected) or False (unselected).

SEE ALSO

Object, *RectObj*, *XmCreateRadioBox*( ), *XmCreateToggleButtonGadget*( ), *XmGadget*, *XmLabelGadget*, *XmRowColumn*, *XmToggleButtonGadgetGetState*( ) and *XmToggleButtonGadgetSetState*( ).

NAME

XmToggleButtonGadgetGetState — a ToggleButtonGadget function that obtains the state of a ToggleButtonGadget

SYNOPSIS

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

Boolean XmToggleButtonGadgetGetState(
      Widget                    widget);
```

DESCRIPTION

XmToggleButtonGadgetGetState( ) obtains the state of a ToggleButtonGadget.

widget Specifies the ToggleButtonGadget ID.

For a complete definition of ToggleButtonGadget and its associated resources, see *XmToggleButtonGadget*.

RETURN VALUE

Returns True if the button is selected and False if the button is unselected.

SEE ALSO

XmToggleButtonGadget.

NAME

XmToggleButtonGadgetSetState — a ToggleButtonGadget function that sets or changes the current state

SYNOPSIS

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

void XmToggleButtonGadgetSetState(
      Widget                     widget,
      Boolean                    state,
      Boolean                    notify);
```

DESCRIPTION

XmToggleButtonGadgetSetState( ) sets or changes the ToggleButtonGadget's current state.

widget Specifies the ToggleButtonGadget widget ID.

state Specifies a Boolean value that indicates whether the ToggleButtonGadget state is selected or unselected. If the value is True, the button state is selected; if it is False, the button state is unselected.

notify Indicates whether **XmNvalueChangedCallback** is called; it can be either True or False. The **XmNvalueChangedCallback** is only called when this function changes the state of the ToggleButtonGadget. When this argument is True and the ToggleButtonGadget is a child of a RowColumn widget whose **XmNradioBehavior** is True, setting the ToggleButtonGadget causes other ToggleButton and ToggleButtonGadget children of the RowColumn to be unselected.

For a complete definition of ToggleButtonGadget and its associated resources, see *XmToggleButtonGadget*.

SEE ALSO

XmToggleButtonGadget.

NAME

XmToggleButtonGetState — a ToggleButton function that obtains the state of a ToggleButton

SYNOPSIS

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

Boolean XmToggleButtonGetState(
      Widget                     widget);
```

DESCRIPTION

XmToggleButtonGetState( ) obtains the state of a ToggleButton.

widget Specifies the ToggleButton widget ID.

For a complete definition of ToggleButton and its associated resources, see *XmToggleButton*.

RETURN VALUE

Returns True if the button is selected and False if the button is unselected.

SEE ALSO

XmToggleButton.

NAME

XmToggleButtonSetState — a ToggleButton function that sets or changes the current state

SYNOPSIS

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

void XmToggleButtonSetState(
      Widget                    widget,
      Boolean                   state,
      Boolean                   notify);
```

DESCRIPTION

XmToggleButtonSetState( ) sets or changes the ToggleButton's current state.

widget Specifies the ToggleButton widget ID.

state Specifies a Boolean value that indicates whether the ToggleButton state is selected or unselected. If the value is True, the button state is selected; if it is False, the button state is unselected.

notify Indicates whether **XmNvalueChangedCallback** is called; it can be either True or False. The **XmNvalueChangedCallback** is only called when this function changes the state of the ToggleButton. When this argument is True and the ToggleButton is a child of a RowColumn widget whose **XmNradioBehavior** is True, setting the ToggleButton causes other ToggleButton and ToggleButtonGadget children of the RowColumn to be unselected.

For a complete definition of ToggleButton and its associated resources, see *XmToggleButton*.

SEE ALSO

XmToggleButton.

NAME

XmTrackingEvent — a Toolkit function that provides a modal interaction

SYNOPSIS

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

Widget XmTrackingEvent(
      Widget                     widget,
      Cursor                     cursor,
      Boolean                    confine_to,
      XEvent                    *event_return);
```

DESCRIPTION

XmTrackingEvent( ) provides a modal interface for selection of a component. It is intended to support context help. The function calls *XmUpdateDisplay*( ) to grab the pointer and discard succeeding events until **BSelect** is released or a key is pressed and then released. The function then returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released, and ungrabs the pointer.

widget Specifies the widget ID of a widget to use as the basis of the modal interaction. That is, the widget within which the interaction must occur, usually a top-level shell.

cursor Specifies the cursor to be used for the pointer during the interaction. This is a standard X cursor name.

confine_to Specifies whether or not the cursor should be confined to *widget*.

event_return Returns the ButtonRelease or KeyRelease event that causes the function to return.

RETURN VALUE

Returns the widget or gadget that contains the pointer when **BSelect** is released or a key is released. If no widget or gadget contains the pointer, the function returns NULL.

SEE ALSO

XmTrackingLocate( ).

NAME

XmTrackingLocate — a Toolkit function that provides a modal interaction

SYNOPSIS

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

Widget XmTrackingLocate(
      Widget                    widget,
      Cursor                    cursor,
      Boolean                   confine_to);
```

DESCRIPTION

XmTrackingLocate( ) provides a modal interface for selection of a component. It is intended to support context help. The function grabs the pointer and returns the widget in which a button press occurs.

widget Specifies the widget ID of a widget to use as the basis of the modal interaction. That is, the widget within which the interaction must occur, usually a top-level shell.

cursor Specifies the cursor to be used for the pointer during the interaction. This is a standard X cursor name.

confine_to Specifies whether or not the cursor should be confined to *widget*.

RETURN VALUE

Returns the widget in which a button press occurs. If the window in which a button press occurs is not a widget, the function returns NULL.

NAME

XmUninstallImage — a pixmap function that removes an image from the image cache

SYNOPSIS

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

Boolean XmUninstallImage(
      XImage                   *image);
```

DESCRIPTION

XmUninstallImage( ) removes an image from the image cache.

image Points to the image structure given to the *XmInstallImage*( ) routine.

RETURN VALUE

Returns True when successful; returns False if the *image* is NULL, or if it cannot be found.

SEE ALSO

XmInstallImage( ), *XmGetPixmap*( ) and *XmDestroyPixmap*( ).

NAME

XmUpdateDisplay — a function that processes all pending exposure events immediately

SYNOPSIS

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

void XmUpdateDisplay(
      Widget                    widget);
```

DESCRIPTION

XmUpdateDisplay( ) provides the application with a mechanism for forcing all pending exposure events to be removed from the input queue and processed immediately.

When a user selects a button within a MenuPane, the MenuPanes are unposted and then any activation callbacks registered by the application are invoked. If one of the callbacks performs a time-consuming action, the portion of the application window that was covered by the MenuPanes is not redrawn; normal exposure processing does not occur until all of the callbacks have been invoked. If the application writer suspects that a callback could take a long time, the callback may invoke *XmUpdateDisplay*( ) before starting its time-consuming operation.

This function is also useful any time a transient window, such as a dialog box, is unposted; callbacks are invoked before normal exposure processing can occur.

widget   Specifies any widget or gadget.

