THE DEFINITIVE GUIDES TO THE X WINDOW SYSTEM

VOLUME SIX B

Motif Reference Manual

for Motif 2.1

Open Source Edition

Antony Fountain and Paula Ferguson

Motif Reference Manual, Open Source Edition

by Antony Fountain and Paula Ferguson

December 2001

Copyright © 1998, 2000, 2001 O'Reilly & Associates, Inc. and Antony Fountain. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at http://www.opencontent.org/openpub/).

This is a modified version of the Motif Reference Manual, Second Edition, published by O'Reilly & Associates in February 2000. The source files for the Second Edition can be found at http://www.oreilly.com/openbook/motif/. A description of the modifications is contained in the Preface to the Open Source Edition.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Contents

Preface

Preface to the Open Source Edition

Many thanks to all at O'Reilly and Associates for releasing this, Volume 6B, and the companion Volume 6A, the Motif Programming Manual, in open source. Both have been extensively revised for Motif 2.1; this, the Motif Reference Manual, has had several alterations to the 2nd edition as printed:

- all the function prototypes and examples have been converted to strict ANSI format
- the UIL sections have been restored
- the Xt Session Shell is documented
- many bug patches have been folded in
- new examples have been added to Motif 2.1 procedure sections
- the book sources have been converted from the original troff into FrameMaker and PDF formats

Removing the UIL portions from the original printed second edition was a hard decision; the Motif 2.1 toolkit was a much expanded library since previous versions of the book, and something had to give - the book was over a thousand pages as it was. However, an electronic copy does not have the same space restrictions as the printed tome, and so these materials, originally in the Motif 1.2 version of the manual, have been restored. They also have been reworked for Motif 2.1.

Antony J. Fountain

Preface to the Second Edition

What to put in, and what to leave out, of this update to the Motif Reference Manual was the hardest decision of all. The guiding principle has been to consider for whom this material is intended. This is a Programmer's Reference, and not a Widget Author's handbook. Accordingly, those aspects of the new Trait mechanisms which an application programmer needs to know have been included, but the Xme utilities have not. Specifying a Trait as a well-defined piece of behaviour which a widget supports, it is enough to know which traits a Widget Class supports, and how this affects objects in the widget instance hierarchy. How a Trait is implemented, and which methods are associated with the given Trait, are generally the domain of the widget author. Hence it is recorded that the VendorShell holds the XmQTspecifyRenderTableTrait, and that this means that widget classes further down the widget instance hierarchy inherit default Render Table information from the VendorShell. This is all that the Application Programmer needs to know: the rest is silence.

Conversely, the Motif Input Method utilities have been included. Although mostly defined originally in the Motif 1.2 release, and although the Motif widget classes generally handle connections to an Input Method when and where this is required, there is an important exception. The Motif Drawing Area does not register itself with an Input Method automatically, and hence anyone who needs to directly implement internationalized input for this widget class most certainly would need to know about the XmIm functions. The World does not all speak English: for these reasons, the XmIm functions are included in the Manual.

A brief note concerning the status of Motif as the premier Unix toolkit. A number of alternative toolkits have arisen, particularly in the Linux domain, which offer an X-based windowing system for the Unix, and other, platforms. I refer principally to the likes of Qt, and GTK+. These on the whole dispense with the Xt layer, in order to provide small, lightweight GUI components which are, from the application programmer's perspective, relatively easy to port to non-Unix domains. Although admirable in many ways, these suffer from one crucial drawback, precisely because Xt has been excluded: there is no object component model associated with any of the objects which can be created in an interface[1]. Compare and contrast with something like JavaBeans, where a GUI builder can be designed which can dynamically load and query objects from whatever source, and from thence inspect the attributes of the object, construct resource panels, and generate code for the components, all without any external configuration. Based on Xt, Motif also has this important property: I can in principle dynamically load into my GUI builder any third party component, construct an internal attribute list, present resource panels for object configuration to the user, and from there generate source code. Just by interrogating the widget class. All the commercial GUI builders available for Motif support this.

The newer alternative Linux toolkits do not have this introspective quality. Writing GUI builders happens to be what I do for a living: sad to say, I cannot write one for these toolkits precisely because these is no component model at the object level. Not surprisingly, no third party component market exists for the toolkits either: there is no GUI builder into which these components can be dynamically slotted. Each needs the other, but there is nothing which allows them to talk. In the absence of either a commercial component market, or a dynamic GUI builder, there remains serious question marks concerning the scalability of the alternative toolkits, whatever merits they hold. The only alternatives are to write all the code by hand, or pass control of the

1.True at the moment of writing. It is still true that all the information required to dynamically introspect an object's entire resource set, particularly if user-defined and not built-in to the basic set, is not completely forthcoming. Introspecting third party components remains troublesome for a dynamic GUI builder.

application to a private piece of hobbyware which masquerades as a support environment. Ironically, the advent of Java has cemented Motif: the JDK relies on Motif for the native implementation on the Unix platform. Until such time as a native toolkit surfaces which has this important introspective property, Motif remains what it has long been, the only native toolkit for Unix which supports large scale internationalized applications.[1]

About the Motif Toolkit

The Motif toolkit, from the Open Software Foundation (OSF), is based on the X Toolkit Intrinsics (Xt), which is the standard mechanism on which many of the toolkits written for the X Window System are based. Xt provides a library of user-interface objects called widgets and gadgets, which provide a convenient interface for creating and manipulating X windows, colormaps, events, and other cosmetic attributes of the display. In short, widgets can be thought of as building blocks that the programmer uses to construct a complete application.

However, the widgets that Xt provides are generic in nature and impose no user-interface policy whatsoever. Providing the look and feel of an interface is the job of a user-interface toolkit such as Motif. Motif provides a complete set of widgets that are designed to implement the application look and feel specified in the *Motif Style Guide* and the *Motif Application Environment Specification*. The Motif toolkit also includes a library of functions for creating and manipulating the widgets and other aspects of the user interface.

The Motif toolkit has other components in addition to the widget set and related functions. Motif provides a User Interface Language (UIL) for describing the initial state of a user interface. UIL is designed to permit rapid prototyping of the user interface for an application. The Motif Resource Manager (Mrm) functions provide the interface between C language application code and UIL. Motif also provides the Motif Window Manager (*mwm*). The appearance and behavior of this window manager is designed to be compatible with the appearance and behavior of the Motif widget set.

About This Manual

This manual contains reference material on the Motif toolkit. This edition is based on Motif 2.1, which is the latest major release of the Motif toolkit. Motif 1.2 is based on

1.The contents of this paragraph were true at the moment of writing. There is now a commercial GUI builder for the Linux toolkits; whether it survives in a free software environment remains to be seen. It is still true that the large scale commercial concerns continue to use Motif for their native Unix toolkit.

Release 6 of the Xlib and Xt specifications (X11R6). This release of Motif provides many new features, including new widget classes and several new functions. In order to cover all of the material, it became necessary to split Volume Six into two separate manuals, a programming manual and a reference manual. Volume Six A is the *Motif Programming Manual* and Volume Six B is the *Motif Reference Manual.*

This manual is part of the sixth volume in the O'Reilly & Associates X Window System Series. It includes reference pages for each of the Motif functions and macros, for the Motif and Xt Intrinsics widget classes, for the Mrm functions, for the Motif clients, and for the UIL file format, data types, and functions. A permuted index and numerous quick reference appendices are also provided.

Volume Six B includes reference pages for all of the new functions and widgets in Motif 2.0 and 2.1. When the functionality of an existing routine or widget has changed in Motif 2.0 or 2.1, the reference page explains the differences between the two versions. Volume Six B also provides a complete set of reference material for UIL and Mrm, which was not covered in the previous edition.

Volumes Six A and B are designed to be used together. Volume Six A provides a complete programmer's guide to the Motif toolkit. Each chapter of the book covers a particular component of the Motif toolkit. Each chapter includes basic tutorial material about creating and manipulating the component, intermediate-level information about the configurable aspects of the component, and any advanced programming topics that are relevant. The chapters also provide numerous programming examples.

To get the most out of the examples in Volume Six A, you will need the exact calling sequences of each function from Volume Six B. To understand fully how to use each of the routines described in Volume Six B, all but the most experienced Motif programmers will need the explanations and examples in Volume Six A.

While the Motif toolkit is based on Xt, the focus of this manual is on Motif itself, not on the X Toolkit Intrinsics. Reference pages for the Xt widget classes are included here to provide a complete picture of the widget class hierarchy. Many reference pages mention related Xt routines, but the functionality of these routines is not described. Detailed information about Xt is provided by Volume 4, *X Toolkit Intrinsics Programming Manual, Motif Edition*, and Volume 5, *X Toolkit Intrinsics Reference Manual.*

How This Manual is Organized

Volume Six B is designed to make it easy and fast to look up virtually any fact about the Motif toolkit. It contains reference pages and numerous helpful appendices.

The book is organized as follows:

Preface	Describes the organization of the book and the conventions it follows.
Section 1	*Motif Functions and Macros*, contains reference pages for all of Motif functions and macros.
Section 2	*Motif and Xt Widget Classes*, contains reference pages for the widget classes defined by the Motif toolkit and the X Toolkit Intrinsics.
Section 3	*Mrm Functions*, contains reference pages for the Motif Resource Manager functions that are used in conjuctions with the User Interface Language.
Section 4	*Motif Clients*, contains reference pages for the Motif clients: *mwm*, *uil*, and *xmbind*.
Section 5	*UIL File Format*, contains reference pages that describe the file format of a User Interface Language module.
Section 6	*UIL Data Types*, contains reference pages for the data types supported by the User Interface Language.
Section 7	*UIL Functions*, contains reference pages for the User Interface Language functions.
Appendix A	*Function Summaries*, provides quick reference tables that list each Motif function alphabetically and also by functional groups.
Appendix B	*Data Types*, lists and explains in alphabetical order the structures, enumerated types, and other typedefs used for arguments to Motif and Mrm functions.
Appendix C	*Table of Motif Resources*, lists all of the resources provided by Motif and Xt widget classes, along with their types and the classes that define them.
Appendix D	*Table of UIL Objects*, lists all of the objects supported by the User Interface Language, along with their corresponding Motif widget classes.
Appendix E	*New Features in Motif 1.2*, lists the new functions, widget classes, and widget resources in Motif 1.2.
Index	Should help you to find what you need to know.

Assumptions

This book assumes that the reader is familiar with the C programming language and the concepts and architecture of the X Toolkit, which are presented in Volume 4, *X Toolkit Intrinsics Programming Manual, Motif Edition*, and Volume 5, *X Toolkit Intrinsics Reference Manual*. A basic understanding of the X Window System is also useful. For some advanced topics, the reader may need to consult Volume 1, *Xlib Programming Manual*, and Volume 2, *Xlib Reference Manual.*

Related Documents

The following books on the X Window System are available from O'Reilly & Associates, Inc.:

Volume Zero	*X Protocol Reference Manual*
Volume One	*Xlib Programming Manual*
Volume Two	*Xlib Reference Manual*
Volume Three	*X Window System User's Guide, Motif Edition*
Volume Four	*X Toolkit Intrinsics Programming Manual, Motif Edition*
Volume Five	*X Toolkit Intrinsics Reference Manual*
Volume Six A	*Motif Programming Manual*
Volume Seven	*XView Programming Manual* with accompanying reference volume.
Volume Eight	*X Window System Administrator's Guide*
PHIGS Programming Manual	
PHIGS Reference Manual	
PEXlib Programming Manual	
PEXlib Reference Manual	
Quick Reference	*The X Window System in a Nutshell*

Programming Supplement for Release 6 of the X Window System

Conventions Used in This Book

Italic is used for:

- UNIX pathnames, filenames, program names, user command names, options for user commands, and variable expressions in syntax sections.
- New terms where they are defined.

`Constant Width` Font is used for:

- Anything that would be typed verbatim into code, such as examples of source code and text on the screen.
- Variables, data structures (and fields), symbols (defined constants and bit flags), functions, macros, and a general assortment of anything relating to the C programming language.
- All functions relating to Motif, Xt, and Xlib.
- Names of subroutines in example programs.

`Constant Width Italic` Font is used for:

- Arguments to functions, since they could be typed in code as shown but are arbitrary names that could be changed.

Helvetica Italic is used for:

- Titles of examples, figures, and tables.

Boldface is used for:

- Chapter headings, section headings, and the names of buttons and menus.

We'd Like to Hear From You

We have tested and verified all of the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing:

O'Reilly & Associates, Inc.

103 Morris Street, Suite A

Sebastopol, CA 95472

1-800-998-9938 (in the US or Canada)

1-707-829-0515 (international / local)

1-707-829-0104 (FAX)

Acknowledgements

This book developed out of the realization that it would be impossible to update the first edition of Volume Six to cover Motif 1.2 without dividing the original book into two books. Dan Heller, David Flanagan, Adrian Nye, and Tim O'Reilly all provided valuable suggestions on how best to expand the original reference appendices into a full-fledged reference manual.

The Motif reference pages in this book are based on the reference appendices from the first edition, which were developed by Daniel Gilly. His work meant that I didn't have to start from scratch, and thus saved many hours of toil. The OSF/Motif reference material also provided a helpful foundation from which to explore the complexities of the Motif toolkit. Many of the Motif examples in the book were borrowed from the first edition of Volume Six. These example were written by Dan Heller, although they have been updated for Motif 1.2

Dave Brennan, of HaL Computer Systems, took on the unenviable task of learning everything there is to know about UIL and Mrm, so that he could write the UIL reference material. He did a great job.

Adrian Nye deserves special recognition for freeing me to work on this project, when I'm sure that he had other projects he would have liked to send my way. I don't think either one of us had any idea how involved this update project would become. The other inhabitants of the "writer's block" at O'Reilly & Associates, Valerie Quercia, Linda Mui, and Ellie Cutler, provided support that kept me sane while I was working on the book. Extra gratitude goes to Linda Mui for her work on the cross references and the reference tables; her knowledge of various tools prevented me from doing things the hard way. Tim O'Reilly also provided editorial support that improved the quality of the reference material.

Special thanks go to the people who worked on the production of this book. The final form of this book is the work of the staff at O'Reilly & Associates. The authors would like to thank Chris Reilly for the figures, Ellie Cutler for indexing, Lenny Muellner for tools support, Eileen Kramer for copy editing and production of the final copy, and Clairemarie Fisher O'Leary for final proofing and printing. Thanks also to Donna Woonteiler for her patience in answering my questions and helping me to understand the production process.

Despite the efforts of all of these people, the authors alone are responsible for any errors or omissions that remain.

Paula M. Ferguson

Acknowledgements to the Motif 2.1 Edition

Many thanks to all at IST who gave me the time and opportunity to perform this work. I would like to thank all those who reviewed the material, which in a Reference Manual of this type is a tedious but necessary task: a very big "Thank You" to Andy Bartlett who took the trouble of sitting down with the Motif sources whilst pouring over every technical detail, and to Tricia Lovell who reviewed the format at particularly short notice.

A special thanks also to Richard Offer and Doug Rand from Silicon Graphics, and Mark Riches for casting expert and independent eyes over the materials. I would also like to thank Andy Lovell and Derek Lambert for allowing and freeing me up to perform the task. To the rest of the company, who have had to wait whilst yet another batch of print jobs ran to completion, all I can say is "Sorry".

A very big “Thank You” indeed to all at O’Reilly for allowing me to undertake this important task, and especially to Paula Ferguson, my editor: I could not have done this without you.

But to my wife Emma, who put up with some seriously late nights over a long period, goes the biggest "Thank You" of all. This would not have happened without any of you, and I am extremely grateful.

Antony J. Fountain

Acknowledgements to the Open Source Edition

Again, many thanks to all at IST who helped me convert the original troff to Frame and PDF formats. A special thank you to Denise Huxtable who enlightened me on the mysteries of Reference Pages, Indexes, and Tables of Contents. Denise also performed much of the cross-referencing in the manual. Thank you also to Ruth Lambert, who showed me how to mark up the document sources.

Again, a very big “Thank You” to all at O’Reilly, and Paula Fergusson in particular, for helping this open source edition come about.

And again, to my wife Emma: a big kiss, and I’ll be home real soon now.

Antony J. Fountain

Section 1 - Motif Functions and Macros

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

Name

Function – a brief description of the function.

Synopsis

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

Inputs

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

Outputs

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

Returns

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

Availability

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

Description

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

Usage

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

Example

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

Structures

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

Procedures

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

See Also

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

Name

XmActivateProtocol – activate a protocol.

Synopsis

#include <Xm/Protocols.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmActivateWMProtocol – activate the XA_WM_PROTOCOLS protocol.

Synopsis

#include <Xm/Protocols.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmAddProtocolCallback – add client callbacks to a protocol.

Synopsis

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

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmAddProtocols – add protocols to the protocol manager.

Synopsis

#include <Xm/Protocols.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

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

Synopsis

void XmAddTabGroup (Widget *tab_group*)

Inputs

tab_group Specifies the widget to be added.

Availability

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

Description

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

Usage

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

See Also

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

Name

XmAddToPostFromList – make a menu accessible from a widget.

Synopsis

#include <Xm/RowColumn.h>

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

Inputs

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

Availability

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

Description

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

Usage

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

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

See Also

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

Name

XmAddWMProtocolCallback – add client callbacks to an XA_WM_PROTOCOLS protocol.