NAME

XmWidgetGetBaselines — retrieves baseline information for a widget

SYNOPSIS

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

Boolean XmWidgetGetBaselines(
      Widget                     widget,
      Dimension                **baselines,
      int                       *line_count);
```

DESCRIPTION

XmWidgetGetBaselines( ) returns an array that contains one or more baseline values associated with the specified widget. The baseline of any given line of text is a vertical offset in pixels from the origin of the widget's bounding box to the given baseline. This routine allocates memory for the returned data. The application must free this memory using *XtFree*( ).

widget Specifies the ID of the widget for which baseline values are requested.

baselines Returns an array that contains the value of each baseline of text in the widget.

line_count Returns the number of lines in the widget.

RETURN VALUE

Returns a Boolean value that indicates whether the widget contains a baseline. If the value is True, the function returns a value for each baseline of text. If it is False, the function was unable to return a baseline value.

SEE ALSO

XmWidgetGetDisplayRect( ).

NAME

XmWidgetGetDisplayRect — retrieves display rectangle information for a widget

SYNOPSIS

```
#include <Xm/Xm.h>

Boolean XmWidgetGetDisplayRect(
      Widget        widget,
      XRectangle   *displayrect);
```

DESCRIPTION

XmWidgetGetDisplayRect( ) returns the width, height and the x and y coordinates of the upper-left corner of the display rectangle of the specified widget. The display rectangle is the smallest rectangle that encloses either a string or a pixmap.

If the widget contains a string, the return values specify the x and y coordinates of the upper-left corner of the display rectangle relative to the origin of the widget and the width and height in pixels.

In the case of a pixmap, the return values specify the x and y coordinates of the upper-left corner of the pixmap relative to the origin, and the width and height of the pixmap in pixels.

widget Specifies the widget ID.

displayrect Specifies a pointer to an XRectangle structure in which the x and y coordinates, width and height of the display rectangle are returned.

RETURN VALUE

Returns True if the specified widget has an associated display rectangle; otherwise, returns False.

SEE ALSO

XmWidgetGetBaselines( ).

Appendix A

Headers

The header listings in this appendix are offered as examples only. Except as noted by explicit requirements in preceding chapters, the text layout of system headers, the values of symbolic constants and enums, the organization of structs (order of members, size of the struct, additional members and their placement), and actual C-language types assigned to typedefs are not specified by this document and may vary from system to system. Note however that implementations using different structures and values incur the risk of being unable to interoperate with other implementations, either between client and Window Manager or between two clients.

A.1 Function Declarations

The following #include headers contain nothing but the function declarations that are already in their specification. For these files, only their names are listed here to prompt Motif programmers as to which file names to include.

```
#include <Xm/ArrowB.h>
#include <Xm/ArrowBG.h>
#include <Xm/AtomMgr.h>
#include <Xm/BulletinB.h>
#include <Xm/CascadeB.h>
#include <Xm/CascadeBG.h>
#include <Xm/Command.h>
#include <Xm/DialogS.h>
#include <Xm/DragOverS.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/FileSB.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/LabelG.h>
#include <Xm/List.h>
#include <Xm/MainW.h>
#include <Xm/MenuShell.h>
#include <Xm/MessageB.h>
#include <Xm/PanedW.h>
#include <Xm/Protocols.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/Scale.h>
#include <Xm/Screen.h>
#include <Xm/ScrollBar.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectioB.h>
#include <Xm/SeparatoG.h>
#include <Xm/Separator.h>
#include <Xm/TextF.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>
```

A.2 <Xm/CutPaste.h>

```
#ifndef _XmCutPaste_h
#define _XmCutPaste_h

typedef enum {
  XmClipboardFail       = 0,
  XmClipboardSuccess    = 1,
  XmClipboardTruncate   = 2,
  XmClipboardLocked     = 4,
  XmClipboardBadFormat  = 5,
  XmClipboardNoData     = 6,
} XmClipboardStatus;

typedef struct {
    long DataId;
    long PrivateId;
} XmClipboardPendingRec, *XmClipboardPendingList;

typedef void (*XmCutPasteProc)( Widget w, long * data_id,
                                long * private_id, int * reason) ;
#endif
```

A.3 <Xm/Display.h>

```
 Only one enum type, the values of which are referred to in the specs. *
enum {
    XmDRAG_NONE,
    XmDRAG_DROP_ONLY,
    XmDRAG_PREFER_PREREGISTER,
    XmDRAG_PREREGISTER,
    XmDRAG_PREFER_DYNAMIC,
    XmDRAG_DYNAMIC,
    XmDRAG_PREFER_RECEIVER
};

<Xm/DragC.h>

#ifndef _XmDragController_h
#define _XmDragController_h

#define XmHELP   2

typedef unsigned int    XmID;

/* DragContext */

/* enums used for the message_type in client messages */

enum{   XmTOP_LEVEL_ENTER,              XmTOP_LEVEL_LEAVE,
        XmDRAG_MOTION,                  XmDROP_SITE_ENTER,
        XmDROP_SITE_LEAVE,              XmDROP_START,
        XmDROP_FINISH,                  XmDRAG_DROP_FINISH,
        XmOPERATION_CHANGED
        } ;

/* enums for completionStatus */
enum{   XmDROP,                         XmDROP_HELP,
        XmDROP_CANCEL,                  XmDROP_INTERRUPT
        } ;

/* values for operation */
#define XmDROP_NOOP     0L
#define XmDROP_MOVE     (1L << 0)
#define XmDROP_COPY     (1L << 1)
#define XmDROP_LINK     (1L << 2)

enum{   XmBLEND_ALL,                    XmBLEND_STATE_SOURCE,
        XmBLEND_JUST_SOURCE,            XmBLEND_NONE
        } ;

enum{   XmDROP_FAILURE,                 XmDROP_SUCCESS
        } ;
```

```
/* enums used for the public callback reason */

enum{   XmCR_TOP_LEVEL_ENTER,           XmCR_TOP_LEVEL_LEAVE,
        XmCR_DRAG_MOTION,               XmCR_DROP_SITE_ENTER,
        XmCR_DROP_SITE_LEAVE,           XmCR_DROP_START,
        XmCR_DROP_FINISH,               XmCR_DRAG_DROP_FINISH,
        XmCR_OPERATION_CHANGED,
        _XmNUMBER_DND_CB_REASONS
        } ;

/* Class record constants */
typedef struct _XmDragContextClassRec  *XmDragContextClass;
typedef struct _XmDragContextRec       *XmDragContext;

typedef struct _XmAnyICCCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
}XmAnyICCCallbackStruct, *XmAnyICCCallback;

typedef struct _XmTopLevelEnterCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
    Screen         *screen;
    Window          window;
    Position        x, y;
    unsigned char   dragProtocolStyle;
    Atom            iccHandle;
}XmTopLevelEnterCallbackStruct, *XmTopLevelEnterCallback;

typedef struct _XmTopLevelLeaveCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
    Screen         *screen;
    Window          window;
}XmTopLevelLeaveCallbackStruct, *XmTopLevelLeaveCallback;

typedef struct _XmDropSiteEnterCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
    Position        x, y;
}XmDropSiteEnterCallbackStruct, *XmDropSiteEnterCallback;
```

```
typedef struct _XmDropSiteLeaveCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
}XmDropSiteLeaveCallbackStruct, *XmDropSiteLeaveCallback;

typedef struct _XmDragMotionCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
    Position        x, y;
}XmDragMotionCallbackStruct, *XmDragMotionCallback;

typedef struct _XmOperationChangedCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
}XmOperationChangedCallbackStruct, *XmOperationChangedCallback;

typedef struct _XmDropStartCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropSiteStatus;
    unsigned char   dropAction;
    Position        x, y;
    Window          window;
    Atom            iccHandle;
}XmDropStartCallbtruct, *XmDropStartCallback;

typedef struct _XmDropFinishCallbackStruct{
    int             reason;
    XEvent         *event;
    Time           timeStamp;
    unsigned char  operation;
    unsigned char  operations;
    unsigned char  dropSiteStatus;
    unsigned char  dropAction;
    unsigned char  completionStatus;
}XmDropFinishCallbackStruct, *XmDropFinishCallback;
```

```
typedef struct _XmDragDropFinishCallbackStruct{
    int             reason;
    XEvent         *event;
    Time            timeStamp;
}XmDragDropFinishCallbackStruct, *XmDragDropFinishCallback;

#endif /* _DragController_h */
```

A.4 <Xm/DragDrop.h>

```
#ifndef _XmDragDrop_h
#define _XmDragDrop_h

#include <Xm/DragC.h>
#include <Xm/DragIcon.h>
#include <Xm/DropTrans.h>
#include <Xm/DropSMgr.h>

#endif /* _XmDragDrop_h */
```

A.5 <Xm/DragIcon.h>

```
#ifndef _XmDragIcon_h
#define _XmDragIcon_h

enum {
    XmATTACH_NORTH_WEST,
    XmATTACH_NORTH,
    XmATTACH_NORTH_EAST,
    XmATTACH_EAST,
    XmATTACH_SOUTH_EAST,
    XmATTACH_SOUTH,
    XmATTACH_SOUTH_WEST,
    XmATTACH_WEST,
    XmATTACH_CENTER,
    XmATTACH_HOT
};

#endif /* _XmDragIcon_h */
```

A.6 <Xm/DropSMgr.h>

```
#ifndef _XmDropSMgr_h
#define _XmDropSMgr_h

#define XmCR_DROP_SITE_LEAVE_MESSAGE  1
#define XmCR_DROP_SITE_ENTER_MESSAGE  2
#define XmCR_DROP_SITE_MOTION_MESSAGE 3
#define XmCR_DROP_MESSAGE             4

#define XmNO_DROP_SITE         1
#define XmINVALID_DROP_SITE    2
#define XmVALID_DROP_SITE    3

enum { XmDRAG_UNDER_NONE, XmDRAG_UNDER_PIXMAP,
    XmDRAG_UNDER_SHADOW_IN, XmDRAG_UNDER_SHADOW_OUT,
    XmDRAG_UNDER_HIGHLIGHT };

enum { XmDROP_SITE_SIMPLE, XmDROP_SITE_COMPOSITE,
    XmDROP_SITE_SIMPLE_CLIP_ONLY = 128,
    XmDROP_SITE_COMPOSITE_CLIP_ONLY };

enum { XmABOVE, XmBELOW };

enum { XmDROP_SITE_ACTIVE, XmDROP_SITE_INACTIVE };

typedef struct _XmDragProcCallbackStruct {
    int             reason;
    XEvent *        event;
    Time            timeStamp;
    Widget          dragContext;
    Position        x, y;
    unsigned char   dropSiteStatus;
    unsigned char   operation;
    unsigned char   operations;
    Boolean         animate;
} XmDragProcCallbackStruct, * XmDragProcCallback;

typedef struct _XmDropProcCallbackStruct {
    int             reason;
    XEvent *        event;
    Time            timeStamp;
    Widget          dragContext;
    Position        x, y;
    unsigned char   dropSiteStatus;
    unsigned char   operation;
    unsigned char   operations;
    unsigned char   dropAction;
} XmDropProcCallbackStruct, * XmDropProcCallback;

#endif /* _XmDropSMgr_h */
```

A.7 <Xm/DropTrans.h>

```
#ifndef _XmDropTrans_h
#define _XmDropTrans_h

#define XmTRANSFER_FAILURE 0
#define XmTRANSFER_SUCCESS 1

typedef struct _XmDropTransferEntryRec {
    XtPointer   client_data;
    Atom        target;
} XmDropTransferEntryRec, * XmDropTransferEntry;

#endif /* _XmDropTrans_h */
```

A.8 <Xm/MrmPublic.h>

```
#ifndef MrmPublic
#define MrmPublic

/* Success or other non-error return codes */

#define MrmSUCCESS          1
#define MrmCREATE_NEW       3
#define MrmINDEX_RETRY      5   /* Retry on entering index required */
#define MrmINDEX_GT         7   /* Index orders greater-than entry  */
#define MrmINDEX_LT         9   /* Index orders less-than entry     */
#define MrmPARTIAL_SUCCESS 11   /* operation partly succeeded       */

/* Failure return codes */

#define MrmFAILURE              0
#define MrmNOT_FOUND            2
#define MrmEXISTS               4
#define MrmNUL_GROUP            6
#define MrmNUL_TYPE             8
#define MrmWRONG_GROUP         10
#define MrmWRONG_TYPE          12
#define MrmOUT_OF_RANGE        14 /* Record number too big            */
#define MrmBAD_RECORD          16 /* Record number wrong type         */
#define MrmNULL_DATA           18 /* No data for entry                */
#define MrmBAD_DATA_INDEX      20 /* Data index in RID out of range   */
#define MrmBAD_ORDER           22 /* Bad ordering specifier           */
#define MrmBAD_CONTEXT         24 /* Invalid Mrm context              */
#define MrmNOT_VALID           26 /* Validation failure               */
#define MrmBAD_BTREE           28 /* GT/LT pointer error in BTree     */
#define MrmBAD_WIDGET_REC      30 /* Validation failure on widget     */
                                 /* record                           */
#define MrmBAD_CLASS_TYPE      32 /* Class type not a                 */
                                 /* valid Mrmwc... value             */
#define MrmNO_CLASS_NAME       34 /* User class name is null          */
#define MrmTOO_MANY            36 /* Too many entries requested in    */
                                 /* some list                        */
#define MrmBAD_IF_MODULE       38 /* invalid interface module         */
#define MrmNULL_DESC           40 /* Arglist or children descriptor   */
                                 /* null                             */
#define MrmOUT_OF_BOUNDS       42 /* Argument index out of arglist    */
                                 /*  bounds                          */
#define MrmBAD_COMPRESS        44 /* Invalid compression code         */
#define MrmBAD_ARG_TYPE        46 /* Invalid type, not in RGMrType... */
#define MrmNOT_IMP             48 /* Not yet implemented              */
#define MrmNULL_INDEX          50 /* empty index string               */
#define MrmBAD_KEY_TYPE        52 /* key must be MrmrIndex or MrmrRID */
#define MrmBAD_CALLBACK        54 /* Invalid callback descriptor      */
#define MrmNULL_ROUTINE        56 /* Empty callback routine name      */
                                 /* string                           */
#define MrmVEC_TOO_BIG         58 /* too many elements in vector      */
#define MrmBAD_HIERARCHY       60 /* invalid Mrm file hierarchy       */
```

```
#define MrmBAD_CLASS_CODE     62 /* Class code not found in Mrmwc... */
#define MrmDISPLAY_NOT_OPENED 63 /* Display not yet created          */
#define MrmEOF                64 /* End of file                      */
#define MrmUNRESOLVED_REFS    65 /* Unresolved widget ref in callback*/

#define MrmCR_CREATE        XmCR_CREATE

/* Code for unknown (user-defined) classes */

#define    MrmwcUnknown        1

/* The data types of values stored in uid files */

#define MrmRtypeMin            1
#define MrmRtypeInteger        1 /* int                              */
#define MrmRtypeBoolean        2
#define MrmRtypeChar8          3 /* a nul-terminated string          */
#define MrmRtypeChar8Vector    4 /* a vector of char_8 strings       */
#define MrmRtypeCString        5 /* a compound string (DDIS)         */
#define MrmRtypeCStringVector  6 /* a vector of compound strings     */
#define MrmRtypeFloat          7
                                 /* 8 = TypeCompressed now unused    */
#define MrmRtypeCallback       9 /* code for a callback descriptor   */
#define MrmRtypePixmapImage   10 /* Pixmap in image form             */
#define MrmRtypePixmapDDIF    11 /* Pixmap in DDIF form              */
#define MrmRtypeResource      12 /* Mrm resource descriptor          */
#define MrmRtypeNull          13 /* no value given                   */
#define MrmRtypeAddrName      14 /* nul-terminated string to be      */
                                 /* interpreted as runtime address   */
#define MrmRtypeIconImage     15 /* icon image                       */
#define MrmRtypeFont          16 /* Mrm font structure               */
#define MrmRtypeFontList      17 /* Mrm font list                    */
#define MrmRtypeColor         18 /* Mrm color descriptor             */
#define MrmRtypeColorTable    19 /* Mrm color table                  */
#define MrmRtypeAny           20 /* Any is allowed in UID file       */
#define MrmRtypeTransTable    21 /* Translation table (ASCIZ string) */
#define MrmRtypeClassRecName  22 /* class record name (ASCIZ string) */
#define MrmRtypeIntegerVector 23 /* a vector of integers             */
#define MrmRtypeXBitmapFile   24 /* X bitmap file to make pixmap     */
#define MrmRtypeCountedVector 25 /* vector with associated count     */
#define MrmRtypeKeysym        26 /* X keysym data type               */
#define MrmRtypeSingleFloat   27 /* single float data type           */
#define MrmRtypeWideCharacter 28 /* wide_character string type       */
#define MrmRtypeFontSet       29
#define MrmRtypeMax           30

#define MrmMaxResourceSize    65535      /* (2)16 - 1 */

/* MRM typedefs */

typedef short int           MrmCode ;    /* Used for codes,      */
                                         /* e.g. Mrmcr...        */
```