Synopsis

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

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

Inputs

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

Description

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

Usage

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

Example

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

```
Atom wm_save_yourself;

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

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

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

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

See Also

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

Name

XmAddWMProtocols – add the XA_WM_PROTOCOLS protocols to the protocol manager.

Synopsis

#include <Xm/Protocols.h>

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

Inputs

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

Description

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

Usage

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

Example

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

```
Atom wm_save_yourself;

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

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

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

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

See Also

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

Name

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

Synopsis

#include <Xm/CascadeB.h>

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

#include <Xm/CascadeBG.h>

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

Inputs

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

Description

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

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

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

Usage

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

See Also

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

Name

XmChangeColor – update the colors for a widget.

Synopsis

void XmChangeColor (Widget widget, Pixel background)

Inputs

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

Description

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

Usage

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

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

See Also

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

Name

XmClipboardBeginCopy – set up storage for a clipboard copy operation.

Synopsis

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

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

Inputs

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

Outputs

item_id	Returns the ID assigned to the data item.

Returns

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

Description

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

Usage

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

Procedures

The VoidProc has the following format:

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

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

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

See Also

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

Name

XmClipboardCancelCopy – cancel a copy operation to the clipboard.

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

Returns

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

Description

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

Usage

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

See Also

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

Name

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

Synopsis

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

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

Inputs

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

Outputs

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

Returns

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

Description

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

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

Usage

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

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

Example

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

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

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

See Also

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

Name

XmClipboardCopyByName – copy a data item passed by name.

Synopsis

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

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

Inputs

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

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

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

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

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

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

Returns

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

Description

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

Usage

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

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

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

Example

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

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

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

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

See Also

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

Name

XmClipboardEndCopy – end a copy operation to the clipboard.

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

Returns

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

Description

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

Usage

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

Example

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

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

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

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

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

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

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

See Also

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

Name

XmClipboardEndRetrieve – end a copy operation from the clipboard.

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

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

Returns

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

Description

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

Usage

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

Example

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

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

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

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

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

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

See Also

XmClipboardRetrieve(1), XmClipboardStartRetrieve(1).

Name

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

Synopsis

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

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

Inputs

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

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

Outputs

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

max_length Returns the maximum length of data item format names.

Returns

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

Description

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

Usage

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

See Also

`XmClipboardInquireFormat`(1).

Name

XmClipboardInquireFormat – get the specified clipboard data format name.

Synopsis

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

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

Inputs

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

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

index Specifies the index of the format name to retrieve.

buffer_len Specifies the length of format_name_buf in bytes.

Outputs

format_name_buf Returns the format name.

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

Returns

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

Description

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

Usage

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

See Also

`XmClipboardInquireCount`(1).

Name

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

Synopsis

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

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

Inputs

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

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

format_name Specifies the format name for the data.

Outputs

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

Returns

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

Description

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

Usage

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

Example

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

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

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

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

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

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

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

See Also

XmClipboardRetrieve(1).

Name

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

Synopsis

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

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

Inputs

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

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

format_name Specifies the format name for the data.

Outputs

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

count Returns the number of items in the item_list array.

Returns

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

Description

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

Usage

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

Structures

The XmClipboardPendingList is defined as follows:

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

See Also

`XmClipboardStartCopy`(1).

Name

XmClipboardLock – lock the clipboard.

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

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

Returns

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

Description

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

Usage

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

See Also

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

Name

XmClipboardRegisterFormat – register a new format for clipboard data items.

Synopsis

#include <Xm/CutPaste.h>

int XmClipboardRegisterFormat (Display *_display_, char *_format_name_, int _format_length_)

Inputs

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

Returns

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

Description

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

Usage

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

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

See Also

`XmClipboardStartCopy`(1).

Name

XmClipboardRetrieve – retrieve a data item from the clipboard.

Synopsis

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

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

Inputs

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

Outputs

num_bytes Returns the number of bytes of data copied into buffer.
private_id Returns the private data that was stored with the data item.

Returns

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

Description

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

Usage

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

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

Example

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

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

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

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

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

See Also

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

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

Name

XmClipboardStartCopy – set up storage for a clipboard copy operation.

Synopsis

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

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

Inputs

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

Outputs

item_id	Returns the ID assigned to the data item.

Returns

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

Description

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

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

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

Usage

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

Example

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

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

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

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

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

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

   clip_label = XmStringCreateLocalized ("Data");

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

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

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

Procedures

The XmCutPasteProc has the following format:

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

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

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

See Also

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

Name

XmClipboardStartRetrieve – start a clipboard retrieval operation.

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

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

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

Returns

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

Description

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

Usage

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

Example

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

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

do
```

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

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

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

See Also

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

Name

XmClipboardUndoCopy – remove the last item copied to the clipboard.

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

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

Returns

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

Description

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

Usage

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

See Also

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

Name

XmClipboardUnlock – unlock the clipboard.

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

Returns

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

Description

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

Usage

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

See Also

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

Name

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

Synopsis

#include <Xm/CutPaste.h>

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

Inputs

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

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

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

Returns

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

Description

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

Usage

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

See Also

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

Name

XmComboBoxAddItem – add a compound string to the ComboBox list.

Synopsis

#include <Xm/ComboBox.h>

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

Inputs

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

Availability

Motif 2.1 and later.

Description

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

Usage

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

See Also

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

Name

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

Synopsis

#include <Xm/ComboBox.h>

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

Inputs

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

Availability

Motif 2.1 and later.

Description

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

Usage

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

See Also

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

Name

XmComboBoxSelectItem – select an item from a ComboBox list.

Synopsis

#include <Xm/ComboBox.h>

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

Inputs

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

Availability

Motif 2.1 and later.

Description

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

Usage

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

See Also

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

Name

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

Synopsis

#include <Xm/ComboBox.h>

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

Inputs

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

Availability

Motif 2.1 and later.

Description

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

Usage

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

See Also

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

Name

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

Synopsis

#include <Xm/ComboBox.h>

void XmComboBoxUpdate (Widget *widget*)

Inputs

widget Specifies the ComboBox widget.

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

See Also

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

Name

XmCommandAppendValue – append a compound string to the command.

Synopsis

#include <Xm/Command.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmCommandError – display an error message in a Command widget.

Synopsis

#include <Xm/Command.h>

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

Inputs

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

Description

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

Usage

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

See Also

`XmCommand`(2).

Name

XmCommandGetChild – get the specified child of a Command widget.

Synopsis

#include <Xm/Command.h>

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

Inputs

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

Returns

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

Availability

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

Description

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

Usage

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

Structures

The possible values for child are:

XmDIALOG_COMMAND_TEXT	XmDIALOG_HISTORY_LIST
XmDIALOG_PROMPT_LABEL	XmDIALOG_WORK_AREA

Widget Hierarchy

The following names are associated with the Command children:

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

See Also

XmCommand(2).

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

Name

XmCommandSetValue – replace the command string.

Synopsis

#include <Xm/Command.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmContainerCopy – copy the Container primary selection onto the clipboard.

Synopsis

#include <Xm/Container.h>

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

Inputs

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

Returns

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

Availability

Motif 2.0 and later.

Description

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

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

Usage

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

See Also

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

Name

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

Synopsis

#include <Xm/Container.h>

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

Inputs

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

Returns

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

Availability

Motif 2.0 and later.

Description

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

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

Usage

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

See Also

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

Name

XmContainerCut – cuts the Container primary selection onto the clipboard.

Synopsis

#include <Xm/Container.h>

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

Inputs

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

Returns

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

Availability

Motif 2.0 and later.

Description

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

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

Usage

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

See Also

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

Name

XmContainerGetItemChildren – find the children of a Container item.

Synopsis

#include <Xm/Container.h>

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

Inputs

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

Outputs

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

Returns

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

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

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

See Also

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

Name

XmContainerPaste – pastes the clipboard selection into a Container.

Synopsis

#include <Xm/Container.h>

Boolean XmContainerPaste (Widget *container*)

Inputs

container Specifies a Container widget.

Returns

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

Availability

Motif 2.0 and later.

Description

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

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

Usage

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

See Also

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

Name

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

Synopsis

#include <Xm/Container.h>

Boolean XmContainerPasteLink (Widget *container*)

Inputs

container Specifies a Container widget.

Returns

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

Availability

Motif 2.0 and later.

Description

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

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

Usage

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

See Also

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

Name

XmContainerRelayout – force relayout of a Container widget.

Synopsis

#include <Xm/Container.h>

void XmContainerRelayout (Widget *container*)

Inputs

container Specifies a Container widget.

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

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

See Also

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

Name

XmContainerReorder – reorder children of a Container.

Synopsis

#include <Xm/Container.h>

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

Inputs

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

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

See Also

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

Name

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

Synopsis

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

Inputs

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

Outputs

error_return	Returns the error status of the conversion.

Returns

The converted value.

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

<float> <unit>

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

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

Structures

The possible values for unit_type are:

XmPIXELS XmCENTIMETERS XmMILLIMETERS
Xm100TH_MILLIMETERS XmINCHES
Xm1000TH_INCHES
XmPOINTS Xm100TH_POINTS
XmFONT_UNITS
Xm100TH_FONT_UNITS

Example

The following are valid string specifications:

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

See Also

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

Name

XmConvertUnits – convert a value to a specified unit type.

Synopsis

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

Inputs

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

Returns

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

Description

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

Usage

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

Structures

The possible values for from_unit_type and to_unit_type are:

XmPIXELS	XmCENTIMETERS
XmMILLIMETERS	Xm100TH_MILLIMETERS
XmINCHES	Xm1000TH_INCHES

XmPOINTS	Xm100TH_POINTS
XmFONT_UNITS	Xm100TH_FONT_UNITS

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

See Also

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

Name

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

Synopsis

Simple Widgets

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

#include <Xm/MenuShell.h>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Dialog Objects

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

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

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

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

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

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

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

Menu Objects

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

Simple Menu Objects

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

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

Scrolled Objects

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

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

Inputs

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

Returns

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

Availability

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

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

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

Description

The `XmCreate`*() routines are convenience routines for creating an instance of a particular widget class or a particular compound object. Each creation routine takes the same four arguments: the *parent*'s widget ID, the *name* of the new widget, a list of resource name/value pairs, and the number of name/value pairs.

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

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

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

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

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

Usage

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

See Also

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

Name

XmCvtByteStreamToXmString – convert a byte stream to a compound string.

Synopsis

XmString XmCvtByteStreamToXmString (unsigned char **property*)

Inputs

property Specifies a byte stream.

Returns

An allocated compound string.

Availability

Motif 2.0 and later.

Description

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

Usage

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

See Also

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

Name

XmCvtCTToXmString – convert compound text to a compound string.

Synopsis

XmString XmCvtCTToXmString (char **text*)

Inputs

text Specifies the compound text that is to be converted.

Returns

The converted compound string.

Description

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

Usage

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

See Also

`XmCvtXmStringToCT`(1).

Name

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

Synopsis

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

Inputs

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

num_args Specifies the number of items in args.

from_val Specifies value to convert.

Outputs

to_val Returns the converted value.

Availability

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

Description

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

Usage

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

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

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

See Also

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

Name

XmCvtTextPropertyToXmStringTable – convert an XTextProperty to a Compound String Table.

Synopsis

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

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

Inputs

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

Outputs

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

Returns

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

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

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

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

Structures

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

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

See Also

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

Name

XmCvtXmStringTableToTextProperty – convert an XmStringTable to an XTextProperty.

Synopsis

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

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

Inputs

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

Outputs

prop_return	The XTextProperty structure converted from *string_table*.

Returns

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

Availability

Motif 2.0 and later.

Description

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

Usage

`XmCvtXmStringTableToTextProperty`() converts an XmStringTable into the elements of an XTextProperty structure. The encoding member contains an Atom representing the requested *style*. The value member contains a list of the

converted items, each separated by NULL bytes, and terminated by two NULL bytes, the nitems member is the number of such items converted.

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

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

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

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

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

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

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

Structures

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

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

The possible values of the XmICCEncodingStyle parameter style are:

```
XmSTYLE_COMPOUND_STRING
XmSTYLE_COMPOUND_TEXT
XmSTYLE_LOCALE
XmSTYLE_STANDARD_ICC_TEXT
XmSTYLE_STRING
XmSTYLE_TEXT
```

See Also

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

Name

XmCvtXmStringToByteStream – convert a compound string to byte stream format.

Synopsis

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

Inputs

string Specifies the compound string that is to be converted.

Outputs

prop_return The converted compound string in byte stream format.

Returns

The number of bytes in the byte stream.

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

See Also

`XmCvtByteStreamToXmString`(1).

Name

XmCvtXmStringToCT – convert a compound string to compound text.

Synopsis

char * XmCvtXmStringToCT (XmString *string*)

Inputs

string Specifies the compound string that is to be converted.

Returns

The converted compound text string.

Description

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

Usage

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

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

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

See Also

`XmCvtCTToXmString`(1), `XmMapSegmentEncoding`(1), `XmRegisterSegmentEncoding`(1).

Name

XmDeactivateProtocol – deactivate a protocol.

Synopsis

#include <Xm/Protocols.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmDeactivateWMProtocol – deactivate the XA_WM_PROTOCOLS protocol.

Synopsis

#include <Xm/Protocols.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmDestroyPixmap – remove a pixmap from the pixmap cache.

Synopsis

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

Inputs

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

Returns

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

Description

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

Usage

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

See Also

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

Name

XmDirectionMatch – compare two directions.

Synopsis

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

Inputs

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

Returns

True if the directions match, otherwise False.

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

Structures

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

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

See Also

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

Name

XmDirectionMatchPartial – partially compare two directions.

Synopsis

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

Inputs

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

Returns

True if the directions match, otherwise False.

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

Structures

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

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

See Also

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

Name

XmDirectionToStringDirection – convert a direction to a string direction.

Synopsis

XmStringDirection XmDirectionToStringDirection (XmDirection *direction*)

Inputs

direction Specifies the direction to be converted.

Returns

The equivalent XmStringDirection.

Availability

Motif 2.0 and later.

Description

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

Usage

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

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

See Also

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

Name

XmDragCancel – cancel a drag operation.

Synopsis

#include <Xm/DragDrop.h>

void XmDragCancel (Widget *dragcontext*)

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmDragStart – start a drag operation.

Synopsis

#include <Xm/DragDrop.h>

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

Inputs

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

Returns

The ID of the DragContext object that is created.

Availability

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

Description

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

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

Usage

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

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

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

Example

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

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

/* global variable */
Atom COMPOUND_TEXT;

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

   exportList[0] = COMPOUND_TEXT;
```

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

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

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

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

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

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

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

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

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

    /* override button two press to start a drag */
```

```
        parsed_trans = XtParseTranslationTable
        (dragTranslations);
        XtAppAddActions (app, dragActions, XtNumber
        (dragActions));

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

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

See Also

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

Name

XmDropSiteConfigureStackingOrder – change the stacking order of a drop site.

Synopsis

#include <Xm/DragDrop.h>

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

Inputs

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

Description

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

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

Usage

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

See Also

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

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

Name

XmDropSiteEndUpdate – end an update of multiple drop sites.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteEndUpdate (Widget *widget*)

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmDropSiteQueryStackingOrder – get the stacking order of a drop site.

Synopsis

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

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

Inputs

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

Outputs

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

Returns

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

Description

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

Usage

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

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

See Also

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

Name

XmDropSiteRegister – register a drop site.