```
typedef unsigned char       MrmSCode ;   /* Short code for small */
                                         /* ranges               */
typedef unsigned short int  MrmOffset ;  /* Used for offsets in  */
                                         /* records              */
typedef short int           MrmType ;    /* Used for types,      */
                                         /* e.g. MrmrType.       */
typedef unsigned short int  MrmSize ;    /* For size fields      */
typedef short int           MrmCount ;   /* For counter fields   */
typedef unsigned char       MrmFlag ;    /* flag fields          */

/* The opaque result of opening a Mrm hierarchy */

typedef struct MrmHierarchyDescStruct *MrmHierarchy;

/* Structure used to pass name/value pairs to MrmRegisterNames   */

typedef struct {
    String      name ;     /* case-sensitive name                */
    XtPointer   value ;    /* value/address associated with name */
} MRMRegisterArg, MrmRegisterArg, *MrmRegisterArglist ;

/* Code for unknown (user-defined) classes */

#define    URMwcUnknown    1

#endif /* MrmPublic */
```

A.9 <Xm/MwmUtil.h>

```
#ifndef _XmMwmUtil_h
#define _XmMwmUtil_h

typedef struct
{
    long    flags;
    long    functions;
    long    decorations;
    int     input_mode;
    long    status;
} MotifWmHints;

typedef MotifWmHints   MwmHints;

/* bit definitions for MwmHints.flags */
#define MWM_HINTS_FUNCTIONS    (1L << 0)
#define MWM_HINTS_DECORATIONS  (1L << 1)
#define MWM_HINTS_INPUT_MODE   (1L << 2)
#define MWM_HINTS_STATUS       (1L << 3)

/* bit definitions for MwmHints.functions */
#define MWM_FUNC_ALL           (1L << 0)
#define MWM_FUNC_RESIZE        (1L << 1)
#define MWM_FUNC_MOVE          (1L << 2)
#define MWM_FUNC_MINIMIZE      (1L << 3)
#define MWM_FUNC_MAXIMIZE      (1L << 4)
#define MWM_FUNC_CLOSE         (1L << 5)

/* bit definitions for MwmHints.decorations */
#define MWM_DECOR_ALL          (1L << 0)
#define MWM_DECOR_BORDER       (1L << 1)
#define MWM_DECOR_RESIZEH      (1L << 2)
#define MWM_DECOR_TITLE        (1L << 3)
#define MWM_DECOR_MENU         (1L << 4)
#define MWM_DECOR_MINIMIZE     (1L << 5)
#define MWM_DECOR_MAXIMIZE     (1L << 6)

/* values for MwmHints.input_mode */
#define MWM_INPUT_MODELESS                     0
#define MWM_INPUT_PRIMARY_APPLICATION_MODAL    1
#define MWM_INPUT_SYSTEM_MODAL                 2
#define MWM_INPUT_FULL_APPLICATION_MODAL       3

/* bit definitions for MwmHints.status */
#define MWM_TEAROFF_WINDOW     (1L << 0)

typedef struct
{
    long      flags;
    Window    wm_window;
} MotifWmInfo;
```

```
typedef MotifWmInfo    MwmInfo;

/* bit definitions for MotifWmInfo flags */
#define MWM_INFO_STARTUP_STANDARD    (1L << 0)
#define MWM_INFO_STARTUP_CUSTOM      (1L << 1)

typedef struct
{
    CARD32    flags;
    CARD32    functions;
    CARD32    decorations;
    INT32     inputMode;
    CARD32    status;
} PropMotifWmHints;

typedef PropMotifWmHints PropMwmHints;

#define PROP_MOTIF_WM_HINTS_ELEMENTS    5

typedef struct
{
    CARD32    flags;
    CARD32    wmWindow;
} PropMotifWmInfo;

typedef PropMotifWmInfo    PropMwmInfo;

#define PROP_MOTIF_WM_INFO_ELEMENTS    2

#endif /* _XmMwmUtil_h */
```

A.10 <Xm/RepType.h>

```
define XmREP_TYPE_INVALID     0x1FFF

typedef unsigned short XmRepTypeId ;

 This file contains also *
typedef struct
{
    String rep_type_name ;
    String *value_names ;
    unsigned char *values ;
    unsigned char num_values ;
    Boolean reverse_installed ;
    XmRepTypeId rep_type_id ;
    } XmRepTypeEntryRec, *XmRepTypeEntry, XmRepTypeListRec,
     *XmRepTypeList;

/* However this data type is *already* specified in the man page */
/* for function XmRepTypeGetRecord()                             */
```

A.11 <Xm/Text.h>

```
#ifndef _XmText_h
#define _XmText_h

typedef struct _XmTextSourceRec *XmTextSource;

#endif /* _XmText_h */
```

A.12 <Xm/VendorS.h>

```
/* This file is a place holder for vendor specific implementation */
/* features. Hence, the file is required to be present in any     */
/* implementation but its content is implementation defined.      */
```

A.13 <Xm/VirtKeys.h>

```
#ifndef _XmVirtKeys_h
#define _XmVirtKeys_h

extern void XmTranslateKey(
                        Display     *dpy,
                        KeyCode      keycode,
                        Modifiers    modifiers,
                        Modifiers   *modifiers_return,
                        KeySym      *keysym_return) ;

#endif /* _XmVirtKeys_h */
```

A.14 <Xm/Xm.h>

```
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <X11/Xatom.h>

#include <Xm/VirtKeys.h>

#define XmVERSION                1
#define XmREVISION               2
#define XmUPDATE_LEVEL           0
#define XmVersion (XmVERSION * 1000 + XmREVISION)
#define XmVERSION_STRING "@(#)OSF/Motif Version 1.2.0"

extern int xmUseVersion;

#define XmUNSPECIFIED_PIXMAP     2

typedef enum{ XmFONT_IS_FONT, XmFONT_IS_FONTSET } XmFontType;

enum{    XmSTRING_DIRECTION_L_TO_R,      XmSTRING_DIRECTION_R_TO_L
    } ;

#define XmSTRING_DIRECTION_DEFAULT       ((XmStringDirection) 255)

typedef unsigned char  *XmString;              /* opaque to outside */
typedef XmString       *XmStringTable;         /* opaque to outside */
typedef char           *XmStringCharSet;       /* Null term string  */
typedef unsigned char   XmStringComponentType; /* component tags    */
typedef unsigned char   XmStringDirection;

enum{    XmSTRING_COMPONENT_UNKNOWN,    XmSTRING_COMPONENT_CHARSET,
    XmSTRING_COMPONENT_TEXT,    XmSTRING_COMPONENT_DIRECTION,
        XmSTRING_COMPONENT_SEPARATOR,   XmSTRING_COMPONENT_LOCALE_TEXT
    /* 6-125 reserved */
    } ;

#define XmSTRING_COMPONENT_END           ((XmStringComponentType) 126)

#define XmSTRING_COMPONENT_USER_BEGIN    ((XmStringComponentType) 128)
            /* 128-255 are user tags */
#define XmSTRING_COMPONENT_USER_END      ((XmStringComponentType) 255)

/* Base widget class and record definitions. Included are   */
/* the definitions for XmPrimitive, XmManager and XmGadget. */

/*  Primitive widget class and record definitions  */

typedef struct _XmPrimitiveClassRec     *XmPrimitiveWidgetClass;
typedef struct _XmPrimitiveRec          *XmPrimitiveWidget;

/*  Gadget widget class and record definitions     */
```

```
typedef struct _XmGadgetClassRec        *XmGadgetClass;
typedef struct _XmGadgetRec             *XmGadget;

/*  Manager widget class and record definitions    */

typedef struct _XmManagerClassRec       *XmManagerWidgetClass;
typedef struct _XmManagerRec            *XmManagerWidget;

/* Primitive Resources and define values           */

/* size policy values  */

enum{    XmCHANGE_ALL,            XmCHANGE_NONE,
    XmCHANGE_WIDTH,               XmCHANGE_HEIGHT
    } ;

/*  unit type values  */

enum{    XmPIXELS,                XmlOOTH_MILLIMETERS,
    Xm1000TH_INCHES,              Xm100TH_POINTS,
    Xm100TH_FONT_UNITS
    } ;

/* DeleteResponse values */

enum{    XmDESTROY,               XmUNMAP,
    XmDO_NOTHING
    } ;

enum{    XmEXPLICIT,              XmPOINTER
    } ;

/*  Navigation defines */

enum{    XmNONE,                  XmTAB_GROUP,
    XmSTICKY_TAB_GROUP,           XmEXCLUSIVE_TAB_GROUP
    } ;

#define XmDYNAMIC_DEFAULT_TAB_GROUP   ((XmNavigationType) 255)

/* Audible warning */

enum{    /* XmNONE */             XmBELL = 1
    } ;

/*  Menu defines */

enum{    XmNO_ORIENTATION,        XmVERTICAL,
    XmHORIZONTAL
    } ;
enum{    XmWORK_AREA,             XmMENU_BAR,
    XmMENU_PULLDOWN,              XmMENU_POPUP,
    XmMENU_OPTION
    } ;
enum{    XmNO_PACKING,            XmPACK_TIGHT,
    XmPACK_COLUMN,                XmPACK_NONE
    } ;
enum{/* XmALIGNMENT_BASELINE_TOP,    XmALIGNMENT_CENTER,
    XmALIGNMENT_BASELINE_BOTTOM, */  XmALIGNMENT_CONTENTS_TOP = 3,
    XmALIGNMENT_CONTENTS_BOTTOM
    } ;
enum{    XmTEAR_OFF_ENABLED,      XmTEAR_OFF_DISABLED
    } ;
enum{    XmUNPOST,                XmUNPOST_AND_REPLAY
    } ;
enum{   XmLAST_POSITION = -1,     XmFIRST_POSITION
    } ;

/*  Label/Frame defines */

enum{    XmALIGNMENT_BEGINNING,   XmALIGNMENT_CENTER,
    XmALIGNMENT_END
    } ;
enum{   XmALIGNMENT_BASELINE_TOP,    /* XmALIGNMENT_CENTER, */
    XmALIGNMENT_BASELINE_BOTTOM = 2, XmALIGNMENT_WIDGET_TOP,
    XmALIGNMENT_WIDGET_BOTTOM
       } ;

/*  Frame defines */

enum{   XmFRAME_GENERIC_CHILD,    XmFRAME_WORKAREA_CHILD,
        XmFRAME_TITLE_CHILD
    } ;

/*  ToggleButton defines */

enum{    XmN_OF_MANY = 1,         XmONE_OF_MANY
    } ;

/* Form defines */

enum{    XmATTACH_NONE,           XmATTACH_FORM,
    XmATTACH_OPPOSITE_FORM,       XmATTACH_WIDGET,
    XmATTACH_OPPOSITE_WIDGET,     XmATTACH_POSITION,
    XmATTACH_SELF
    } ;
enum{    XmRESIZE_NONE,           XmRESIZE_GROW,
    XmRESIZE_ANY
    } ;

/*  Callback reasons */

enum{    XmCR_NONE,               XmCR_HELP,
    XmCR_VALUE_CHANGED,           XmCR_INCREMENT,
    XmCR_DECREMENT,               XmCR_PAGE_INCREMENT,
    XmCR_PAGE_DECREMENT,          XmCR_TO_TOP,
    XmCR_TO_BOTTOM,               XmCR_DRAG,
    XmCR_ACTIVATE,                XmCR_ARM,
    XmCR_DISARM,                  XmCR_MAP = 16,
    XmCR_UNMAP,                   XmCR_FOCUS,
    XmCR_LOSING_FOCUS,            XmCR_MODIFYING_TEXT_VALUE,
    XmCR_MOVING_INSERT_CURSOR,    XmCR_EXECUTE,
    XmCR_SINGLE_SELECT,           XmCR_MULTIPLE_SELECT,
    XmCR_EXTENDED_SELECT,         XmCR_BROWSE_SELECT,
    XmCR_DEFAULT_ACTION,          XmCR_CLIPBOARD_DATA_REQUEST,
    XmCR_CLIPBOARD_DATA_DELETE,   XmCR_CASCADING,
    XmCR_OK,                      XmCR_CANCEL,
    XmCR_APPLY = 34,              XmCR_NO_MATCH,
    XmCR_COMMAND_ENTERED,         XmCR_COMMAND_CHANGED,
    XmCR_EXPOSE,                  XmCR_RESIZE,
    XmCR_INPUT,                   XmCR_GAIN_PRIMARY,
    XmCR_LOSE_PRIMARY,            XmCR_CREATE,
    XmCR_TEAR_OFF_ACTIVATE,       XmCR_TEAR_OFF_DEACTIVATE,
    XmCR_OBSCURED_TRAVERSAL
    } ;

/* Callback structures */

typedef struct
{
    int      reason;
    XEvent  *event;
} XmAnyCallbackStruct;

typedef struct
{
    int      reason;
    XEvent  *event;
    int        click_count;
} XmArrowButtonCallbackStruct;

typedef struct
{
    int      reason;
    XEvent  *event;
    Window   window;
} XmDrawingAreaCallbackStruct;

typedef struct
{
    int      reason;
    XEvent  *event;
    Window   window;
    int      click_count;
} XmDrawnButtonCallbackStruct;
```

```
typedef struct
{
    int      reason;
    XEvent  *event;
    int      click_count;
} XmPushButtonCallbackStruct;

typedef struct
{
    int      reason;
    XEvent  *event;
    Widget   widget;
    char    *data;
    char    *callbackstruct;
} XmRowColumnCallbackStruct;

typedef struct
{
    int      reason;
    XEvent  *event;
    int      value;
    int      pixel;
} XmScrollBarCallbackStruct;

typedef struct
{
    int      reason;
    XEvent  *event;
    int      set;
} XmToggleButtonCallbackStruct;

typedef struct
{
    int       reason;
    XEvent   *event;
    XmString  item;
    int       item_length;
    int       item_position;
    XmString *selected_items;
    int       selected_item_count;
    int      *selected_item_positions;
    char      selection_type;
} XmListCallbackStruct;

typedef struct
{
    int       reason;
    XEvent   *event;
    XmString  value;
    int       length;
} XmSelectionBoxCallbackStruct;
```

```
typedef struct
{
    int reason;
    XEvent   *event;
    XmString  value;
    int       length;
} XmCommandCallbackStruct;

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;

typedef struct
{
    int       reason;
    XEvent   *event;
    int       value;
} XmScaleCallbackStruct;

/*  PushButton defines */

enum{    XmMULTICLICK_DISCARD,   XmMULTICLICK_KEEP
    } ;

/*  DrawnButton defines */

enum{    XmSHADOW_IN = 7,        XmSHADOW_OUT
    } ;

/*  Arrow defines */

enum{    XmARROW_UP,             XmARROW_DOWN,
    XmARROW_LEFT,                XmARROW_RIGHT
    } ;

/*  Separator defines                                              */
/*  Note: XmINVALID_SEPARATOR_TYPE marks the last+1 separator type */

enum{    XmNO_LINE,              XmSINGLE_LINE,
    XmDOUBLE_LINE,               XmSINGLE_DASHED_LINE,
    XmDOUBLE_DASHED_LINE,        XmSHADOW_ETCHED_IN,
    XmSHADOW_ETCHED_OUT,         XmSHADOW_ETCHED_IN_DASH,
    XmSHADOW_ETCHED_OUT_DASH,    XmINVALID_SEPARATOR_TYPE
    } ;

enum{    XmPIXMAP = 1,           XmSTRING
    } ;

/*  Drag and Drop #defines */

enum{    XmWINDOW,               /* XmPIXMAP, */
    XmCURSOR = 2
    } ;

/*  ScrollBar #defines */

enum{    XmMAX_ON_TOP,           XmMAX_ON_BOTTOM,
    XmMAX_ON_LEFT,               XmMAX_ON_RIGHT
    } ;

/* List Widget defines */

enum{    XmSINGLE_SELECT,        XmMULTIPLE_SELECT,
    XmEXTENDED_SELECT,           XmBROWSE_SELECT
    } ;
enum{    XmSTATIC,               XmDYNAMIC
    } ;

/* Scrolled Window defines */

enum{    XmVARIABLE,             XmCONSTANT,
    XmRESIZE_IF_POSSIBLE
    } ;
enum{    XmAUTOMATIC,            XmAPPLICATION_DEFINED
    } ;
enum{    /* XmSTATIC */          XmAS_NEEDED = 1
    } ;

#define SW_TOP        1
#define SW_BOTTOM     0
#define SW_LEFT       2
#define SW_RIGHT      0