Synopsis

#include <Xm/DragDrop.h>

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

Inputs

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

Availability

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

Description

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

Usage

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

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

Example

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

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

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

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

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

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

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

```
    XtManageChild (label);

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

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

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

See Also

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

Name

XmDropSiteRetrieve – get the resource values for a drop site.

Synopsis

#include <Xm/DragDrop.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

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

Name

XmDropSiteStartUpdate – start an update of multiple drop sites.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteStartUpdate (Widget *widget*)

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmDropSiteUnregister – remove a drop site.

Synopsis

#include <Xm/DragDrop.h>

void XmDropSiteUnregister (Widget *widget*)

Inputs

widget Specifies the widget ID associated with the drop site.

Description

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

Usage

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

See Also

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

Name

XmDropSiteUpdate – change the resource values for a drop site.

Synopsis

#include <Xm/DragDrop.h>

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

Inputs

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

Description

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

Usage

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

See Also

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

Name

XmDropTransferAdd – add drop transfer entries to a drop operation.

Synopsis

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

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

Inputs

drop_transfer	Specifies the ID of the DropTransfer object to which the entries are being added.
transfers	Specifies the additional drop transfer entries.
num_transfer	Specifies the number of drop transfer entries in *transfers*.

Availability

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

Description

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

Usage

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

Structures

XmDropTransferEntryRec is defined as follows:

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

See Also

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

Name

XmDropTransferStart – start a drop operation.

Synopsis

#include <Xm/DragDrop.h>

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

Inputs

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

Returns

The ID of the DropTransfer object that is created.

Availability

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

Description

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

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

Usage

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

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

Example

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

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

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

   DropData = (XmDropProcCallback) call_data;

   n = 0;

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

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

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

See Also

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

Name

XmFileSelectionBoxGetChild – get the specified child of a FileSelectionBox widget.

Synopsis

#include <Xm/FileSB.h>

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

Inputs

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

Returns

The widget ID of the specified child of the FileSelectionBox.

Availability

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

Description

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

Usage

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

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

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

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

Widget Hierarchy

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

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

1.Called internally by XmFileSelectionBoxGetChild().

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

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

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

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

Structures

The possible values for child are:

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

See Also

`XmFileSelectionBox`(2).

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

Name

XmFileSelectionDoSearch – start a directory search.

Synopsis

#include <Xm/FileSB.h>

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

Inputs

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

Description

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

Usage

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

See Also

`XmFileSelectionBox`(2).

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

Name

XmFontListAdd – create a new font list.

Synopsis

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

Inputs

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

Returns

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

Availability

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

Description

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

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

Usage

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

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

See Also

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

Name

XmFontListAppendEntry – append a font entry to a font list.

Synopsis

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

Inputs

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

Returns

The new font list or oldlist if entry is NULL.

Availability

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

Description

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

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

Usage

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

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

See Also

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

Name

XmFontListCopy – copy a font list.

Synopsis

XmFontList XmFontListCopy (XmFontList *fontlist*)

Inputs

fontlist Specifies the font list to be copied.

Returns

The new font list or NULL if fontlist is NULL.

Availability

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

Description

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

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

Usage

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

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

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

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

See Also

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

Name

XmFontListCreate – create a font list.

Synopsis

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

Inputs

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

Returns

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

Availability

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

Description

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

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

Usage

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

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

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

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

See Also

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

Name

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

Synopsis

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

Inputs

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

Returns

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

Availability

Motif 2.1 and later.

Description

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

Usage

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

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

See Also

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

Name

XmFontListEntryCreate – create a font list entry.

Synopsis

XmFontListEntry XmFontListEntryCreate (char **tag*, XmFontType *type*, XtPointer *font*)

Inputs

tag	Specifies the tag for the font list entry.
type	Specifies the type of the font argument. Pass either XmFONT_IS_FONT or XmFONT_IS_FONTSET.
font	Specifies the font or font set.

Returns

A font list entry.

Availability

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

Description

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

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

Usage

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

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

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

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

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

Example

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

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

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

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

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

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

See Also

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

Name

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

Synopsis

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

Inputs

tag	Specifies the tag for the font list entry.
type	Specifies the type of the font argument. Pass either XmFONT_IS_FONT or XmFONT_IS_FONTSET.
font	Specifies the font or font set.
widget	Specifies a widget.

Returns

A font list entry.

Availability

Motif 2.1 and later.

Description

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

Usage

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

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

See Also

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

Name

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

Synopsis

void XmFontListEntryFree (XmFontListEntry **entry*)

Inputs

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

Availability

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

Description

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

Usage

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

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

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

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

See Also

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

Name

XmFontListEntryGetFont – get the font information from a font list entry.

Synopsis

XtPointer XmFontListEntryGetFont (XmFontListEntry *entry*, XmFontType **type_return*)

Inputs

entry Specifies the font list entry.

Outputs

type_return Returns the type of the font information that is returned. Valid types are XmFONT_IS_FONT or XmFONT_IS_FONTSET.

Returns

An XFontSet or a pointer to an XFontStruct.

Availability

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

Description

`XmFontListEntryGetFont`() retrieves the font information for the specified font list *entry*. When the font list *entry* contains a font, *type_return* is XmFONT_IS_FONT and the routine returns a pointer to an XFontStruct. When the font list *entry* contains a font set, *type_return* is XmFONT_IS_FONTSET and the routine returns the XFontSet. The XFontSet or XFontStruct that is returned is not a copy of the data structure, so it must not be freed by an application.

Usage

The XmFontList and XmFontListEntry types are opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list entries and retrieve information about them. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListEntryGetFont`() can be used to get the font structure for a font list entry once it has been retrieved from the font list using `XmFontListNextEntry`().

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryGetFont`() is a convenience routine which fetches the XmNfont and XmNfontType values of the rendition object represented by entry. The values are fetched through the function `XmRenditionRetrieve`().

type_return is set to the value of the XmNfontType resource, and the function `XmFontListEntryGetFont`() returns the value of the XmNfont resource of the rendition object.

`XmFontListEntryGetFont`()[1] is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListEntryCreate`(1), `XmFontListEntryGetTag`(1), `XmFontListEntryLoad`(1), `XmFontListNextEntry`(1), `XmRenditionRetrieve`(1), `XmRendition`(2).

1.Erroneously given as XmFontListGetFont() in 2nd edition.

Name

XmFontListEntryGetTag – get the tag of a font list entry.

Synopsis

char* XmFontListEntryGetTag (XmFontListEntry *entry*)

Inputs

entry Specifies the font list entry.

Returns

The tag for the font list entry.

Availability

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

Description

`XmFontListEntryGetTag`() retrieves the tag of the specified font list *entry.* The routine allocates storage for the tag string; the application is responsible for freeing the memory using `XtFree`().

Usage

The XmFontList and XmFontListEntry types are opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list entries and retrieve information about them. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListEntryGetTag`() can be used to get the tag of a font list entry once it has been retrieved from the font list using `XmFontListNextEntry`().

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryGetTag`() is a convenience routine which fetches and returns a copy of the XmNtag value of the rendition object represented by entry. The value is fetched through the function `XmRenditionRetrieve`().

`XmFontListEntryGetTag`()[1] is retained for compatibility with Motif 1.2 and should not be used in newer applications.

See Also

`XmFontListEntryCreate`(1), `XmFontListEntryGetFont`(1), `XmFontListEntryLoad`(1), `XmFontListNextEntry`(1), `XmRenditionRetrieve`(1), `XmRendition`(2).

1.Erroneously given as XmFontListGetTag() in 2nd edition.

Name

XmFontListEntryLoad – load a font or create a font set and then create a font list entry.

Synopsis

```
XmFontListEntry XmFontListEntryLoad (  Display     *display,
                                       char        *font_name,
                                       XmFontType  type,
                                       char        *tag)
```

Inputs

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

font_name Specifies an X Logical Font Description (XLFD) string.

type Specifies the type of font_name. Pass either XmFONT_IS_FONT or XmFONT_IS_FONTSET.

tag Specifies the tag for the font list entry.

Returns

A font list entry or NULL if the font cannot be found or the font set cannot be created.

Availability

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

Description

`XmFontListEntryLoad`() either loads a font or creates a font set depending on the value of *type* and then creates a font list entry that contains the font data and the specified *tag*. *font_name* is an XLFD string which is parsed as either a font name or a base font name list. *tag* is a NULL-terminated string that identifies the font list entry. It can have the value XmFONTLIST_DEFAULT_TAG, which identifies the default font list entry in a font list.

If *type* is set to XmFONT_IS_FONT, the routine uses the `XtCvtStringToFontStruct`() converter to load the font struct specified by font_name. If the value of *type* is XmFONT_IS_FONTSET, XmFontListEntryLoad uses the `XtCvtStringToFontSet`() converter to create a font set in the current locale.

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

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. `XmFontListEntryLoad`() sets up the font data and creates a font list entry. The font list entry can be added to a font list using `XmFontListAppendEntry`().

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListEntryLoad`() is a convenience routine which creates and returns a rendition object whose XmNfontName resource is set to font_name, and XmNfontType value is type. The rendition object is created with an XmNloadModel of XmLOAD_IMMEDIATE.

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

See Also

`XmFontListAppendEntry`(1), `XmFontListEntryCreate`(1), `XmFontListEntryFree`(1), `XmFontListEntryGetFont`(1), `XmFontListEntryGetTag`(1), `XmFontListRemoveEntry`(1), `XmRenditionCreate`(1), `XmRendition`(2).

Name

XmFontListFree – free the memory used by a font list.

Synopsis

void XmFontListFree (XmFontList *fontlist*)

Inputs

fontlist Specifies the font list that is to be freed.

Availability

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

Description

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

Usage

A font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a font or a font set and an associated tag. `XmFontListFree`() frees the storage used by the font list but does not free the associated font data structures. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. `XmFontListFree`() is a convenience function which simply calls `XmRenderTableFree`().

It is important to call `XmFontListFree`() rather than `XtFree`() because Motif caches font lists. A call to `XmFontListFree`() decrements the reference count for the font list; the font list is not actually freed until the reference count reaches 0 (zero).

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

See Also

`XmFontListAppendEntry`(1), `XmFontListCopy`(1), `XmFontListEntryFree`(1), `XmFontListRemoveEntry`(1), `XmRenderTableFree`(1).

Name

XmFontListFreeFontContext – free a font context.

Synopsis

void XmFontListFreeFontContext (XmFontContext *context*)

Inputs

context Specifies the font list context that is to be freed.

Availability

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

Description

`XmFontListFreeFontContext`() deallocates storage used by the specified font list *context*.

Usage

The XmFontList type is opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListFreeFontContext`() is the last of the three font context routines that an application should call when processing a font list, as it frees the font context data structure. An application begins by calling `XmFontListInitFontContext`() to create a font context and then makes repeated calls to `XmFontListNextEntry`() or `XmFontListGetNextFont`() to cycle through the font list.

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

See Also

`XmFontListGetNextFont`(1), `XmFontListInitFontContext`(1), `XmFontListNextEntry`(1), `XmRenderTableAddRendition`(1), `XmRenditionCreate`(1), `XmRendition`(2).

Name

XmFontListGetNextFont – retrieve information about the next font list element.

Synopsis

```
Boolean XmFontListGetNextFont (  XmFontContext     context,
                                 XmStringCharSet   *charset,
                                 XFontStruct       **font)
```

Inputs

context Specifies the font context for the font list.

Outputs

charset Returns the tag that identifies the character set for the font.

font Returns the font structure for the current font list element.

Returns

True if the values being returned are valid or False otherwise.

Availability

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

Description

`XmFontListGetNextFont`() returns the character set and font for the next element of the font list. *context* is the font context created by `XmFontListInitFontContext`(). The first call to `XmFontListGetNextFont`() returns the first font list element. Repeated calls to `XmFontListGetNextFont`() using the same *context* access successive font list elements. The routine returns False when it has reached the end of the font list.

Usage

A font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a font or a font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

If the routine is called with a font context that contains a font set, it returns the first font of the font set.

The XmFontList type is opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list. These routines use a XmFontContext to maintain an arbitrary position in a

font list. XmFontListGetNextFont() cycles through the fonts in a font list. XmFontListInitFontContext() is called first to create the font context. When an application is done processing the font list, it should call XmFontListFreeFontContext() with the same context to free the allocated data.

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

See Also

XmFontListFreeFontContext(1), XmFontListInitFontContext(1), XmFontListNextEntry(1), XmRendition(2).

Name

XmFontListInitFontContext – create a font context.

Synopsis

Boolean XmFontListInitFontContext (XmFontContext **context*, XmFontList *fontlist*)

Inputs

fontlist Specifies the font list.

Outputs

context Returns the allocated font context structure.

Returns

True if the font context is allocated or False otherwise.

Availability

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

Description

`XmFontListInitFontContext`() creates a font context for the specified *fontlist*. This font context allows an application to access the information that is stored in the font list. `XmFontListInitFontContext`() allocates space for the font *context*. The application is responsible for managing the memory associated with the font context. When the application is done using the font *context*, it should be freed using `XmFontListFreeFontContext`().

Usage

The XmFontList type is opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListInitFontContext`() is the first of the three font context routines that an application should call when processing a font list, as it creates the font context data structure. The context is passed to `XmFontListNextEntry`() or `XmFontListGetNextFont`() to cycle through the font list. When an application is done processing the font list, it should call `XmFontListFreeFontContext`() with the same context to free the allocated data.

In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

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

See Also

`XmFontListFreeFontContext`(1), `XmFontListGetNextFont`(1), `XmFontListInitFontContext`(1), `XmFontListNextEntry`(1), `XmRendition`(2).

Name

XmFontListNextEntry – retrieve the next font list entry in a font list.

Synopsis

XmFontListEntry XmFontListNextEntry (XmFontContext *context*)

Inputs

context Specifies the font context for the font list.

Returns

A font list entry or NULL if the context refers to an invalid entry or if it is at the end of the font list.

Availability

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

Description

`XmFontListNextEntry`() returns the next font list entry in a font list. *context* is the font context created by `XmFontListInitFontContext`(). The first call to `XmFontListNextEntry`() returns the first entry in the font list. Repeated calls to `XmFontListNextEntry`() using the same *context* access successive font list entries. The routine returns NULL when it has reached the end of the font list.

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

The XmFontList and XmFontListEntry types are opaque, so if an application needs to perform any processing on a font list, it has to use special functions to cycle through the font list entries and retrieve information about them. These routines use a XmFontContext to maintain an arbitrary position in a font list. `XmFontListInitFontContext`() is called first to create the font context. `XmFontListNextEntry`() cycles through the font entries in a font list. `XmFontListEntryGetFont`() and `XmFontListEntryGetTag`() access the information in a font list entry. When an application is done processing the font list, it should call `XmFontListFreeFontContext`() with the same context to free the allocated data.

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

See Also

`XmFontListEntryFree`(1), `XmFontListEntryGetFont`(1), `XmFontListEntryGetTag`(1), `XmFontListFreeFontContext`(1), `XmFontListInitFontContext`(1), `XmRendition`(2).

Name

XmFontListRemoveEntry – remove a font list entry from a font list.

Synopsis

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

Inputs

oldlist	Specifies the font list from which entry is removed.
entry	Specifies the font list entry.

Returns

The new font list, oldlist if entry is NULL or no entries are removed, or NULL if oldlist is NULL.

Availability

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

Description

`XmFontListRemoveEntry`() makes a new font list by removing any entries in *oldlist* that match the specified *entry.* The routine returns the new font list and deallocates *oldlist*. `XmFontListRemoveEntry`() does not deallocate the font list *entry*, so the application should free the storage using `XmFontListEntryFree`().

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

Usage

In Motif 1.2, a font list contains font list entries, where each entry consists of a font or font set and an associated tag. In Motif 2.0 and later, the XmFontList is an alias for the XmRenderTable type. XmRendition objects within a render table represent the font entries. The XmFontContext is an opaque type which contains an index into the renditions of a render table.

An application can use `XmFontListRemoveEntry`() to remove a font list entry from a font list. If an application needs to process the font list to determine which entries to remove, it can use `XmFontListInitFontContext`() and `XmFontListNextEntry`() to cycle through the entries in the font list.

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

See Also

`XmFontListAppendEntry`(1), `XmFontListEntryCreate`(1), `XmFontListEntryFree`(1), `XmFontListEntryLoad`(1), `XmFontListFree`(1), `XmRendition`(2).

Name

XmGetAtomName – get the string representation of an atom.

Synopsis

#include <Xm/AtomMgr.h>

String XmGetAtomName (Display **display,* Atom *atom*)

Inputs

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

atom Specifies the atom for the property name to be returned.

Returns

The string that represents atom.

Availability

In Motif 2.0 and later, `XGetAtomName`() is preferred.

Description

`XmGetAtomName`() returns the string that is used to represent a given *atom*. This routine works like Xlib's `XGetAtomName`() routine, but the Motif routine provides the added feature of client-side caching. `XmGetAtomName`() allocates space for the returned string; the application is responsible for freeing this storage using `XtFree`() when the atom is no longer needed.

Usage

An Atom is a number that identifies a property. Properties also have string names. XmGetAtomName() returns the string name specified in the original call to `XmInternAtom`() or `XInternAtom`(), or for predefined atoms, a string version of the symbolic constant without the XA_ attached.

In Motif 2.0 and later, `XmGetAtomName`() is no more than a convenience routine which calls `XGetAtomName`(). While `XmGetAtomName`() is not yet obsolete, `XGetAtomName`() is to be preferred.

See Also

`XmInternAtom`(1).

Name

XmGetColorCalculation – get the procedure that calculates default colors.

Synopsis

XmColorProc XmGetColorCalculation (void)

Returns

The procedure that calculates default colors.

Description

`XmGetColorCalculation`() returns the procedure that calculates the default foreground, top and bottom shadow, and select colors. The procedure calculates these colors based on the background color that is passed to the procedure.

Usage

Motif widgets rely on the use of shadowed borders to achieve their three-dimensional appearance. The top and bottom shadow colors are lighter and darker shades of the background color; these colors are reversed to make a component appear raised out of the screen or recessed into the screen. The select color is a slightly darker shade of the background color that indicates that a component is selected. The default foreground color is either black or white, depending on which color provides the most contrast with the background color. `XmGetColorCalculation`() returns the procedure that calculates these colors. Use `XmSetColorCalculation`() to change the calculation procedure.

In Motif 2.0 and later, color calculation procedures can be specified on a per-screen basis by specifying a value for the XmScreen object XmNcolorCalculationProc resource. Where a particular XmScreen does not have an assigned calculator, the procedure specified by `XmGetColorCalculation`() is used as the default.

Procedures

The XmColorProc has the following syntax:

```
typedef void (*XmColorProc) (   XColor  *bg_color,    /* specifies the
background color */
                                XColor  *fg_color,    /* returns the fore-
ground color */
                                XColor  *sel_color,   /* returns the select
color */
                                XColor  *ts_color,    /* returns the top
shadow color */
                                XColor  *bs_color)    /* returns the bot-
tom shadow color */
```

An XmColorProc takes five arguments. The first argument, *bg_color*, is a pointer to an XColor structure that specifies the background color. The red, green, blue, and pixel fields in the structure contain valid values. The rest of the arguments are pointers to XColor structures for the colors that are to be calculated. The procedure fills in the red, green, and blue fields in these structures.

See Also

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

Name

XmGetColors – update the colors for a widget.

Synopsis

```
void XmGetColors (  Screen      *screen,
                    Colormap    color_map,
                    Pixel       background,
                    Pixel       *foreground_return,
                    Pixel       *top_shadow_return,
                    Pixel       *bottom_shadow_return,
                    Pixel       *select_return)
```

Inputs

screen	Specifies the screen for which colors are to be allocated.
color_map	Specifies a Colormap from which the colors are allocated.
background	Specifies the background from which to calculate allocated colors.

Outputs

foreground_return	Specifies an address into which the foreground Pixel is returned.
top_shadow_return	Specifies an address into which the top shadow Pixel is returned.
bottom_shadow_return	Specifies an address into which the bottom shadow Pixel is returned.
select_return	Specifies an address into which the select Pixel is returned.

Description

`XmGetColors`() allocates and returns a set of pixels within a Colormap associated with a given *screen* for use as the foreground, top shadow, bottom shadow, and select colors of a widget. The returned values are calculated based upon a supplied background.

Usage

`XmGetColors`() allocates a set of pixels from a colormap. The pixels required are based upon a supplied background pixel. If any return address is specified as NULL, the relevant pixel is not allocated. In Motif 1.2 and earlier, pixels are allocated using the current color calculation procedure, which can be specified using `XmSetColorCalculation`(). In Motif 2.0 and later, per-screen color calculation procedures are supported: if the XmNcolorCalculationProc resource of the XmScreen object associated with screen is not NULL, the procedure specified by the resource is used to calculate the pixels. Otherwise, the current color calculation procedure is used.

See Also

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

Name

XmGetDestination – get the current destination widget.

Synopsis

Widget XmGetDestination (Display **display*)

Inputs

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

Returns

The widget ID of the current destination widget or NULL if there is no current destination widget.

Description

`XmGetDestination`() returns the widget ID of the current destination widget for the specified *display.* The destination widget is usually the widget most recently changed by a select, edit, insert, or paste operation. `XmGetDestination`() identifies the widget that serves as the destination for quick paste operations and some clipboard routines. This routine returns NULL if there is no current destination, which occurs when no edit operations have been performed on a widget.

Usage

XmGetDestination() provides a way for an application to retrieve the widget that would be acted on by various selection operations, so that the application can do any necessary processing before the operation occurs.

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1).

Name

XmGetDragContext – get information about a drag and drop operation.

Synopsis

#include <Xm/DragDrop.h>

Widget XmGetDragContext (Widget *widget*, Time *timestamp*)

Inputs

widget Specifies a widget on the display where the drag and drop operation is taking place.

timestamp Specifies a timestamp that identifies a DragContext.

Returns

The ID of the DragContext object or NULL if no active DragContext is found.

Availability

Motif 1.2 and later.

Description

`XmGetDragContext`() retrieves the DragContext object associated with the display of the specified *widget* that is active at the specified *timestamp*. When more that one drag operation has been started on a display, a timestamp can uniquely identify the active DragContext. If the specified *timestamp* corresponds to a timestamp processed between the beginning and end of a single drag and drop operation, `XmGetDragContext`() returns the DragContext associated with the operation. If there is no active DragContext for the time-stamp, the routine returns NULL.

Usage

Motif 1.2 and later supports the drag and drop model of selection actions. Every drag and drop operation has a DragContext object associated with it that stores information about the drag operation. Both the initiating and the receiving clients use information in the DragContext to process the drag transaction. The DragContext object is widget-like, in that it uses resources to specify its attributes. These resources can be checked using `XtGetValues`() and modified using `XtSetValues`().

`XmGetDragContext`() provides a way for an application to retrieve a DragContext object. The application can then use `XtGetValues`() and `XtSetValues`() to manipulate the DragContext.

See Also

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

Name

XmGetFocusWidget – get the widget that has the keyboard focus.

Synopsis

Widget XmGetFocusWidget (Widget *widget*)

Inputs

widget Specifies the widget whose hierarchy is to be traversed.

Returns

The widget ID of the widget with the keyboard focus or NULL if no widget has the focus.

Availability

Motif 1.2 and later.

Description

`XmGetFocusWidget`() returns the widget ID of the widget that has keyboard focus in the widget hierarchy that contains the specified *widget*. The routine searches the widget hierarchy that contains the specified widget up to the nearest shell ancestor. `XmGetFocusWidget`() returns the widget in the hierarchy that currently has the focus, or the widget that last had the focus when the user navigated to another hierarchy. If no widget in the hierarchy has the focus, the routine returns NULL.

Usage

XmGetFocusWidget() provides a means of determining the widget that currently has the keyboard focus, which can be useful if you are trying to control keyboard navigation in an application.

See Also

`XmGetTabGroup`(1), `XmGetVisibility`(1), `XmIsTraversable`(1), `XmProcessTraversal`(1).

Name

XmGetMenuCursor – get the current menu cursor.

Synopsis

Cursor XmGetMenuCursor (Display **display*)

Inputs

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

Returns

The cursor ID for the current menu cursor or None if no cursor has been defined.

Availability

In Motif 1.2 and later, `XmGetMenuCursor`() is obsolete. It has been superseded by getting the Screen resource XmNmenuCursor.

Description

`XmGetMenuCursor`() returns the cursor ID of the menu cursor currently in use by the application on the specified display. The routine returns the cursor for the default screen of the display. If the cursor is not yet defined because the application called the routine before any menus were created, then XmGetMenuCursor() returns the value None.

Usage

The menu cursor is the pointer shape that is used whenever a menu is posted. This cursor can be different from the normal pointer shape. In Motif 1.2 and later, the new Screen object has a resource, XmNmenuCursor, that specifies the menu cursor. `XmGetMenuCursor`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

See Also

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

Name

XmGetPixmap – create and return a pixmap.

Synopsis

Pixmap XmGetPixmap (Screen **screen*, char **image_name*, Pixel *foreground*, Pixel *background*)

Inputs

screen Specifies the screen on which the pixmap is to be drawn.
image_name Specifies the string name of the image used to make the pixmap.
foreground Specifies the foreground color that is combined with the image when it is a bitmap.
background Specifies the background color that is combined with the image when it is a bitmap.

Returns

A pixmap on success or XmUNSPECIFIED_PIXMAP when the specified image_name cannot be found.

Description

`XmGetPixmap`() generates a pixmap, stores it in the pixmap cache, and returns its resource ID. Before the routine actually creates the pixmap, it checks the pixmap cache for a pixmap that matches the specified *image_name*, *screen*, *foreground*, and *background*. If a match is found, the reference count for the pixmap is incremented and the resource ID for the pixmap is returned. If no pixmap is found, `XmGetPixmap`() checks the image cache for a image that matches the specified *image_name*. If a matching image is found, it is used to create the pixmap that is returned.

When no matches are found, `XmGetPixmap`() begins a search for an X10 or X11 bitmap file, using *image_name* as the filename. If a file is found, its contents are read, converted into an image, and cached in the image cache. Then, the image is used to generate a pixmap that is subsequently cached and returned. The depth of the pixmap is the default depth of the *screen*. If *image_name* specifies a bitmap, the *foreground* and *background* colors are combined with the image. If no file is found, the routine returns XmUNSPECIFIED_PIXMAP.

Usage

When *image_name* starts with a slash (/), it specifies a full pathname and `XmGetPixmap`() opens the specified file. Otherwise, *image_name* specifies a filename which causes `XmGetPixmap`() to look for the file using a search path. In Motif 1.2 and earlier, the XBMLANGPATH environment variable specifies the search path for X bitmap files. In Motif 2.0 and later, the environment variables XMICONSEARCHPATH and XMICONBMSEARCHPATH specify search

paths for pixmap files: XMICONSEARCHPATH is used if a color server is running, XMICONBMSEARCHPATH otherwise, and XBMLANGPATH is used as a fallback.

The search path can contain the substitution character %B, where image_name is substituted for %B. The search path can also use the substitution characters accepted by `XtResolvePathname`(), where %T is mapped to bitmaps and %S is mapped to NULL.

If XBMLANGPATH is not set, `XmGetPixmap`() uses a default search path. If the XAPPLRESDIR environment variable is set, the routine searches the following paths:

%B	
$XAPPLRESDIR/%L/bitmaps/%N/%B	/usr/lib/X11/%L/bitmaps/%N/%B
$XAPPLRESDIR/%l_%t/bitmaps/%N/%B	/usr/lib/X11/%l_%t/bitmaps/%N/%B
$XAPPLRESDIR/%l/bitmaps/%N/%B	/usr/lib/X11/%l/bitmaps/%N/%B
$XAPPLRESDIR/bitmaps/%N/%B	/usr/lib/X11/bitmaps/%N/%B
$XAPPLRESDIR/%L/bitmaps/%B	/usr/lib/X11/%L/bitmaps/%B
$XAPPLRESDIR/%l_%t/bitmaps/%B	/usr/lib/X11/%l_%t/bitmaps/%B
$XAPPLRESDIR/%l/bitmaps/%B	/usr/lib/X11/%l/bitmaps/%B
$XAPPLRESDIR/bitmaps/%B	/usr/lib/X11/bitmaps/%B
	/usr/include/X11/bitmaps/%B
$HOME/bitmaps/%B	$HOME/%B

If XAPPLRESDIR is not set, `XmGetPixmap`() searches the same paths, except that XAPPLRESDIR is replaced by HOME. These search paths are vendor-dependent and a vendor may use different directories for /usr/lib/X11 and /usr/include/X11. In the search paths, the image name is substituted for %B, the class name of the application is substituted for %N, the language string of the display is substituted for %L, the language component of the language string is substituted for %l, and the territory string is substituted for %t.

See Also

`XmDestroyPixmap`(1), `XmGetPixmapByDepth`(1), `XmInstallImage`(1), `XmUninstallImage`(1).

Name

XmGetPixmapByDepth – create and return a pixmap of the specified depth.

Synopsis

```
Pixmap XmGetPixmapByDepth (Screen   *screen,
                           char     *image_name,
                           Pixel    foreground,
                           Pixel    background,
                           int      depth)
```

Inputs

screen Specifies the screen on which the pixmap is to be drawn.
image_name Specifies the string name of the image used to make the pixmap.
foreground Specifies the foreground color that is combined with the image when it is a bitmap.
background Specifies the background color that is combined with the image when it is a bitmap.
depth Specifies the depth of the pixmap.

Returns

A pixmap on success or XmUNSPECIFIED_PIXMAP when the specified image_name cannot be found.

Availability

Motif 1.2 and later.

Description

`XmGetPixmapByDepth`() generates a pixmap, stores it in the pixmap cache, and returns its resource ID. Before the routine actually creates the pixmap, it checks the pixmap cache for a pixmap that matches the specified *image_name*, *screen*, *foreground*, *background*, and *depth*. If a match is found, the reference count for the pixmap is incremented and the resource ID for the pixmap is returned. If no pixmap is found, `XmGetPixmapByDepth`() checks the image cache for a image that matches the specified *image_name*. If a matching image is found, it is used to create the pixmap that is returned.

When no matches are found, `XmGetPixmapByDepth`() begins a search for an X10 or X11 bitmap file, using *image_name* as the filename. If a file is found, its contents are read, converted into an image, and cached in the image cache. Then, the image is used to generate a pixmap that is subsequently cached and returned. The depth of the pixmap is the specified *depth*. If *image_name* specifies a bitmap, the foreground and background colors are combined with the image. If no file is found, the routine returns XmUNSPECIFIED_PIXMAP.

Usage

XmGetPixmapByDepth() works just like XmGetPixmap() except that the depth of the pixmap can be specified. With XmGetPixmap(), the depth of the returned pixmap is the default depth of the screen. See XmGetPixmap() for an explanation of the search path that is used to find the image.

See Also

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

Name

XmGetPostedFromWidget – get the widget that posted a menu.

Synopsis

#include <Xm/RowColumn.h>

Widget XmGetPostedFromWidget (Widget *menu*)

Inputs

menu Specifies the menu widget.

Returns

The widget ID of the widget that posted the menu.

Description

`XmGetPostedFromWidget`() returns the widget from which the specified *menu* is posted. The value that is returned depends on the type of menu that is specified. For a PopupMenu, the routine returns the widget from which *menu* is popped up. For a PulldownMenu, the routine returns the RowColumn widget from which *menu* is pulled down. For cascading submenus, the returned widget is the original RowColumn widget at the top of the menu system. For tear-off menus in Motif 1.2 and later, `XmGetPostedFromWidget`() returns the widget from which the menu is torn off.

Usage

If an application uses the same menu in different contexts, it can use `XmGet-PostedFromWidget`() in an activate callback to determine the context in which the menu callback should be interpreted.

See Also

`XmRowColumn`(2), `XmPopupMenu`(2), `XmPulldownMenu`(2).

Name

XmGetScaledPixmap – create and return a scaled pixmap.

Synopsis

```
Pixmap XmGetScaledPixmap (Widget   widget,
                          char     *image_name,
                          Pixel    foreground,
                          Pixel    background,
                          int      depth,
                          double   scaling_ratio)
```

Inputs

widget Specifies a widget.

image_name Specifies the string name of the image used to make the pixmap.

foreground Specifies the foreground color that is combined with the image when it is a bitmap.

background Specifies the background color that is combined with the image when it is a bitmap.

depth Specifies the depth of the pixmap.

scaling_ratio Specifies a scaling ratio applied to the pixmap.

Returns

A pixmap on success or XmUNSPECIFIED_PIXMAP when the specified image_name cannot be found.

Availability

Motif 2.1 and later.

Description

`XmGetScaledPixmap`() is similar to `XmGetPixmapByDepth`() except that the returned pixmap is scaled.

Usage

widget is used to find a PrintShell by wandering up the widget hierarchy, and secondly to find a Screen on which to create the pixmap. If *scaling_ratio* is zero and an ancestral PrintShell is found, the ratio applied is given by

(printer resolution / default pixmap resolution)

where the default pixmap resolution is the XmNdefaultPixmapResolution resource of the PrintShell, and the printer resolution is fetched by the PrintShell using Xp extensions to communicate with the XPrint server. The default value of the PrintShell XmNdefaultPixmapResolution resource is 100.

At present, any resolution specified within the pixmap file itself is currently ignored, although it is intended that this should take precedence over any PrintShell setting.
Although otherwise fully documented, the function does not have a functional prototype in any of the supplied public headers.

See Also

`XmDestroyPixmap`(1), `XmGetPixmapByDepth`(1), `XmPrintShell`(2).

Name

XmGetSecondaryResourceData – retrieve secondary widget resource data.

Synopsis

Cardinal XmGetSecondaryResourceData (WidgetClass *widget_class*,
XmSecondaryResourceData *******secondary_data_return*)

Inputs

widget_class Specifies the widget class.

Outputs

secondary_data_return Returns an array of XmSecondaryResourceData pointers.

Returns

The number of secondary resource data structures associated with the widget class.

Availability

Motif 1.2 and later.

Description

`XmGetSecondaryResourceData`() provides access to the secondary widget resource data associated with a widget class. Some Motif widget classes have resources that are not accessible with the functions `XtGetResourceList`() and `XtGetConstraintResourceList`(). If the specified *widget_class* has secondary resources, `XmGetSecondaryResourceData`() provides descriptions of the resources in one or more data structures and returns the number such structures. If the *widget_class* does not have secondary resources, the routine returns 0 (zero) and the value of *secondary_data_return* is undefined.
If the *widget_class* has secondary resources, `XmGetSecondaryResource-Data()` allocates an array of pointers to the corresponding data structures. The application is responsible for freeing the allocated memory using `XtFree`(). The resource list in each structure (the value of the resources field), the structures, and the array of pointers to the structures all need to be freed.

Usage

`XmGetSecondaryResourceData`()[1] only returns the secondary resources for a widget class if the class has been initialized. You can initialize a widget class by creating an instance of the class or any of its subclass. VendorShell and Text are two Motif widget classes that have secondary resources. The two fields in the XmSecondaryResourceData structure that are of interest to an application are resources and num_resources. These fields contain a list of the secondary resources and the number of such resources.

Most applications do not need to query a widget class for the resources it supports. `XmGetSecondaryResourceData`() is intended to support interface builders and applications like *editres* that allow a user to view the available resources and set them interactively. Use `XtGetResourceList`() and `XtGetConstraintResourceList`*()* to get the regular and constraint resources for a widget class.

Example

The following code fragment shows the use of `XmGetSecondaryResourceData`() to print the names of the secondary resources of the VendorShell widget:

```
XmSecondaryResourceData *res; Cardinal num_res, i,
j;
if (num_res = XmGetSecondaryResourceData (vendor-
                                          Shell-
                                          WidgetCl
                                          ass,
                                          &res)) {
   for (i = 0; i < num_res; i++) {
      for (j = 0; j < res[i]->num_resources; j++) {
         printf ("%s\n", res[i]-
         >resources[j].resource_name);
      }
      XtFree ((char*) res[i]->resources);
      XtFree ((char*) res[i]);
   }
   XtFree ((char*) res);
}
```

1.Erroneously given as XmGetSecondaryResources() in 1st and 2nd edition.

Structures

The XmSecondaryResourceData structure is defined as follows:

```
typedef struct {
    XmResourceBaseProc   base_proc;
    XtPointer            client_data;
    String               name;
    String               res_class;
    XtResourceList       resources;
    Cardinal             num_resources;
}XmSecondaryResourceDataRec, *XmSecondaryResourceData;
```

See Also

`VendorShell`(2), `XmText`(2).

Name

XmGetTabGroup – get the tab group for a widget.

Synopsis

Widget XmGetTabGroup (Widget *widget*)

Inputs

widget Specifies the widget whose tab group is to be returned.

Returns

The widget ID of the tab group of widget.

Availability

Motif 1.2 and later.

Description

`XmGetTabGroup`() returns the widget ID of the widget that is the tab group for the specified widget. If *widget* is a tab group or a shell, the routine returns *widget*. If *widget* is not a tab group and no ancestor up to the nearest shell ancestor is a tab group, the routine returns the nearest shell ancestor. Otherwise, `XmGetTabGroup`() returns the nearest ancestor of *widget* that is a tab group.

Usage

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

See Also

`XmGetFocusWidget`(1), `XmGetVisibility`(1), `XmIsTraversable`(1), `XmProcessTraversal`(1), `XmManager`(2), `XmPrimitive`(2).

Name

XmGetTearOffControl – get the tear-off control for a menu.

Synopsis

#include <Xm/RowColumn.h>

Widget XmGetTearOffControl (Widget *menu*)

Inputs

menu Specifies the RowColumn widget whose tear-off control is to be returned.

Returns

The widget ID of the tear-off control or NULL if no tear-off control exists.

Availability

Motif 1.2 and later.

Description

`XmGetTearOffControl`() retrieves the widget ID of the widget that is the tear-off control for the specified *menu*. When the XmNtearOffModel resource of a RowColumn widget is set to XmTEAR_OFF_ENABLED for a PulldownMenu or a PopupMenu, the RowColumn creates a tear-off button for the menu. The tear-off button, which contains a dashed line by default, is the first element in the menu. When the button is activated, the menu is torn off. If the specified *menu* does not have a tear-off control, `XmGetTearOffControl`() returns NULL.

Usage

In Motif 1.2, a RowColumn that is configured as a PopupMenu or a PulldownMenu supports tear-off menus. When a menu is torn off, it remains on the screen after a selection is made so that additional selections can be made. The tear-off control is a button that has a Separator-like appearance. Once you retrieve the widget ID of the tear-off control, you can set resources to specify its appearance. You can specify values for the following resources: XmNbackground, XmNbackgroundPixmap, XmNbottomShadowColor, XmNforeground, XmNheight, XmNmargin, XmNseparatorType, XmNshadowThickness, and XmNtopShadowColor. You can also set these resources in a resource file by using the name of the control, which is TearOffControl.

See Also

`XmRepTypeInstallTearOffModelConverter`(1), `XmPopupMenu`(2), `XmPulldownMenu`(2), `XmRowColumn`(2), `XmSeparator`(2).

Name

XmGetVisibility – determine whether or not a widget is visible.

Synopsis

XmVisibility XmGetVisibility (Widget *widget*)

Inputs

widget Specifies the widget whose visibility state is to be returned.

Returns

XmVISIBILITY_UNOBSCURED if widget is completely visible, XmVISIBILITY_PARTIALLY_OBSCURED if widget is partially visible, XmVISIBILITY_FULLY_OBSCURED or if widget is not visible.

Availability

Motif 1.2 and later.

Description

`XmGetVisibility`() determines whether or not the specified *widget* is visible. The routine returns XmVISIBILITY_UNOBSCURED if the entire rectangular area of the widget is visible. It returns XmVISIBILITY_PARTIALLY_OBSCURED if a part of the rectangular area of the widget is obscured by its ancestors. `XmGetVisibility`() returns XmVISIBILITY_FULLY_OBSCURED if the widget is completely obscured by its ancestors or if it is not visible for some other reason, such as if it is unmapped or unrealized.

Usage

XmGetVisibility() provides a way for an application to find out the visibility state of a particular widget. This information can be used to help determine whether or not a widget is eligible to receive the keyboard focus. In order for a widget to receive the keyboard focus, it and all of its ancestors must not be in the process of being destroyed and they must be sensitive to input. The widget and its ancestors must also have their XmNtraversalOn resources set to True. If the widget is viewable, which means that it and its ancestors are managed, mapped, and realized and some part of the widget is visible, then the widget is eligible to receive the keyboard focus. A fully-obscured widget is not eligible to receive the focus unless part of it is within the work area of a ScrolledWindow with an XmNscrollingPolicy of XmAUTOMATIC that has an XmNtraverseObscuredCallback.

Structures

XmVisibility is defined as follows:

```
typedef enum {
    XmVISIBILITY_UNOBSCURED,
    XmVISIBILITY_PARTIALLY_OBSCURED,
    XmVISIBILITY_FULLY_OBSCURED
} XmVisibility;
```

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1), `XmIsTraversable`(1), `XmProcessTraversal`(1), `XmManager`(2), `XmScrolledWindow`(2).

Name

XmGetXmDisplay – get the Display object for a display.

Synopsis

#include <Xm/Display.h>

Widget XmGetXmDisplay (Display **display*)

Inputs

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

Returns

The Display object for the display.

Availability

Motif 1.2 and later.

Description

`XmGetXmDisplay`() retrieves the Display object for the specified *display.*

Usage

In Motif 1.2, the Display object stores display-specific information for use by the toolkit. An application has a Display object for each display it accesses. When an application creates its first shell on a display, typically by calling `XtAppInitialize`() or `XtAppCreateShell`(), a Display object is created automatically. There is no way to create a Display independently. Use `XmGetXmDisplay`() to get the ID of the Display object, so that you can use `XtGetValues`() and `XtSetValues`() to access and modify Display resources.

See Also

`XmDisplay`(2), `XmScreen`(2).

Name

XmGetXmScreen – get the Screen object for a screen.

Synopsis

Widget XmGetXmScreen (Screen **screen*)

Inputs

screen Specifies a screen on a display; returned by `XtScreen`().

Returns

The Screen object for the screen.

Availability

Motif 1.2 and later.

Description

`XmGetXmScreen`() retrieves the Screen object for the specified *screen*.

Usage

In Motif 1.2, the Screen object stores screen-specific information for use by the toolkit. An application has a Screen object for each screen that it accesses. When an application creates its first shell on a screen, typically by calling `XtAppInitialize`() or `XtAppCreateShell`(), a Screen object is created automatically. There is no way to create a Screen independently. Use `XmGetXmScreen`() to get the ID of the Screen object, so that you can use `XtGetValues`() and `XtSetValues`() to access and modify Screen resources.

See Also

`XmDisplay`(2), `XmScreen`(2).

Name

XmIm – introduction to input methods.

Synopsis

Public Header:
<Xm/XmIm.h>

Functions/Macros:
`XmImCloseXIM`(), `XmImFreeXIC`(), `XmImGetXIC`(), `XmImGetXIM`(),
`XmImMbLookupString`(), `XmImMbResetIC`(), `XmImRegister`(),
`XmImSetFocusValues`(), `XmImSetValues`(), `XmImSetXIC`(),
`XmImUnregister`(), `XmImUnsetFocus`(), `XmImVaSetFocusValues`(),
`XmImVaSetValues`()

Availability

Motif 1.2 and later.

Description

Many languages are ideographic, and have considerably more characters than there are keys on the keyboard: the Ascii keyboard was not originally designed for languages that are not based upon the Latin alphabet. For such languages, in order to provide a mapping between the alphabet and the keyboard, it is necessary to represent particular characters by a key sequence rather than a single keystroke. An input method is the means by which X maps between the characters of the language, and the representative key sequences. The most common use of an input method is in implementing language-independent text widget input. As the user types the key sequences, the input method displays the actual keystrokes until the sequence completes a character, when the required character is displayed in the text widget. The process of composing a character from a key sequence is called pre-editing.

In order to facilitate pre-editing, the input method may maintain several areas on the screen: a status area, a pre-edit area, and an auxiliary area. The status area is an output-only window which provides feedback on the interaction with the input method. The pre-edit area displays the keyboard sequence as it is typed. The auxiliary area is used for popup menus, or for providing customized controls required by the particular input method. The location of the pre-edit area is determined by the XmNpreeditType resource of VendorShell. The value OnTheSpot displays the key sequence as it is typed into the destination text widget itself. OverTheSpot superimposes an editing window over the top of the text widget. OffTheSpot creates a dedicated editing window, usually at the bottom of the dia-

log. Root uses a pre-edit window which is a child of the root window of the display.
To control the interaction between the application and the input method, X defines a structure called an input context, which the programmer can fetch and manipulate where the need arises. Each widget registered with the input method has an associated input context, which may or may not be shared amongst the registered widgets. Motif extends the mechanisms provided by the lower level X libraries, and provides a caching mechanism whereby input contexts are shared between widgets.

Usage

Input methods are usually supplied by the vendors of the hardware, and the application generally connects to the input method without the need for any special coding by the programmer. The Motif widgets are fully capable of connecting to an input method when required, and although Motif provides a functional interface to enable the programmer to interact with an input method, the interface is not required for the Motif widgets. The exceptions are where the programmer is writing new widgets, or where internationalized input is required for the DrawingArea.

`XmImRegister`() registers a widget with an input method. `XmImSetValues`() manipulates an input context by registering callbacks which respond to specific states. `XmImSetFocusValues`() is similar, except that after the input context has been modified, the focus is reset to the widget providing the input. `XmImMbLookupString`() performs the necessary key sequence to character translation on behalf of the input widget. `XmImUnRegister`() unregisters the widget with the input method. Typically, `XmImRegister`() is called within the Initialize method of a widget, `XmImUnRegister`() is called by the Destroy method, and `XmImMbLookupString`() is called within an action or callback routine of the widget in response to an event. These are the primary functions which a programmer may need to call, and are all that are required to implement internationalized input for the Motif text widget.

Note that an input method does not need to support all styles of XmNpreeditType.

See Also

`XmImCloseXIM`(1), `XmImFreeXIC`(1), `XmImGetXIM`(1), `XmImGetXIC`(1), `XmImMbLookupString`(1), `XmImMbResetIC`(1), `XmImRegister`(1), `XmImSetFocusValues`(1), `XmImSetValues`(1), `XmImSetXIC`(1), `XmImUnregister`(1), `XmImUnsetFocus`(1), `XmImVaSetFocusValues`(1), `XmImVaSetValues(1).`

Name

XmImCloseXIM – close all input contexts.

Synopsis

#include <Xm/XmIm.h>

void XmImCloseXIM (Widget *widget*)

Inputs

widget Specifies a widget used to determine the display connection.

Availability

Motif 2.0 and later.

Description

`XmImCloseXIM`() is a convenience function which closes all input contexts associated with the current input method. The *widget* parameter is used to identify the XmDisplay object of the application.

Usage

`XmImCloseXIM`() uses the *widget* parameter to deduce the input method associated with the XmDisplay object. The application's connection to the input method is closed, and all widgets which are registered with any input context associated with the input method are unregistered. In order to close the input context associated with a single widget, rather than closing down all connections, use XmImUnregister().

The Motif widgets internally register and unregister themselves with the input manager using XmImRegister() and `XmImUnregister`() as required. The VendorShell calls `XmImCloseXIM`() within its Destroy method once the last VendorShell is destroyed in order to clean up the connection to the input method. An application which dynamically switches between input methods in a multi-language application may need to invoke `XmImCloseXIM`() because Motif only supports a single input method at any given instance. Application programmers will not normally need to use `XmImCloseXIM`() directly.

See Also

`XmImRegister`(1), `XmImUnregister`(1), `XmIm`(1).

Name

XmImFreeXIC – free an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImFreeXIC (Widget *widget*, XIC *xic*)

Inputs

widget	Specifies a widget from which the input context registry is deduced.
xic	Specifies the input context which is to be freed.

Availability

Motif 2.0 and later.

Description

`XmImFreeXIC`() is a convenience function which unregisters all widgets associated with the input context *xic*, and then frees the input context.

Usage

`XmImFreeXIC`() uses the *widget* parameter to deduce an ancestral VendorShell, from which the X input context registry is found. All widgets associated with the input context *xic* within the registry are unregistered, and the input context is freed.

See Also

`XmImGetXIC`(1), `XmImRegister`(1). `XmImSetXIC`(1), `XmImUnregister`(1), `XmIm`(1).

Name

XmImGetXIC – create an input context for a widget.

Synopsis

#include <Xm/XmIm.h>

XIC XmImGetXIC (Widget *widget*, XmInputPolicy *input_policy*, ArgList *arglist*, Cardinal *argcount*)

Inputs

widget	Specifies a widget for which the input context is required.
input_policy	Specifies the policy for creating input contexts.
arglist	Specifies a list of arguments consisting of name/value pairs.
argcount	Specifies the number of arguments in arglist.

Returns

The input context associated with widget.

Availability

Motif 2.0 and later.

Description

`XmImGetXIC`() creates and registers a new input context for a widget, depending upon the *input_policy*. If *input_policy* is XmPER_WIDGET, a new input context is created for the widget. If the value is XmPER_SHELL, a new input context is created only if an input context associated with the ancestral shell of *widget* does not already exist, otherwise the widget is registered with the existing input context. If the policy is XmINHERIT_POLICY, the input policy is inherited by taking the value of the XmNinputPolicy resource from the nearest ancestral VendorShell. The set of attributes for the input context is specified through the resource list *arglist*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *argcount*. The name/value pairs are passed through to the function `XCreateIC`() if the input context is created. `XmImGetXIC`() returns either the input context which is newly created if the input policy is XmPER_WIDGET, otherwise it returns the shared context.

Usage

In Motif 1.2, the supported attributes for configuring the created input context are XmNbackground, XmNforeground, XmNbackgroundPixmap, XmNspotLocation, XmNfontList, and XmNarea.

In Motif 2.0 and later, the list is extended to include XmNpreeditCaretCallback, XmNpreeditDoneCallback, XmNpreeditDrawCallback, and XmNpreeditStartCallback resources.

You are referred to the `XCreateIC`() entry within the Xlib Reference Manual for the interpretation of each of the resource types. The function allocates storage associated with the created input context, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmImFreeXIC`().

Structures

The enumerated type XmInputPolicy has the following possible values:

XmINHERIT_POLICY
XmPER_WIDGET
XmPER_SHELL

See Also

`XmImFreeXIC`(1), `XmImSetXIC`(1), `XmIm`(1).

Name

XmImGetXIM – retrieve the input method for a widget.

Synopsis

#include <Xm/XmIm.h>

XIM XmImGetXIM (Widget *widget*)

Inputs

widget Specifies a widget registered with the input manager.

Returns

The input method associated with *widget*.

Availability

Motif 1.2 and later.

Description

`XmImGetXIM`() returns a pointer to an opaque data structure which represents the input method which the input manager has opened for the specified widget.

Usage

Widgets are normally registered with the input manager through a call to `XmIm-Register`(). If no input method is associated with the *widget*, the procedure uses any specified XmNinputMethod resource of the nearest ancestral VendorShell in order to open an input method. If the resource is NULL, the input method associated with the current locale is opened. If no input method can be opened, the function returns NULL.

`XmImGetXIM`() allocates storage for the opaque data structure which is returned, and it is the responsibility of the programmer to reclaim the space by a call to `XmImCloseXIM`() at a suitable point. `XmImGetXIM`() is not a procedure which an application programmer needs to use: the routine is of more use to the programmer of new widgets.

See Also

`XmImRegister`(1), `XmImCloseXIM`(1), `XmIm`(1).

Name

XmImMbLookupString – retrieve a composed string from an input method.

Synopsis

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

int XmImMbLookupString ( Widget             widget,
                         XKeyPressedEvent   *event,
                         char               *buffer,
                         int                num_bytes,
                         KeySym             *keysym,
                         int                *status)
```

Inputs

widget	Specifies a widget registered with the input manager.
event	Specifies a key press event.
num_bytes	Specifies the length of the buffer array.

Outputs

buffer	Returns the composed string.
keysym	Returns any keysym associated with the input keyboard event.
status	Returns the status of the lookup.

Returns

The length of the composed string in bytes.

Availability

Motif 1.2 and later.

Description

`XmImMbLookupString`() translates an *event* into a composed character, and/or a keysym, using the input context associated with a given *widget*. Any composed string which can be deduced from the *event* is placed in *buffer*; the composed string consists of multi-byte characters in the encoding of the locale of the input context. If a keysym is associated with the *event*, this is returned at the address specified by *keysym*. The function returns the number of bytes placed into *buffer*.

Usage

A widget is registered with an input method through the function `XmImRegister`(). If no input context is associated with the *widget*, the function uses `XLookupString`() to map the key *event* into composed text. Otherwise the function calls `XmbLookupString`() with the input context as the first parameter. If the programmer is not interested in keysym values, a NULL value can be passed as the *keysym* parameter. `XmImMbLookupString`() places into *buffer* any composed character string associated with the key event: if the event at the given point in the input sequence does not signify a unique character in the language of the current locale, the function returns zero: subsequent key events may be required before a character is composed.

Structures

The possible values returned in *status* are the same as those returned from `XmbLookupString`(): you are referred to the Xlib Reference Manual for a full description and interpretation of the values.

```
XBufferOverflow     /* buffer size insufficient to hold composed sequence */
XLookupNone         /* no character sequence matching the input exists    */
XLookupChars        /* input characters were composed                     */
XLookupKeysym       /* input is keysym rather than composed character     */
XLookupBoth         /* both a keysym and composed character are returned */
```

See Also

`XmImRegister`(1), `XmIm`(1).

Name

XmImMbResetIC – reset an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImMbResetIC (Widget *widget*, char ***mb_text*)

Inputs

widget Specifies a widget registered with the Input Manager.

Outputs

mb_text Returns pending input on the input context.

Availability

Motif 2.0 and later.

Description

`XmImMbResetIC`() resets the input context associated with a widget.

Usage

`XmImMbResetIC`() is a convenience function which resets an input context to the initial state. The function is no more than a wrapper onto the function `XmbResetIC`(), which clears the pre-edit area and updates the status area of the input context. The return value of `XmbResetIC`() is placed into the address specified by *mb_text*. This data is implementation dependent, and may be NULL. If data is returned, the programmer is responsible for freeing it by calling `XFree`().

See Also

`XmImRegister`(1), `XmIm`(1).

Name

XmImRegister – register a widget with an Input Manager.

Synopsis

#include <Xm/XmIm.h>

void XmImRegister (Widget *widget*, unsigned int *reserved*)

Inputs

widget	Specifies a widget to register with the input manager.
reserved	This parameter is current unused.