#define XmTOP_LEFT       (SW_TOP | SW_LEFT)
#define XmBOTTOM_LEFT    (SW_BOTTOM  | SW_LEFT)
#define XmTOP_RIGHT      (SW_TOP | SW_RIGHT)
#define XmBOTTOM_RIGHT   (SW_BOTTOM  | SW_RIGHT)

/* MainWindow Resources */

enum{    XmCOMMAND_ABOVE_WORKSPACE,    XmCOMMAND_BELOW_WORKSPACE
    } ;
```

```
/* Text Widget defines */

enum{    XmMULTI_LINE_EDIT,    XmSINGLE_LINE_EDIT
    } ;

typedef enum{
    XmTEXT_FORWARD,
    XmTEXT_BACKWARD
    } XmTextDirection;

typedef long XmTextPosition;
typedef Atom XmTextFormat;

typedef enum{
    XmSELECT_POSITION,          XmSELECT_WHITESPACE,
    XmSELECT_WORD,              XmSELECT_LINE,
    XmSELECT_ALL,               XmSELECT_PARAGRAPH
    } XmTextScanType ;

typedef enum{
    XmHIGHLIGHT_NORMAL,         XmHIGHLIGHT_SELECTED,
    XmHIGHLIGHT_SECONDARY_SELECTED
    } XmHighlightMode ;

/* XmTextBlock's are used to pass text around. */

typedef struct {
    char *ptr;                  /* Pointer to data          */
    int length;                 /* Number of bytes of data */
    XmTextFormat format;        /* Representations format  */
} XmTextBlockRec, *XmTextBlock;

typedef struct
{
    int          reason;
    XEvent      *event;
    Boolean      doit;
    long         currInsert, newInsert;
    long         startPos, endPos;
    XmTextBlock  text;
} XmTextVerifyCallbackStruct, *XmTextVerifyPtr;

/* XmTextBlockWcs's are used in 1.2 modifyVerifyWcs callbacks */
/* for Text[Field] * widgets.                                 */

typedef struct {
    wchar_t     *wcsptr;        /* Pointer to data.            */
    int          length;        /* Number of bytes of data.   */
} XmTextBlockRecWcs, *XmTextBlockWcs;
```

```
typedef struct
{
    int              reason;
    XEvent          *event;
    Boolean          doit;
    long             currInsert, newInsert;
    long             startPos, endPos;
    XmTextBlockWcs   text;
} XmTextVerifyCallbackStructWcs, *XmTextVerifyPtrWcs;

/* functions renamed after 1.0 release due to resource name overlap */
#define XmTextGetTopPosition    XmTextGetTopCharacter
#define XmTextSetTopPosition    XmTextSetTopCharacter

#define XmCOPY_FAILED        0
#define XmCOPY_SUCCEEDED     1
#define XmCOPY_TRUNCATED     2

/* DIALOG defines..  BulletinBoard and things common to its    */
/* subclasses CommandBox, MessageBox, Selection, FileSelection */

/* child type defines for Xm...GetChild() */

enum{    XmDIALOG_NONE,              XmDIALOG_APPLY_BUTTON,
    XmDIALOG_CANCEL_BUTTON,         XmDIALOG_DEFAULT_BUTTON,
    XmDIALOG_OK_BUTTON,             XmDIALOG_FILTER_LABEL,
    XmDIALOG_FILTER_TEXT,           XmDIALOG_HELP_BUTTON,
    XmDIALOG_LIST,                  XmDIALOG_LIST_LABEL,
    XmDIALOG_MESSAGE_LABEL,         XmDIALOG_SELECTION_LABEL,
    XmDIALOG_SYMBOL_LABEL,          XmDIALOG_TEXT,
    XmDIALOG_SEPARATOR,             XmDIALOG_DIR_LIST,
    XmDIALOG_DIR_LIST_LABEL
    } ;

#define XmDIALOG_HISTORY_LIST       XmDIALOG_LIST
#define XmDIALOG_PROMPT_LABEL       XmDIALOG_SELECTION_LABEL
#define XmDIALOG_VALUE_TEXT         XmDIALOG_TEXT
#define XmDIALOG_COMMAND_TEXT       XmDIALOG_TEXT
#define XmDIALOG_FILE_LIST          XmDIALOG_LIST
#define XmDIALOG_FILE_LIST_LABEL    XmDIALOG_LIST_LABEL

/*  dialog style defines  */

enum{    XmDIALOG_MODELESS,          XmDIALOG_PRIMARY_APPLICATION_MODAL,
    XmDIALOG_FULL_APPLICATION_MODAL,XmDIALOG_SYSTEM_MODAL
    } ;

/* The following is for compatibility only. Its use is deprecated. */

#define XmDIALOG_APPLICATION_MODAL  XmDIALOG_PRIMARY_APPLICATION_MODAL

/* XmSelectionBox, XmFileSelectionBox and XmCommand - misc. stuff  */
/* Defines for Selection child placement 8?

enum{    XmPLACE_TOP,                XmPLACE_ABOVE_SELECTION,
    XmPLACE_BELOW_SELECTION
    } ;

/* Defines for file type mask: */

#define XmFILE_DIRECTORY (1 << 0)
#define XmFILE_REGULAR   (1 << 1)
#define XmFILE_ANY_TYPE  (XmFILE_DIRECTORY | XmFILE_REGULAR)

/* Defines for selection dialog type: */

enum{    XmDIALOG_WORK_AREA,         XmDIALOG_PROMPT,
    XmDIALOG_SELECTION,              XmDIALOG_COMMAND,
    XmDIALOG_FILE_SELECTION
    } ;

/*  XmMessageBox things not common to other dialogs */

/* defines for dialog type */

enum{    XmDIALOG_TEMPLATE,          XmDIALOG_ERROR,
    XmDIALOG_INFORMATION,            XmDIALOG_MESSAGE,
    XmDIALOG_QUESTION,               XmDIALOG_WARNING,
    XmDIALOG_WORKING
    } ;

/*  Traversal types  */

typedef enum{
    XmVISIBILITY_UNOBSCURED,         XmVISIBILITY_PARTIALLY_OBSCURED,
    XmVISIBILITY_FULLY_OBSCURED
    } XmVisibility ;

typedef enum{
    XmTRAVERSE_CURRENT,              XmTRAVERSE_NEXT,
    XmTRAVERSE_PREV,                 XmTRAVERSE_HOME,
    XmTRAVERSE_NEXT_TAB_GROUP,       XmTRAVERSE_PREV_TAB_GROUP,
    XmTRAVERSE_UP,                   XmTRAVERSE_DOWN,
    XmTRAVERSE_LEFT,                 XmTRAVERSE_RIGHT
    } XmTraversalDirection ;

typedef struct _XmTraverseObscuredCallbackStruct
{    int                  reason ;
    XEvent               *event ;
    Widget                traversal_destination ;
    XmTraversalDirection   direction ;
    } XmTraverseObscuredCallbackStruct ;

typedef unsigned char                XmNavigationType;
```

```
/* SimpleMenu declarations and definitions. */

typedef unsigned char               XmButtonType;
typedef XmButtonType               *XmButtonTypeTable;
typedef KeySym                     *XmKeySymTable;
typedef XmStringCharSet            *XmStringCharSetTable;

enum{    XmPUSHBUTTON = 1,          XmTOGGLEBUTTON,
    XmRADIOBUTTON,                  XmCASCADEBUTTON,
    XmSEPARATOR,                    XmDOUBLE_SEPARATOR,
    XmTITLE
    } ;
#define XmCHECKBUTTON               XmTOGGLEBUTTON

typedef XtPointer (*XmResourceBaseProc)( Widget, XtPointer) ;

typedef struct _XmSecondaryResourceDataRec{
    XmResourceBaseProc  base_proc;
    XtPointer           client_data;
    String              name;
    String              res_class;
    XtResourceList      resources;
    Cardinal            num_resources;
} XmSecondaryResourceDataRec, *XmSecondaryResourceData;

extern Cardinal XmGetSecondaryResourceData(
                        WidgetClass w_class,
                        XmSecondaryResourceData **secondaryDataRtn) ;

extern Boolean XmInstallImage(
                        XImage *image,
                        char *image_name) ;
extern Boolean XmUninstallImage(
                        XImage *image) ;
extern Pixmap XmGetPixmap(
                        Screen *screen,
                        char *image_name,
                        Pixel foreground,
                        Pixel background) ;
extern Pixmap XmGetPixmapByDepth(
                        Screen *screen,
                        char *image_name,
                        Pixel foreground,
                        Pixel background,
                        int depth) ;
extern Boolean XmDestroyPixmap(
                        Screen *screen,
                        Pixmap pixmap) ;
extern void XmUpdateDisplay(
                        Widget w) ;