Availability

Motif 1.2 and later.

Description

`XmImRegister`() is a convenience function which registers a widget with the input manager to establish a connection to the current input method. The function is called when an application needs to specially arrange for internationalized input to a widget.

Usage

The Motif widgets internally register themselves with the input manager as required. Only a programmer who is writing a new widget, or who requires internationalized input for the DrawingArea needs to call `XmImRegister`() directly. If the VendorShell ancestor containing the *widget* already has an associated input context, the function simply returns. Otherwise, the XmNinputPolicy resource of the nearest VendorShell ancestor is fetched to determine whether to share an existing input context. The function opens an input method by inspecting the XmNinputMethod resource of the VendorShell. If the resource is NULL, a default input method is opened using information from the current locale. `XmImRegister`() should not be called twice using the same *widget* parameter without unregistering the widget from the input method first.

The programmer is responsible for closing down the connection to the input method by calling `XmImUnregister`(). The Destroy method of the widget is an appropriate place to call this.

See Also

`XmImUnregister`(1), `XmIm`(1).

Name

XmImSetFocusValues – set the values and focus for an input context.

Synopsis

#include <Xm/XmIm.h>

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

Inputs

widget	Specifies a widget registered with the input manager.
arglist	Specifies a list of resources consisting of name/value pairs.
argcount	Specifies the number of arguments in arglist.

Availability

Motif 1.2 and later.

Description

`XmImSetFocusValues`() notifies the input manager that a widget has received the input focus. If the previous values of the input context associated with the widget do not allow the context to be reused, the old context is unregistered, and a new one registered with the widget.

Usage

`XmImSetFocusValues`() is identical in all respects to `XmImSetValues`(), except that after the input context has been reset, the focus window attribute of the input context is set to the window of the input *widget*.

The Motif widgets invoke `XmImSetFocusValues`() as and when required. For example, the Text and TextField widgets automatically invoke XmImSetFocusValues() in response to FocusIn and EnterNotify events. A programmer who is implementing internationalized input for a DrawingArea or creating a new widget may need to call this function when the widget receives the input focus.

See Also

`XmImRegister`(1), `XmImSetValues`(1), `XmIm`(1).

Name

XmImSetValues – set the values for an input context.

Synopsis

#include <Xm/XmIm.h>

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

Inputs

widget	Specifies a widget registered with the Input Manager.
arglist	Specifies a list of resources consisting of name/value pairs.
argcount	Specifies the number of arguments in arglist.

Availability

Motif 1.2 and later.

Description

`XmImSetValues`() sets the attributes for the input context associated with the specified *widget*. The set of attributes to be modified is specified through the resource list *arglist*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *argcount*.

Usage

`XmImSetValues`() is a convenience routine which invokes `XSetICValues`() in order to configure an input context. You are referred to the Xlib Reference Manual for the set of attributes supported by `XSetICValues`(), and for their interpretation.

The Motif widgets invoke `XmImSetValues`() as and when required. For example, the Text and TextField widgets automatically invoke `XmImSetValues`() when the widget is resized or the font changed. A programmer who is implementing internationalized input for a DrawingArea or creating a new widget may need to call this function when, for example, the widget needs to reconfigure the spot location.

See Also

`XmImSetFocusValues`(1), `XmImRegister`(1), `XmIm`(1).

Name

XmImSetXIC – register a widget with an existing input context.

Synopsis

#include <Xm/XmIm.h>

XIC XmImSetXIC (Widget *widget*, XIC *xic*)

Inputs

widget	Specifies a widget to be registered with the input context.
xic	Specifies an input context where the widget is to be registered.

Returns

The input context where the widget is registered.

Availability

Motif 2.0 and later.

Description

`XmImSetXIC`() is a convenience function which registers a *widget* with an input context. If the *widget* is registered with another input context, the *widget* is firstly unregistered with that context. The widget is then registered with the input context *xic*. If *xic* is NULL, the function creates a new input context and registers the widget with it. The function returns the input context where the widget is registered.

Usage

`XmImSetXIC`() allocates storage when it creates a new input context, and it is the responsibility of the programmer to free the space at an appropriate point by calling `XmImFreeXIC`().

See Also

`XmImFreeXIC`(1), `XmImRegister`(1), `XmIm`(1).

Name

XmImUnregister – unregister the input context for a widget.

Synopsis

#include <Xm/XmIm.h>

void XmImUnregister (Widget *widget*)

Inputs

widget Specifies a widget whose input context is to be unregistered.

Availability

Motif 1.2 and later.

Description

`XmImUnregister`() is a convenience function which unregisters the input context associated with a given *widget*. The function is the inverse of `XmImRegister`(), which is called when an application needs to specially arrange for internationalized input to a widget.

Usage

The Motif widgets internally register themselves with the input manager as required. Only a programmer who is writing a new widget, or who requires internationalized input for the DrawingArea needs to call `XmImRegister`() directly. Where `XmImRegister`() has been called by the application, it is the responsibility of the programmer to also call `XmImUnregister`(), usually within the `Destroy`() method of the widget for which internationalized input is required. `XmImUnregister`() uses the *widget* parameter to deduce the input method associated with a display connection. Any input context associated with the input method is unregistered.

See Also

`XmImRegister`(1), `XmIm`(1).

Name

XmImUnsetFocus – unset focus for input context.

Synopsis

#include <Xm/XmIm.h>

void XmImUnsetFocus (Widget *widget*)

Inputs

widget Specifies a widget which has lost the input focus.

Availability

Motif 1.2 and later.

Description

`XmImUnsetFocus`() notifies the input manager that a widget has lost the input focus.

Usage

`XmImUnsetFocus`() is a convenience routine which invokes `XUnsetICFocus`() using the input context associated with the specified *widget*. The input method is notified that no more input is expected from the widget.

The Motif widgets invoke `XmImUnsetFocus`() as and when required. For example, the Text and TextField widgets automatically invoke `XmImUnsetFocus`()[1] in response to FocusOut and LeaveNotify events. A programmer who is implementing internationalized input for a DrawingArea or creating a new widget may need to call this function when the widget loses the input focus.

See Also

`XmImSetFocusValues`(1), `XmImVaSetFocusValues`(1), `XmIm`(1).

1.Erroneously given as XmUnsetFocus() in 2nd edition.

Name

XmImVaSetFocusValues – set the values and focus for an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImVaSetFocusValues (Widget *widget*,....,NULL)

Inputs

widget	Specifies a widget registered with the Input Manager.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Availability

Motif 1.2 and later.

Description

`XmImVaSetFocusValues`() notifies the input manager that a *widget* has received the input focus. If the previous values of the input context associated with the *widget* do not allow the context to be reused, the old context is unregistered, and a new one registered with the widget.

Usage

`XmImVaSetFocusValues`() is simply a convenience routine with a variable length argument list which constructs internal arglist and argcount parameters to a `XmImSetFocusValues`() call.

See Also

`XmImSetFocusValues`(1).

Name

XmImVaSetValues – set the values for an input context.

Synopsis

#include <Xm/XmIm.h>

void XmImVaSetValues (Widget *widget*,...,NULL)

Inputs

widget	Specifies a widget registered with the Input Manager.
...,NULL	A NULL-terminated variable-length list of resource name/value pairs.

Availability

Motif 1.2 and later.

Description

`XmImVaSetValues`()[1] sets the attributes for the input context associated with the specified *widget*.

Usage

`XmImVaSetValues`() is simply a convenience routine with a variable length argument list which constructs internal arglist and argcount parameters to a `XmImSetValues`() call.

See Also

`XmImSetValues`(1).

1.Erroneously given as XmImSetValues() in 2nd edition.

Name

XmInstallImage – install an image in the image cache.

Synopsis

Boolean XmInstallImage (XImage **image*, char **image_name*)

Inputs

image	Specifies the image to be installed.
image_name	Specifies the string name of the image.

Returns

True on success or False if *image* or *image_name* is NULL or *image_name* duplicates an image name already in the cache.

Description

`XmInstallImage`() installs the specified *image* in the image cache. The *image* can later be used to create a pixmap. When the routine installs the image, it does not make a copy of the image, so an application should not destroy the image until it has been uninstalled. The routine also expands the resource converter that handles images so that *image_name* can be used in a resource file. In order to allow references from a resource file, `XmInstallImage`() must be called to install an image before any widgets that use the image are created.

Usage

An application can use `XmInstallImage`() to install and cache images, so that the images can be shared throughout the application. Once an image is installed, it can be used to create a pixmap with `XmGetPixmap`(). The toolkit provides the following pre-installed images that can be referenced in a resource file or used to create a pixmap:

Image Name	Image Description
background	Solid background tile
25_foreground	A 25% foreground, 75% background tile
50_foreground	A 50% foreground, 50% background tile
75_foreground	A 75% foreground, 25% background tile
horizontal_tile	Horizontal lines tile, in Motif 1.2.3 and later.
vertical_tile	Vertical lines tile, in Motif 1.2.3 and later.
horizontal	As horizontal_tile: maintained for 1.2.2 compatibility.
vertical	As vertical_tile: maintained for 1.2.2 compatibility.
slant_right	Right slanting lines tile
slant_left	Left slanting lines tile

Image Name	Image Description
menu_cascade	An arrow pointing to the right, in Motif 2.0 and later.
menu_cascade_rtol	An arrow pointing to the left, in Motif 2.0 and later.
menu_checkmark	A tick mark, in Motif 2.0 and later.
menu_dash	A horizontal line, in Motif 2.0 and later.
collapsed	A filled arrow pointing to the right, in Motif 2.0 and later.
collapsed_rtol	A filled arrow pointing to the left, in Motif 2.0 and later.
expanded	A filled arrow pointing downwards, in Motif 2.0 and later.

Example

You might use the following code to define and install an image:

```
#define bitmap_width 16
#define bitmap_height 16

static char bitmap_bits[] = {
    0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
    0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
    0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF,
    0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF
};

static XImage ximage = {
    bitmap_width,       /* width */
    bitmap_height,      /* height */
    0,                  /* xoffset */
    XYBitmap,           /* format */
    bitmap_bits,        /* data */
    MSBFirst,           /* byte_order */
    8,                  /* bitmap_unit */
    LSBFirst,           /* bitmap_bit_order */
    8,                  /* bitmap_pad */
    1,                  /* depth */
    2,                  /* bytes_per_line */
    NULL                /* obdata */
};
...
XmInstallImage (&ximage, "image_name");
...
```

See Also

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

Name

XmInternAtom – return an atom for a given property name string.

Synopsis

#include <Xm/AtomMgr.h>

Atom XmInternAtom (Display *_display_, String _name_, Boolean _only_if_exists_)

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
name	Specifies the string name of the property for which you want the atom.
only_if_exists	Specifies a Boolean value that indicates whether or not the atom is created if it does not exist.

Returns

An atom on success or None.

Availability

In Motif 2.0 and later, `XInternAtom`() is preferred.

Description

`XmInternAtom`() returns the atom that corresponds to the given property *name*. This routine works like Xlib's `XInternAtom`() routine, but the Motif routine provides the added feature of client-side caching. If no atom exists with the specified *name* and *only_if_exists* is True, `XmInternAtom`() does not create a new atom; it simply returns None. If *only_if_exists* is False, the routine creates the atom and returns it.

Usage

An atom is a number that identifies a property. Properties also have string names. XmInternAtom() returns the atom associated with a property if it exists, or it may create the atom if it does not exist. The atom remains defined even after the client that defined it has exited. An atom does not become undefined until the last connection to the X server closes. Predefined atoms are defined in <X11/Xatom.h> and begin with the prefix XA_. Predefined atoms do not need to be interned with `XmInternAtom`().

In Motif 2.0 and later, `XmInternAtom`() is no more than a convenience routine which calls `XInternAtom`(). While `XmInternAtom`() is not yet officially obsolete, `XInternAtom`() is to be preferred.

See Also

`XmGetAtomName`(1).

Name

XmIsMotifWMRunning – check whether the Motif Window Manager (*mwm*) is running.

Synopsis

Boolean XmIsMotifWMRunning (*Widget shell*)

Inputs

shell *Specifies the shell widget whose screen is queried.*

Returns

True if *mwm* is running or False *otherwise.*

Description

`XmIsMotifWMRunning`() checks for the presence of the _MOTIF_WM_INFO property on the root window of the screen of the specified *shell* to determine whether the Motif Window Manager (*mwm*) is running on the screen.

Usage

mwm defines additional types of communication between itself and client programs. This communication is optional, so an application should not depend on the communication or the presence of *mwm* for any functionality. `XmIsMotifWMRunning`() allows an application to check if *mwm* is running and act accordingly.

See Also

`mwm`(4).

Name

XmIs*Object* – determine whether a widget is a subclass of a class.

Synopsis

```
#include <Xm/Gadget.h>
Boolean XmIsGadget (Widget widget)

#include <Xm/Manager.h>
Boolean XmIsManager (Widget widget)

#include <Xm/Primitive.h>
Boolean XmIsPrimitive (Widget widget)

#include <Xm/ArrowB.h>
Boolean XmIsArrowButton (Widget widget)

#include <Xm/ArrowBG.h>
Boolean XmIsArrowButtonGadget (Widget widget)

#include <Xm/BulletinB.h>
Boolean XmIsBulletinBoard (Widget widget)

#include <Xm/CascadeB.h>
Boolean XmIsCascadeButton (Widget widget)

#include <Xm/CascadeBG.h>
Boolean XmIsCascadeButtonGadget (Widget widget)

#include <Xm/ComboBox.h>
Boolean XmIsComboBox (Widget widget)

#include <Xm/Command.h>
Boolean XmIsCommand (Widget widget)

#include <Xm/Container.h>
Boolean XmIsContainer (Widget widget)

#include <Xm/DialogS.h>
Boolean XmIsDialogShell (Widget widget)

#include <Xm/Display.h>
Boolean XmIsDisplay (Widget widget)

#include <Xm/DragC.h>
Boolean XmIsDragContext (Widget widget)

#include <Xm/DragIcon.h>
Boolean XmIsDragIconObjectClass (Widget widget)
```

#include <Xm/DrawingA.h>
Boolean XmIsDrawingArea (Widget widget)

#include <Xm/DrawnB.h>
Boolean XmIsDrawnButton (Widget widget)

#include <Xm/DropSMgr.h>
Boolean XmIsDropSiteManager (Widget widget)

#include <Xm/DropTrans.h>
Boolean XmIsDropTransfer (Widget widget)

#include <Xm/FileSB.h>
Boolean XmIsFileSelectionBox (Widget widget)

#include <Xm/Form.h>
Boolean XmIsForm (Widget widget)

#include <Xm/Frame.h>
Boolean XmIsFrame (Widget widget)

#include <Xm/GrabShell.h>
Boolean XmIsGrabShell (Widget widget)

#include <Xm/IconG.h>
Boolean XmIsIconGadget (Widget widget)

#include <Xm/Label.h>
Boolean XmIsLabel (Widget widget)

#include <Xm/LabelG.h>
Boolean XmIsLabelGadget (Widget widget)

#include <Xm/List.h>
Boolean XmIsList (Widget widget)

#include <Xm/MainW.h>
Boolean XmIsMainWindow (Widget widget)

#include <Xm/MenuShell.h>
Boolean XmIsMenuShell (Widget widget)

#include <Xm/MessageB.h>
Boolean XmIsMessageBox (Widget widget)

#include <Xm/Notebook.h>
Boolean XmIsNotebook (Widget widget)

#include <Xm/PanedW.h>
Boolean XmIsPanedWindow (Widget widget)

#include <Xm/PrintS.h>
Boolean XmIsPrintShell (Widget widget)

#include <Xm/PushB.h>
Boolean XmIsPushButton (Widget widget)

#include <Xm/PushBG.h>
Boolean XmIsPushButtonGadget (Widget widget)

#include <Xm/RowColumn.h>
Boolean XmIsRowColumn (Widget widget)

#include <Xm/Scale.h>
Boolean XmIsScale (Widget widget)

#include <Xm/Screen.h>
Boolean XmIsScreen (Widget widget)

#include <Xm/ScrollBar.h>
Boolean XmIsScrollBar (Widget widget)

#include <Xm/ScrolledW.h>
Boolean XmIsScrolledWindow (Widget widget)

#include <Xm/SelectioB.h>
Boolean XmIsSelectionBox (Widget widget)

#include <Xm/Separator.h>
Boolean XmIsSeparator (Widget widget)

#include <Xm/SeparatoG.h>
Boolean XmIsSeparatorGadget (Widget widget)

#include <Xm/Text.h>
Boolean XmIsText (Widget widget)

#include <Xm/TextF.h>
Boolean XmIsTextField (Widget widget)

#include <Xm/ToggleB.h>
Boolean XmIsToggleButton (Widget widget)

#include <Xm/ToggleBG.h>
Boolean XmIsToggleButtonGadget (Widget widget)

#include <Xm/VendorS.h>
Boolean XmIsVendorShell (Widget widget)

Inputs

widget Specifies the widget ID of the widget whose class is to be checked.

Returns

True if widget is of the specified class or False otherwise.

Availability

`XmIsDisplay`(), `XmIsDragContext`(), `XmIsDragIconObject-Class`(), `XmIsDropSiteManager`(), `XmIsDropTransfer`(), and `XmIsScreen`() are only available in Motif 1.2 and later.

`XmIsComboBox`(), `XmIsContainer`(), `XmIsNotebook`(), `XmIsIcon-Gadget`(), and `XmIsGrabShell`() are available in Motif 2.0 and later.