/**    Primitive.c    **/
```

```
typedef long XmOffset;
typedef XmOffset *XmOffsetPtr;

extern void XmResolvePartOffsets(
                        WidgetClass w_class,
                        XmOffsetPtr *offset) ;
extern void XmResolveAllPartOffsets(
                        WidgetClass w_class,
                        XmOffsetPtr *offset,
                        XmOffsetPtr *constraint_offset) ;
extern Boolean XmWidgetGetBaselines(
                        Widget wid,
                        Dimension **baselines,
                        int *line_count);
extern Boolean XmWidgetGetDisplayRect(
                        Widget wid,
                        XRectangle *displayrect);

/**    Public Function Declarations for ResConvert.c    **/

extern void XmRegisterConverters( void ) ;
extern void XmCvtStringToUnitType(
                        XrmValuePtr args,
                        Cardinal *num_args,
                        XrmValue *from_val,
                        XrmValue *to_val) ;
extern char * XmRegisterSegmentEncoding(
                        char *fontlist_tag,
                        char *ct_encoding) ;
extern char * XmMapSegmentEncoding(
                        char *fontlist_tag) ;
extern XmString XmCvtCTToXmString(
                        char *text) ;
extern Boolean XmCvtTextToXmString(
                        Display *display,
                        XrmValuePtr args,
                        Cardinal *num_args,
                        XrmValue *from_val,
                        XrmValue *to_val,
                        XtPointer *converter_data) ;
extern char * XmCvtXmStringToCT(
                        XmString string) ;
extern Boolean XmCvtXmStringToText(
                        Display *display,
                        XrmValuePtr args,
                        Cardinal *num_args,
                        XrmValue *from_val,
                        XrmValue *to_val,
                        XtPointer *converter_data) ;

/**    Public Function Declarations for ResInd.c    **/
```

```
extern int XmConvertUnits(
                        Widget widget,
                        int dimension,
                        register int from_type,
                        register int from_val,
                        register int to_type) ;
extern int XmCvtToHorizontalPixels(
                        Screen *screen,
                        register int from_val,
                        register int from_type) ;
extern int XmCvtToVerticalPixels(
                        Screen *screen,
                        register int from_val,
                        register int from_type) ;
extern int XmCvtFromHorizontalPixels(
                        Screen *screen,
                        register int from_val,
                        register int to_type) ;
extern int XmCvtFromVerticalPixels(
                        Screen *screen,
                        register int from_val,
                        register int to_type) ;
extern void XmSetFontUnits(
                        Display *display,
                        int h_value,
                        int v_value) ;
extern void XmSetFontUnit(
                        Display *display,
                        int value) ;

/**    Public Function Declarations for MenuUtil.c    **/

extern void XmSetMenuCursor(
                        Display *display,
                        Cursor cursorId) ;
extern Cursor XmGetMenuCursor(
                        Display *display) ;

/**    Public Function Declarations for Simple.c    **/

extern Widget XmCreateSimpleMenuBar(
                        Widget parent,
                        String name,
                        ArgList args,
                        Cardinal arg_count) ;
extern Widget XmCreateSimplePopupMenu(
                        Widget parent,
                        String name,
                        ArgList args,
                        Cardinal arg_count) ;
extern Widget XmCreateSimplePulldownMenu(
                        Widget parent,
                        String name,
                        ArgList args,
                        Cardinal arg_count) ;
extern Widget XmCreateSimpleOptionMenu(
                        Widget parent,
                        String name,
                        ArgList args,
                        Cardinal arg_count) ;
extern Widget XmCreateSimpleRadioBox(
                        Widget parent,
                        String name,
                        ArgList args,
                        Cardinal arg_count) ;
extern Widget XmCreateSimpleCheckBox(
                        Widget parent,
                        String name,
                        ArgList args,
                        Cardinal arg_count) ;

/**    Public Function Declarations for VaSimple.c   **/

extern Widget XmVaCreateSimpleMenuBar(
            Widget parent,
            String name,
            ...) ;
extern Widget XmVaCreateSimplePopupMenu(
            Widget parent,
            String name,
            XtCallbackProc callback,
            ...) ;
extern Widget XmVaCreateSimplePulldownMenu(
            Widget parent,
            String name,
            int post_from_button,
            XtCallbackProc callback,
            ...) ;
extern Widget XmVaCreateSimpleOptionMenu(
            Widget parent,
            String name,
                        XmString option_label,
                        KeySym option_mnemonic,
                        int button_set,
                        XtCallbackProc callback,
            ...) ;
extern Widget XmVaCreateSimpleRadioBox(
            Widget parent,
            String name,
            int button_set,
            XtCallbackProc callback,
            ...) ;
extern Widget XmVaCreateSimpleCheckBox(
            Widget parent,
            String name,
            XtCallbackProc callback,
            ...) ;

/**    Public Function Declarations for TrackLoc.c    **/

extern Widget XmTrackingLocate(
                        Widget widget,
                        Cursor cursor,
                        Boolean confineTo) ;

/**    Visual.c    **/

typedef void (*XmColorProc) (XColor *bg_color, XColor *fg_color,
    XColor *sel_color, XColor *ts_color, XColor *bs_color);

extern XmColorProc XmSetColorCalculation(
                        XmColorProc proc) ;
extern XmColorProc XmGetColorCalculation( void ) ;
extern void XmGetColors(
                        Screen *screen,
                        Colormap color_map,
                        Pixel background,
                        Pixel *foreground_ret,
                        Pixel *top_shadow_ret,
                        Pixel *bottom_shadow_ret,
                        Pixel *select_ret) ;
extern void XmChangeColor(
                        Widget widget,
                        Pixel background) ;

/**    Public Function Declarations for XmString.c    **/

extern XmString XmStringCreate(
                        char *text,
                        XmStringCharSet charset) ;
extern XmString XmStringCreateSimple(
                        char *text) ;
extern XmString XmStringCreateLocalized(
                        String text) ;
extern XmString XmStringDirectionCreate(
                        XmStringDirection direction) ;

extern XmString XmStringSeparatorCreate( void ) ;
extern XmString XmStringSegmentCreate(
                        char *text,
                        XmStringCharSet charset,
                        XmStringDirection direction,
                        Boolean separator) ;