`XmIsPrintShell`() is available in Motif 2.1. Note that although the SpinBox class is available in Motif 2.0, and the SimpleSpinBox class in Motif 2.1, neither `XmIsSpinBox`() nor `XmIsSimpleSpinBox`() are defined. [1]

Description

The XmIs*() routines are macros that check the class of the specified widget. The macros returns True if widget is of the specified class or a subclass of the specified class. Otherwise, the macros return False.

Usage

An application can use the XmIs*() macros to check the class of a particular widget. All of the macros use `XtIsSubclass`() to determine the class of the widget.

Example

The missing macro `XmIsSpinBox`() could be defined as follows:

```
#include <Xm/SpinB.h>
#ifndef   XmIsSpinBox
#define   XmIsSpinBox(w)  XtIsSubclass(w, xmSpinBoxWidgetClass)
#endif /* XmIsSpinBox */
```

1.Be warned that certain platforms, although they ship the PrintShell headers, do not compile the component into the native Motif toolkit. Sun Solaris is a case in point.

See Also

XmCreateObject(1), VendorShell(2), XmArrowButton(2), XmArrowButtonGadget(2), XmBulletinBoard(2), XmCascadeButton(2), XmCascadeButtonGadget(2), XmComboBox(2), XmCommand(2), XmContainer(2), XmDialogShell(2), XmDisplay(2), XmDragContext(2), XmDragIcon(2), XmDrawingArea(2), XmDrawnButton(2), XmDropSite(2), XmDropTransfer(2), XmFileSelectionBox(2), XmForm(2), XmFrame(2), XmGadget(2), XmGrabShell(2), XmIconGadget(2), XmLabel(2), XmLabelGadget(2), XmList(2), XmMainWindow(2), XmManager(2), XmMenuShell(2), XmMessageBox(2), XmNotebook(2), XmPanedWindow(2), XmPrimitive(2), XmPrintShell(2), XmPushButton(2), XmPushButtonGadget(2), XmRowColumn(2), XmScale(2), XmScreen(2), XmScrollBar(2), XmScrolledWindow(2), XmSelectionBox(2), XmSeparator(2), XmSeparatorGadget(2), XmSpinBox(2), XmSimpleSpinBox(2), XmText(2), XmTextField(2), XmToggleButton(2), XmToggleButtonGadget(2).

Name

XmIsTraversable – determine whether or not a widget can receive the keyboard focus.

Synopsis

Boolean XmIsTraversable (Widget *widget*)

Inputs

widget Specifies the widget whose traversability state is to be returned.

Returns

True if widget is eligible to receive the keyboard focus or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmIsTraversable`() determines whether or not the specified *widget* can receive the keyboard focus. The routine returns True if the *widget* is eligible to receive the keyboard focus; otherwise it returns False.

Usage

In order for a widget to receive the keyboard focus, it and all of its ancestors must not be in the process of being destroyed and they must be sensitive to input. The widget and its ancestors must also have their XmNtraversalOn resources set to True. If the widget is viewable, which means that it and its ancestors are managed, mapped, and realized and some part of the widget is visible, then the widget is eligible to receive the keyboard focus. A fully-obscured widget is not eligible to receive the focus unless part of it is within the work area of a ScrolledWindow with an XmNscrollingPolicy of XmAUTOMATIC that has an XmNtraverseObscuredCallback.

Primitive widgets and gadgets can receive the keyboard focus, while most manager widgets cannot, even if they have traversable children. However, some managers may be eligible to receive the keyboard focus under certain conditions. For example, a DrawingArea can receive the keyboard focus if it meets the conditions above and it does not have any children with the XmNtraversalOn resource set to True.

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1), `XmGetVisibility`(1), `XmProcessTraversal`(1), `XmManager`(2), `XmScrolledWindow`(2).

Name

XmListAddItem, XmListAddItems – add an item/items to a list.

Synopsis

#include <Xm/List.h>

void XmListAddItem (Widget *widget*, XmString *item*, int *position*)
void XmListAddItems (Widget *widget*, XmString **items*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be added.
items	Specifies a list of items that are to be added.
item_count	Specifies the number of items to be added.
position	Specifies the position at which to add the new item(s).

Description

`XmListAddItem`() inserts the specified *item* into the list, while `XmListAddItems`() inserts the specified list of *items*. If *item_count* is smaller than the number of items, only the first *item_count* items of the array are added. The *position* argument specifies the location of the new item(s) in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. An inserted item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListAddItem`() and `XmListAddItems`() are convenience routines that allow you to add items to a list. The routines add items to the list by internally manipulating the arrays of compound strings specified by the XmNitems, XmNitemCount, XmNselectedItems, and XmNselectedItemCount resources. If an item being added to the list duplicates an item that is already selected, the new item appears as selected. You should only use these routines if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to add items with these routines, you have to create a compound string for each item.

See Also

`XmListAddItemUnselected`(1), `XmListReplaceItems`(1), `XmListReplaceItemsPos`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplacePositions`(1), `XmList`(2).

Name

XmListAddItemUnselected, XmListAddItemsUnselected – add an item/items to a list.

Synopsis

#include <Xm/List.h>

void XmListAddItemUnselected (Widget *widget*, XmString *item*, int *position*)
void XmListAddItemsUnselected (Widget *widget*, XmString **items*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be added.
items	Specifies a list of items that are to be added.
item_count	Specifies the number of items to be added.
position	Specifies the position at which to add the new item(s).

Availability

`XmListAddItemsUnselected`() is only available in Motif 1.2 and later.

Description

`XmListAddItemUnselected`() inserts the specified *item* into the list, while `XmListAddItemsUnselected`() inserts the specified list of *items*. If *item_count* is smaller than the number of items, only the first *item_count* items of the array are added. The *position* argument specifies the location of the new item(s) in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. An inserted item does not appear selected, even if it matches an item in the XmNselectedItems list.

Usage

`XmListAddItemUnselected`() and `XmListAddItemsUnselected`() are convenience routines that allow you to add items to a list. These routines add items to the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If an item being added to the list duplicates an item that is already selected, the new item does not appear as selected. In order to add items with these routines, you have to create a compound string for each item.

Example

The following callback routine shows how to use of XmListAddItemUnselected() to insert an item into a list in alphabetical order:

```
void add_item (Widget      text_w,
```

```
                XtPointer   client_data,
                XtPointer   call_data)
{
   char       *text, *newtext = XmTextFieldGetString
   (text_w);
   XmString   str, *strlist;
   int        u_bound, l_bound = 0;
   Widget     list_w = (Widget) client_data;

   /* newtext is the text typed in the TextField
   widget */
   if (!newtext || !*newtext) {
      XtFree (newtext);
      return;
   }

   /* get the current entries (and number of entries)
   from the List */
   XtVaGetValues (list_w, XmNitemCount, &u_bound,
                          XmNitems, &strlist, NULL);

   u_bound--;

   /* perform binary search */
   while (u_bound >= l_bound) {
      int i = l_bound + (u_bound - l_bound)/2;

      text = (char *) XmStringUnparse (strlist[i],
                                       NULL,
                                       XmCHARSET_TEXT,
                                       XmCHARSET_TEXT,
                                       NULL, 0,
                                       XmOUTPUT_ALL);

      if (!text)
         break;

      if (strcmp (text, newtext) > 0)
         u_bound = i-1;
      else
         l_bound = i+1;

      XtFree (text);
   }

   /* insert item at appropriate location */
```

```
        str = XmStringCreateLocalized (newtext);
        XmListAddItemUnselected (list_w, str, l_bound+1);
        XmStringFree (str);
        XtFree (newtext);
    }
```

See Also

XmListAddItem(1), XmListReplaceItems(1), XmListReplaceItemsPos(1), XmListReplaceItemsPosUnselected(1), XmListReplaceItemsUnselected(1), XmListReplacePositions(1), XmList(2).

Name

XmListDeleteAllItems – delete all of the items from a list.

Synopsis

#include <Xm/List.h>

void XmListDeleteAllItems (Widget *widget*)

Inputs

widget Specifies the List widget.

Description

`XmListDeleteAllItems`() removes all of the items from the specified List *widget*.

Usage

`XmListDeleteAllItems`() is a convenience routine that allows you to remove all of the items from a list. The routine removes items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources.

See Also

`XmListDeleteItem`(1), `XmListDeleteItemsPos`(1), `XmListDeletePos`(1), `XmListDeletePositions`(1), `XmList`(2).

Name

XmListDeleteItem, XmListDeleteItems – delete an item/items from a list.

Synopsis

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

void XmListDeleteItem (Widget widget, XmString item)
void XmListDeleteItems (Widget widget, XmString *items, int item_count)
```

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be deleted.
items	Specifies a list of items that are to be deleted.
item_count	Specifies the number of items to be deleted.

Description

`XmListDeleteItem`()[1] removes the first occurrence of the specified *item* from the list, while `XmListDeleteItems`() removes the first occurrence of each of the elements of *items*. If an item does not exist, a warning message is displayed.

Usage

`XmListDeleteItem`() and `XmListDeleteItems`() are convenience routines that allow you to remove items from a list. The routines remove items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If there is more than one occurrence of an item in the list, the routines only remove the first occurrence. In order to remove items with these routines, you have to create a compound string for each item. The routines use a linear search to locate the items to be deleted.

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItemsPos`(1), `XmListDeletePos`(1), `XmListDeletePositions`(1), `XmList`(2).

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

Name

XmListDeleteItemsPos – delete items starting at a specified position from a list.

Synopsis

#include <Xm/List.h>

void XmListDeleteItemsPos (Widget *widget*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
item_count	Specifies the number of items to be deleted.
position	Specifies the position from which to delete items.

Description

`XmListDeleteItemsPos`() removes *item_count* items from the list, starting at the specified *position*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. If the number of items between *position* and the end of the list is less than *item_count*, the routine deletes all of the items up through the last item in the list.

Usage

`XmListDeleteItemsPos`() is a convenience routine that allows you to remove items from a list. The routine removes items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. Since you are specifying the position of the items to be removed, you do not have to create compound strings for the items. The routine does not have to search for the items, so it avoids the linear search that is used by XmListDeleteItems().

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItem`(1), `XmListDeletePos`(1), `XmListDeletePositions`(1), `XmList`(2).

Name

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

Synopsis

#include <Xm/List.h>

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

Inputs

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

Description

`XmListDeletePos`() removes the item at the specified *position* from the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. If the list does not have the specified *position*, a warning message is displayed.

Usage

`XmListDeletePos`() is a convenience routine that allows you to remove an item from a list. The routine removes items from the list by internally manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. Since you are specifying the position of the item to be removed, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListDeleteItem`().

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItem`(1), `XmListDeleteItemsPos`(1), `XmListDeletePositions`(1), `XmList`(2).

Name

XmListDeletePositions – delete items at the specified positions from a list.

Synopsis

#include <Xm/List.h>

void XmListDeletePositions (Widget *widget*, int **position_list*, int *position_count*)

Inputs

widget	Specifies the List widget.
position_list	Specifies a list of positions from which to delete items.
position_count	Specifies the number of positions to be deleted.

Availability

Motif 1.2 and later.

Description

`XmListDeletePositions`() removes the items that appear at the positions specified in *position_list* from the list. A position value of 1 indicates the first item, a value of 2 indicates the second item, and so on. If the list does not have the specified position, a warning message is displayed. If *position_count* is smaller than the number of positions in *position_list*, only the first *position_count* items of the array are deleted.

Usage

`XmListDeletePositions`() is a convenience routine that allows you to remove items from a list. The routine remove the items by modifying the XmNitems and XmNitemCount resources. Since you are specifying the positions of the items to be removed, you do not have to create compound strings for the items. The routine does not have to search for the items, so it avoids the linear search that is used by `XmListDeleteItems`().

See Also

`XmListDeleteAllItems`(1), `XmListDeleteItem`(1), `XmListDeleteItemsPos`(1), `XmListDeletePos`(1), `XmList`(2).

Name

XmListDeselectAllItems – deselect all items in a list.

Synopsis

#include <Xm/List.h>

void XmListDeselectAllItems (Widget *widget*)

Inputs

widget Specifies the List widget.

Description

`XmListDeselectAllItems`() unhighlights all of the selected items in the specified *widget* and removes these items from the XmNselectedItems list. If the list is in normal mode, the item with the keyboard focus remains selected; if the list is in add mode, all of the items are deselected.

Usage

`XmListDeselectAllItems`() is a convenience routine that allows you to deselect all of the items in a list. The routine deselects items in the list by internally manipulating the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. This routine does not invoke any selection callbacks for the list when the items are deselected.

See Also

`XmListDeselectItem`(1), `XmListDeselectPos`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListDeselectItem – deselect an item from a list.

Synopsis

#include <Xm/List.h>

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

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be deselected.

Description

`XmListDeselectItem`() unhighlights and removes from the XmNselectedItems list the first occurrence of the specified *item*.

Usage

`XmListDeselectItem`() is a convenience routine that allows you to deselect an item in a list. The routine deselects items in the list by internally manipulating the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. This routine does not invoke any selection callbacks for the list when the item is deselected. If there is more than one occurrence of an item in the list, the routine only deselects the first occurrence. In order to deselect an item with this routine, you have to create a compound string for the item. The routine uses a linear search to locate the item to be deselected.

See Also

`XmListDeselectAllItems`(1), `XmListDeselectPos`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListDeselectPos – deselect an item at the specified position from a list.

Synopsis

#include <Xm/List.h>

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

Inputs

widget	Specifies the List widget.
position	Specifies the position at which to deselect an item.

Description

`XmListDeselectPos`() unhighlights the item at the specified *position* in the list and removes the item from the XmNselectedItems list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. If the list does not have the specified *position*, the routine does nothing.

Usage

`XmListDeselectPos`() is a convenience routine that allows you to deselect an item in a list. The routine deselects items in the list by internally manipulating the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. This routine does not invoke any selection callbacks for the list when the item is deselected. Since you are specifying the position of the item to be deselected, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListDeselectItem`().

See Also

`XmListDeselectAllItems`(1), `XmListDeselectPos`(1), `XmListGetSelectedPos`(1), `XmListPosSelected`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListGetKbdItemPos – get the position of the item in a list that has the location cursor.

Synopsis

#include <Xm/List.h>

int XmListGetKbdItemPos (Widget *widget*)

Inputs

widget Specifies the List widget.

Returns

The position of the item that has the location cursor.

Availability

Motif 1.2 and later.

Description

`XmListGetKbdItemPos`() retrieves the position of the item in the specified List *widget* that has the location cursor. A returned value of 1 indicates the first item, a value of 2 indicates the second item, and so on. The value 0 (zero) specifies that the list is empty.

Usage

`XmListGetKbdItemPos`() provides a way to determine which item in a list has the keyboard focus. This information is useful if you need to perform actions based on the position of the location cursor in the list.

See Also

`XmListSetAddMode`(1), `XmListSetKbdItemPos`(1), `XmList`(2).

Name

XmListGetMatchPos – get all occurrences of an item in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListGetMatchPos (Widget *widget*, XmString *item*, int **position_list*, int **position_count*)

Inputs

widget	Specifies the List widget.
item	Specifies the item whose positions are to be retrieved.

Outputs

position_list	Returns a list of the positions of the item.
position_count	Returns the number of items in position_list.

Returns

True if the item is in the list or False otherwise.

Description

`XmListGetMatchPos`() determines whether the specified *item* exists in the list. If the list contains *item*, the routine returns True and *position_list* returns a list of positions that specify the location(s) of the *item*. A position value of 1 indicates the first item, a position value of 2 indicates the second item, and so on. `XmListGetMatchPos`() allocates storage for the *position_list* array when the item is found; the application is responsible for freeing this storage using `XtFree`(). If the list does not contain *item*, the routine returns False, and *position_count* is set to zero. In Motif 1.2.3 and earlier, the value of *position_list* is undefined if *item* is not within the list. From Motif 1.2.4 and later, *position_list* is set to NULL.

Usage

`XmListGetMatchPos`() is a convenience routine that provides a way to locate all of the occurrences of an item in a list. Alternatively, you could obtain this information yourself using the XmNitems resource and `XmListItemPos`().

Example

The following code fragments show the use of `XmListGetMatchPos`():

```
Widget    list_w;
int      *pos_list;
int       pos_cnt, i;
char     *choice = "A Sample Text String";
XmString  str    = XmStringCreateLocalized
(choice);
```