extern XmString XmStringLtoRCreate(
                        char *text,
                        XmStringCharSet charset) ;
extern XmString XmStringCreateLtoR(
                        char *text,
                        XmStringCharSet charset) ;
extern Boolean XmStringInitContext(
                        XmStringContext *context,
                        XmString string) ;
extern void XmStringFreeContext(
                        XmStringContext context) ;
extern XmStringComponentType XmStringGetNextComponent(
                        XmStringContext context,
                        char **text,
                        XmStringCharSet *charset,
                        XmStringDirection *direction,
                        XmStringComponentType *unknown_tag,
                        unsigned short *unknown_length,
                        unsigned char **unknown_value) ;
extern XmStringComponentType XmStringPeekNextComponent(
                        XmStringContext context) ;
extern Boolean XmStringGetNextSegment(
                        XmStringContext context,
                        char **text,
                        XmStringCharSet *charset,
                        XmStringDirection *direction,
                        Boolean *separator) ;
extern Boolean XmStringGetLtoR(
                        XmString string,
                        XmStringCharSet charset,
                        char **text) ;
extern XmFontListEntry XmFontListEntryCreate(
                        char *tag,
                        XmFontType type,
                        XtPointer font) ;
extern void XmFontListEntryFree(
                        XmFontListEntry *entry) ;
extern XtPointer XmFontListEntryGetFont(
                        XmFontListEntry entry,
                        XmFontType *typeReturn) ;
extern char * XmFontListEntryGetTag(
                        XmFontListEntry entry) ;
extern XmFontList XmFontListAppendEntry(
                        XmFontList old,
                        XmFontListEntry entry) ;
extern XmFontListEntry XmFontListNextEntry(
                        XmFontContext context) ;
extern XmFontList XmFontListRemoveEntry(
                        XmFontList old,
                        XmFontListEntry entry) ;
extern XmFontListEntry XmFontListEntryLoad(
                        Display *display,
                        char *fontName,
                        XmFontType type,
                        char *tag) ;
extern XmFontList XmFontListCreate(
                        XFontStruct *font,
                        XmStringCharSet charset) ;
extern XmFontList XmStringCreateFontList(
                        XFontStruct *font,
                        XmStringCharSet charset) ;
extern void XmFontListFree(
                        XmFontList fontlist) ;
extern XmFontList XmFontListAdd(
                        XmFontList old,
                        XFontStruct *font,
                        XmStringCharSet charset) ;
extern XmFontList XmFontListCopy(
                        XmFontList fontlist) ;
extern Boolean XmFontListInitFontContext(
                        XmFontContext *context,
                        XmFontList fontlist) ;
extern Boolean XmFontListGetNextFont(
                        XmFontContext context,
                        XmStringCharSet *charset,
                        XFontStruct **font) ;
extern void XmFontListFreeFontContext(
                        XmFontContext context) ;
extern XmString XmStringConcat(
                        XmString a,
                        XmString b) ;
extern XmString XmStringNConcat(
                        XmString first,
                        XmString second,
                        int n) ;
extern XmString XmStringCopy(
                        XmString string) ;
extern XmString XmStringNCopy(
                        XmString str,
                        int n) ;
extern Boolean XmStringByteCompare(
                        XmString a1,
                        XmString b1) ;
extern Boolean XmStringCompare(
                        XmString a,
                        XmString b) ;
extern int XmStringLength(
                        XmString string) ;
extern Boolean XmStringEmpty(
                        XmString string) ;
extern Boolean XmStringHasSubstring(
                        XmString string,
                        XmString substring) ;
extern void XmStringFree(
                        XmString string) ;
extern Dimension XmStringBaseline(
                        XmFontList fontlist,
                        XmString string) ;
extern Dimension XmStringWidth(
                        XmFontList fontlist,
                        XmString string) ;
extern Dimension XmStringHeight(
                        XmFontList fontlist,
                        XmString string) ;
extern void XmStringExtent(
                        XmFontList fontlist,
                        XmString string,
                        Dimension *width,
                        Dimension *height) ;
extern int XmStringLineCount(
                        XmString string) ;
extern void XmStringDraw(
                        Display *d,
                        Window w,
                        XmFontList fontlist,
                        XmString string,
                        GC gc,
                        Position x,
                        Position y,
                        Dimension width,
                        unsigned char align,
                        unsigned char lay_dir,
                        XRectangle *clip) ;
extern void XmStringDrawImage(
                        Display *d,
                        Window w,
                        XmFontList fontlist,
                        XmString string,
                        GC gc,
                        Position x,
                        Position y,
                        Dimension width,
                        unsigned char align,
                        unsigned char lay_dir,
                        XRectangle *clip) ;
extern void XmStringDrawUnderline(
                        Display *d,
                        Window w,
                        XmFontList fntlst,
                        XmString str,
                        GC gc,
                        Position x,
                        Position y,
                        Dimension width,
                        unsigned char align,
                        unsigned char lay_dir,
                        XRectangle *clip,
                        XmString under) ;
```

```
/**    Public Function Declarations for Dest.c    **/

extern Widget XmGetDestination(
                        Display *display) ;

/**    Public Function Declarations for Traversal.c    **/

extern Boolean XmIsTraversable(
                        Widget wid) ;
extern XmVisibility XmGetVisibility(
                        Widget wid) ;
extern Widget XmGetTabGroup(
                        Widget wid) ;
extern Widget XmGetFocusWidget(
                        Widget wid) ;
extern Boolean XmProcessTraversal(
                        Widget w,
                        XmTraversalDirection dir) ;
extern void XmAddTabGroup(
                        Widget tabGroup) ;
extern void XmRemoveTabGroup(
                        Widget w) ;
```

Glossary

atom
An atom is a unique ID corresponding to a string name. Atoms are used to identify properties, types and selections.

border
An *InputOutput* window can have a border of equal thickness on all four sides of the window. A pixmap defines the contents of the border, and the server automatically maintains the contents of the border. Exposure events are never generated for border regions.

class
A category into which objects are placed on the basis of both their purpose and their internal structure. A class lists the data relating to an instantiation of an object, and the methods that act on the data.

clipboard
A facility used to store text or graphics during cut-and-paste operations in a windowing environment.

command
In Motif, the mwm service type is command.

composite
In Motif, composite widgets are containers for other widgets, and can have an arbitrary number of children.

compound string
Compound string functions manipulate compound strings, including conversion to and from compound text.

compound text
Compound text is a format, based on ISO standards, for encoding data such as multi-lingual text, using multiple character sets. Compound Text provides for the internationalisation of certain named items, such as resources in Xlib and the Xt Intrinsics.

core
The name given to the Xt Intrinsic base class for windowed widgets.

drag and drop
A windows environment transfer function in which selected display information is moved from one position to another, using the mouse facilities.

GUI
(Graphical User Interface) A user interface technology employing visual techniques in addition to the display of text.

icon
In a windowing environment, a small graphic image used to represent a window. This is used to reduce the size a window occupies while still showing its availability on the screen.

look and feel
The commonly used expression describing how an application appears on the computer display (look) and how the user interacts with it (feel).

menu
In a windows environment, a list of available selections, from which the user chooses.

Motif
A graphical user interface (GUI) windowing system environment, that specifies how a user interface for graphical computers should *look and feel*. It was designed by the OSF (sic), to accommodate the requirements of system and application technologies from different sources. It is based on the X Window System, and the supporting X Toolkit facilitates production of Motif-compliant applications.

Mrm
Motif resource manager.

Mwm
Motif window manager.

OSF
The Open Software Foundation.

object
An object is an identifiable, encapsulated entity that provides one or more services that can be requested by a client.

pointer
In a windows environment, the pointer is the pointing device attached to the cursor which shows the current working position on the screen. Usually its position is controlled by a mouse, trackerball, joystick or other pointing device.

protocol
A specification for an agreed procedure to enable exchange of information between cooperating entities, by means of interfaces that provide the necessary capability to cover format of messages, data checks, flow control and error handling.

resource
Windows, pixmaps, cursors, fonts, graphics contexts and colormaps are known as resources. They all have unique identifiers associated with them for naming purposes. The lifetime of a resource usually is bounded by the lifetime of the connection over which the resource was created.

A resource is also a named piece of data in a widget that can be set by a client, by an application or by user defaults.

scroll
To move the representation of data vertically or horizontally relative to the terminal screen. There are two types of scrolling:

- The cursor moves with the data.
- The cursor remains stationary while the data moves.

scrollbar
In a windowing environment, the side border (for vertical movement) or bottom border (for horizontal movement) of a window. Pointing to and then clicking on the arrows in these borders causes corresponding movement inside the associated window. Pointing to and then moving the slider in a border similarly causes corresponding movement inside that window.

UID
In Motif, a user interface description output from the UIL compiler. UID files are not directly available to applications; they can only be accessed through functions from the Mrm library.

UIL
In Motif, the user interface language. Modules in the UIL are compiled with the UIL compiler, which generates **.uid** files. These files can be loaded at run time by the Mrm library.

widget
A reusable, configurable, user interface visual object. A widget combines an X window with associated input and display semantics. A widget is dynamically-allocated and contains state information. A Scrollbar is an example of a widget. The general group to which a specific widget belongs is the widget class. See also Motif.

window
A data structure that presents on the screen a graphical user interface for an application or utility. The user interacts with the display of the window, to select and perform operations using that window. Visually, a window may occupy all or part of the screen display area, depending on what resizing has been done on it. When not in active use it may also be *minimized*, in which case it is displayed as an icon.

window manager
Manipulation of windows on the screen, and much of the user interface (policy), is typically provided by a window manager client.

Xm
X/Open Motif.

Xt
X Toolkit.

Index