```
if (!XmListGetMatchPos (list_w, str, &pos_list,
&pos_cnt))
   XtWarning ("Can't get items in list");
else {
   printf ("%s exists at %d positions:", choice,
   pos_cnt);
   for (i = 0; i < pos_cnt; i++)
      printf (" %d", pos_list[i]);
   puts ("");
   XtFree (pos_list);
}
XmStringFree (str);
```

See Also

XmListGetSelectedPos(1), XmList(2).

Name

XmListGetSelectedPos – get the positions of the selected items in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListGetSelectedPos (Widget *widget*, int **position_list*, int *position_count*)

Inputs

widget	Specifies the List widget.

Outputs

position_list	Returns a list of the positions of the selected items.
position_count	Returns the number of items in position_list.

Returns

True if there are selected items in the list or False otherwise.

Description

`XmListGetSelectedPos`() determines whether there are any selected items in the list. If the list has selected items, the routine returns True and *position_list* returns a list of positions that specify the location(s) of the items. A position value of 1 indicates the first item, a position value of 2 indicates the second item, and so on. `XmListGetSelectedPos`() allocates storage for the *position_list* array when there are selected items; the application is responsible for freeing this storage using *XtFree*(). If the list does not contain any selected items, the routine returns False and *position_count* is set to zero. In Motif 1.2.3 and earlier, the value of *position_list* is undefined if there are no selected items within the list. From Motif 1.2.4 and later, *position_list* is set to NULL.

Usage

`XmListGetSelectedPos`() is a convenience routine that provides a way to determine the positions of all of the selected items in a list. Alternatively, you could obtain this information yourself using the XmNselectedItems resource and `XmListItemPos`().

See Also

`XmListGetMatchPos`(1), `XmList`(2).

Name

XmListItemExists – determine if a specified item is in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListItemExists (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the List widget.
item	Specifies the item whose presence in the list is checked.

Returns

True if the item is in the list or False otherwise.

Description

`XmListItemExists`() determines whether the list contains the specified *item*. The routine returns True if the *item* is present and False if it is not.

Usage

`XmListItemExists`() is a convenience routine that determines whether or not an item is in a list. In order to use the routine, you have to create a compound string for the item. The routine uses a linear search to locate the item. You may be able to obtain this information more effectively by searching the XmNitems list using your own search procedure.

See Also

`XmListGetMatchPos`(1), `XmListItemPos`(1), `XmList`(2).

Name

XmListItemPos – return the position of an item in a list.

Synopsis

#include <Xm/List.h>

int XmListItemPos (Widget *widget*, XmString *item*)

Inputs

widget	Specifies the List widget.
item	Specifies the item whose position is returned.

Returns

The position of the item in the list or 0 (zero) if the *item* is not in the list.

Description

`XmListItemPos`() returns the position of the first occurrence of the specified *item* in the list. A position value of 1 indicates the first item, a position value of 2 indicates the second item, and so on. If item is not in the list, `XmListItem-Pos`() returns 0 (zero).

Usage

`XmListItemPos`() is a convenience routine that finds the position of an item in a list. If there is more than one occurrence of the item in the list, the routine only returns the position of the first occurrence. In order to use the routine, you have to create a compound string for the item. The routine uses a linear search to locate the item.

Example

The following routines show how to make sure that a given item in a list is visible:

```
void MakePosVisible (Widget list_w, int item_no)
{
   int top, visible;

   XtVaGetValues (list_w,  XmNtopItemPosition,
   &top,
                           XmNvisibleItemCount,
   &visible,
                           NULL);

   if (item_no < top)
      XmListSetPos (list_w, item_no);
   else if (item_no >= top+visible)
      XmListSetBottomPos (list_w, item_no);
```

```
}

void MakeItemVisible (Widget list_w, XmString item)
{
   int item_no = XmListItemPos (list_w, item);

   if (item_no > 0)
      MakePosVisible (list_w, item_no);
}
```

See Also

`XmListItemExists`(1), `XmListPosSelected`(1), `XmList`(2).

Name

XmListPosSelected – check if the item at a specified position is selected in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListPosSelected (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position that is checked.

Returns

True if the item is selected or False if the item is not selected or the *position* is invalid.

Availability

Motif 1.2 and later.

Description

`XmListPosSelected`() determines whether or not the list item at the specified *position* is selected. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. The value 0 (zero) specifies the last item in the list. The routine returns True if the list item is selected. It returns False if the item is not selected or the list does not have the specified *position*.

Usage

XmListPosSelected() is a convenience routine that lets you check if an item at a particular position is selected. Alternatively, you could check the list of positions returned by `XmListGetSelectedPos`() to see if the item at a position is selected.

See Also

`XmListDeselectPos`(1), `XmListGetSelectedPos`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListPosToBounds – return the bounding box of an item at the specified position in a list.

Synopsis

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

Boolean XmListPosToBounds (  Widget     widget,
                             int        position,
                             Position   *x,
                             Position   *y,
                             Dimension  *width,
                             Dimension  *height)
```

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item for which to return the bounding box.

Outputs

x	Returns the x-coordinate of the bounding box for the item.
y	Returns the y-coordinate of the bounding box for the item.
width	Returns the width of the bounding box for the item.
height	Returns the height of the bounding box for the item.

Returns

True if item at the specified position is visible or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmListPosToBounds`() returns the bounding box of the item at the specified *position* in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. The routine returns the *x* and *y* coordinates of the upper left corner of the bounding box in relation to the upper left corner of the List widget. `XmListPosToBounds`() also returns the *width* and *height* of the bounding box. Passing a NULL value for any of the *x*, *y*, *width*, or *height* parameters indicates that the value for the parameter should not be returned. If the item at the specified *position* is not visible, `XmListPosToBounds`() returns False and the return values are undefined.

Usage

XmListPosToBounds() provides a way to determine the bounding box of an item in a list. This information is useful if you want to perform additional event processing or draw special graphics for the list item.

See Also

XmListYToPos(1), XmList(2).

Name

XmListReplaceItems – replace specified items in a list.

Synopsis

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

void XmListReplaceItems (  Widget     widget,
                           XmString   *old_items,
                           int        item_count,
                           XmString   *new_items)
```

Inputs

widget	Specifies the List widget.
old_items	Specifies a list of the items that are to be replaced.
item_count	Specifies the number of items that are to be replaced.
new_items	Specifies a list of the new items.

Description

`XmListReplaceItems`() replaces the first occurrence of each item in the *old_items* list with the corresponding item from the *new_items* list. If an item in the *old_items* list does not exist in the specified List *widget*, the corresponding item in *new_items*[1] is skipped. If *item_count* is smaller than the number of *old_items* or *new_items*, only the first *item_count* items are replaced. A new item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItems`() is a convenience routine that allows you to replace particular items in a list. The routine replaces items by manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item appears as selected. You should only use this routine if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to replace items with this routine, you have to create compound strings for all of the old and new items. The routine uses a linear search to locate the items to be replaced.

See Also

`XmListAddItem`(1), `XmListAddItemUnselected`(1), `XmListReplaceItemsPos`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplaceItemsUnselected`(1), `XmListReplacePositions`(1), `XmList`(2).

1.Erroneously given as *new_list* in 1st and 2nd edition.

Name

XmListReplaceItemsPos – replace specified items in a list.

Synopsis

#include <Xm/List.h>

void XmListReplaceItemsPos (Widget *widget*, XmString **new_items*, int *item_count*, int *position*)

Inputs

widget	Specifies the List widget.
new_items	Specifies a list of the new items.
item_count	Specifies the number of items that are to be replaced.
position	Specifies the position at which to replace items.

Description

`XmListReplaceItemsPos`() replaces a consecutive number of items in the list with items from the *new_items* list. The first item that is replaced is located at the specified *position* and each subsequent item is replaced by the corresponding item from *new_items*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. If *item_count* is smaller than the number of *new_items*, only the first *item_count* items are replaced. If the number of items between *position* and the end of the list is less than *item_count*, the routine replaces all of the items up through the last item in the list. A new item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItemsPos`() is a convenience routine that allows you to replace a contiguous sequence of items in a list. The routine replaces items by manipulating the array of compound strings specified by the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item appears as selected. You should only use this routine if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to replace items with this routine, you have to create compound strings for all of the new items. The routine does not have to search for the items, so it avoids the linear searches that are used by `XmListReplaceItems`().

See Also

`XmListAddItem`(1), `XmListAddItemUnselected`(1), `XmListReplaceItems`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplaceItemsUnselected`(1), `XmListReplacePositions`(1), `XmList`(2).

Name

XmListReplaceItemsPosUnselected – replace specified items in a list.

Synopsis

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

void XmListReplaceItemsPosUnselected (  Widget      widget,
                                        XmString    *new_items,
                                        int         item_count,
                                        int         position)
```

Inputs

widget	Specifies the List widget.
new_items	Specifies a list of the new items.
item_count	Specifies the number of items that are to be replaced.
position	Specifies the position at which to replace items.

Availability

Motif 1.2 and later.

Description

`XmListReplaceItemsPosUnselected`() replaces a consecutive number of items in the list with items from the *new_items* list. The first item that is replaced is located at the specified *position* and each subsequent item is replaced by the corresponding item from *new_items*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. If *item_count* is smaller than the number of *new_items*, only the first *item_count* items are replaced. If the number of items between *position* and the end of the list is less than *item_count*, the routine replaces all of the items up through the last item in the list. A new item does not appear selected, even if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItemsPosUnselected`() is a convenience routine that allows you to replace a contiguous sequence of items in a list. The routine replaces items by modifying the array of compound strings specified through the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item does not appear as selected. In order to replace items with this routine, you have to create compound strings for all of the new items. The routine does not have to search for the items, so it avoids the linear searches that are used by `XmListReplaceItemsUnselected`().

See Also

XmListAddItem(1), XmListAddItemUnselected(1), XmListReplaceItems(1), XmListReplaceItemsPos(1), XmListReplaceItemsUnselected(1), XmListReplacePositions(1), XmList(2).

Name

XmListReplaceItemsUnselected – replace specified items in a list.

Synopsis

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

void XmListReplaceItemsUnselected (  Widget     widget,
                                     XmString   *old_items,
                                     int        item_count,
                                     XmString   *new_items)
```

Inputs

widget	Specifies the List widget.
old_items	Specifies a list of the items that are to be replaced.
item_count	Specifies the number of items that are to be replaced.
new_items	Specifies a list of the new items.

Availability

Motif 1.2 and later.

Description

`XmListReplaceItemsUnselected`() replaces the first occurrence of each item in the *old_items* list with the corresponding item from the *new_items* list. If an item in the *old_items* list does not exist in the specified List *widget*, the corresponding item in *new_items*[1] is skipped. If *item_count* is smaller than the number of *old_items* or *new_items*, only the first *item_count* items are replaced. A new item does not appear selected, even if it matches an item in the XmNselectedItems list.

Usage

`XmListReplaceItemsUnselected`() is a convenience routine that allows you to replace particular items in a list. The routine replaces items by modifying the array of compound strings specified through the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item does not appear as selected. In order to replace items with this routine, you have to create compound strings for all of the old and new items. The routine uses a linear search to locate the items to be replaced.

See Also

`XmListAddItem(1), XmListAddItemUnselected(1), XmListReplaceItems(1), XmListReplaceItemsPos(1), XmListReplaceItemsPosUnselected(1), XmListReplacePositions(1), XmList(2).`

1.Erroneously given as *new_list* in 1st and 2nd editions.

Name

XmListReplacePositions – replace items at the specified positions in a list.

Synopsis

#include <Xm/List.h>

void XmListReplacePositions (Widget *widget*, int **position_list*, XmString **item_list*, int *item_count*)

Inputs

widget	Specifies the List widget.
position_list	Specifies a list of positions at which to replace items.
item_list	Specifies a list of the new items.
item_count	Specifies the number of items that are to be replaced.

Availability

Motif 1.2 and later.

Description

`XmListReplacePositions`() replaces the items that appear at the positions specified in *position_list* with the corresponding items from *item_list*. A position value of 1 indicates the first item, a value of 2 indicates the second item, and so on. If the list does not have the specified position, a warning message is displayed. If *item_count* is smaller than the number of positions in *position_list*, only the first *item_count* items are replaced. A new item appears selected if it matches an item in the XmNselectedItems list.

Usage

`XmListReplacePositions`() is a convenience routine that allows you to replace items at particular positions in a list. The routine replaces items by modifying the array of compound strings specified through the XmNitems and XmNitemCount resources. If a new item duplicates an item that is already selected, the new item appears as selected. You should only use this routine if the list supports multiple selections and you want to select the new items whose duplicates are already selected. In order to replace items with this routine, you have to create compound strings for all of the new items. The routine does not have to search for the items, so it avoids the linear searches that are used by `XmListReplaceItems`().

See Also

`XmListAddItem`(1), `XmListAddItemUnselected`(1), `XmListReplaceItems`(1), `XmListReplaceItemsPos`(1), `XmListReplaceItemsPosUnselected`(1), `XmListReplaceItemsUnselected`(1), `XmList`(2).

Name

XmListSelectItem – select an item from a list.

Synopsis

#include <Xm/List.h>

void XmListSelectItem (Widget *widget*, XmString *item*, Boolean *notify*)

Inputs

widget	Specifies the List widget.
item	Specifies the item that is to be selected.
notify	Specifies whether or not the selection callback is invoked.

Description

`XmListSelectItem`() highlights and selects the first occurrence of the specified *item* in the list. If the XmNselectionPolicy resource of the list is XmMULTIPLE_SELECT, the routine toggles the selection state of *item*. For any other selection policy, `XmListSelectItem`() replaces the currently selected item(s) with *item*. The XmNselectedItems resource specifies the current selection of the list. If notify is True, `XmListSelectItem`() invokes the selection callback for the current selection policy.

Usage

`XmListSelectItem`() is a convenience routine that allows you to select an item in a list. The routine selects the item by modifying the array of compound strings specified by the XmNselectedItems and XmNselectedItemCount resources. In order to select an item with this routine, you have to create a compound string for the item. The routine uses a linear search to locate the item to be selected. XmListSelectItem() only allows you to select a single item; there are no routines for selecting multiple items. If you need to select more than one item, use `XtSetValues`() to set XmNselectedItems and XmNselectedItemCount.

The *notify* parameter indicates whether or not the selection callbacks for the current selection policy are invoked. You can avoid redundant code by setting this parameter to True. If you are calling `XmListSelectItem`() from a selection callback routine, you probably want to set the parameter to False to avoid the possibility of an infinite loop. Calling `XmListSelectItem`() with *notify* set to True causes the callback routines to be invoked in a way that is indistinguishable from a user-initiated selection action.

See Also

`XmListDeselectAllItems`(1), `XmListDeselectItem`(1), `XmListDeselectPos`(1), `XmListSelectPos`(1), `XmListUpdateSelectedList`(1), `XmList`(2).

Name

XmListSelectPos – select an item at the specified position from a list.

Synopsis

#include <Xm/List.h>

void XmListSelectPos (Widget *widget*, int *position*, Boolean *notify*)

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item that is to be selected.
notify	Specifies whether or not the selection callback is invoked.

Description

`XmListSelectPos`() highlights and selects the item at the specified *position* in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. If the XmNselectionPolicy resource of the list is XmMULTIPLE_SELECT, the routine toggles the selection state of the item. For any other selection policy, `XmListSelectPos`() replaces the currently selected item with the specified item. The XmNselectedItems resource lists the current selection of the list. If notify is True, `XmListSelectPos`() invokes the selection callback for the current selection policy.

Usage

`XmListSelectPos`() is a convenience routine that allows you to select an item at a particular position in a list. The routine selects the item by modifying the array of compound strings specified through the XmNselectedItems and XmNselectedItemCount resources. Since you are specifying the position of the item to be selected, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListSelectItem`(). `XmListSelectPos`() only allows you to select a single item; there are no routines for selecting multiple items. If you need to select more than one item, use XtSetValues() to set XmNselectedItems and XmNselectedItemCount.

The *notify* parameter indicates whether or not the selection callbacks for the current selection policy are invoked. You can avoid redundant code by setting this parameter to True. If you are calling XmListSelectPos() from a selection callback routine, you probably want to set the parameter to False to avoid the possibility of an infinite loop. Calling `XmListSelectPos`() with *notify* set to True causes the callback routines to be invoked in a way that is indistinguishable from a user-initiated selection action.

See Also

XmListDeselectAllItems(1), XmListDeselectItem(1), XmListDeselectPos(1), XmListGetSelectedPos(1), XmListPosSelected(1), XmListSelectItem(1), XmList(2).

Name

XmListSetAddMode – set add mode in a list.

Synopsis

#include <Xm/List.h>

void XmListSetAddMode (Widget *widget*, Boolean *mode*)

Inputs

widget	Specifies the List widget.
mode	Specifies whether to set add mode on or off.

Description

`XmListSetAddMode`() sets the state of add mode when the XmNselectionPolicy is XmEXTENDED_SELECT. If *mode* is True, add mode is turned on; if *mode* is False, add mode is turned off. When a List widget is in add mode, the user can move the location cursor without disturbing the current selection.

Usage

XmListSetAddMode() provides a way to change the state of add mode in a list. The distinction between normal mode and add mode is only important for making keyboard-based selections. In normal mode, the location cursor and the selection move together, while in add mode, the location cursor and the selection can be separate.

See Also

`XmListGetKbdItemPos`(1), `XmListSetKbdItemPos`(1), `XmList`(2).

Name

XmListSetBottomItem – set the last visible item in a list.

Synopsis

#include <Xm/List.h>

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

Inputs

widget	Specifies the List widget.
item	Specifies the item that is made the last visible item.

Description

`XmListSetBottomItem`() scrolls the List *widget* so that the first occurrence of the specified *item* appears as the last visible item in the list.

Usage

`XmListSetBottomItem`() provides a way to make sure that a particular *item* is visible in a list. The routine changes the viewable portion of the list so that the specified *item* is displayed at the bottom of the viewport. If there is more than one occurrence of the *item* in the list, the routine uses the first occurrence. In order to use this routine, you have to create a compound string for the *item*. The routine uses a linear search to locate the *item*.

See Also

`XmListSetBottomPos`(1), `XmListSetHorizPos`(1), `XmListSetItem`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetBottomPos – set the last visible item in a list.

Synopsis

#include <Xm/List.h>

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

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item that is made the last visible item.

Description

`XmListSetBottomPos`() scrolls the List *widget* so that the item at the specified *position* appears as the last visible item in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list.

Usage

`XmListSetBottomPos`() provides a way to make sure that an item at a particular position is visible in a list. The routine changes the viewable portion of the list so that the item at the specified *position* is displayed at the bottom of the viewport. Since you are specifying the position of the item, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListSetBottomItem`().

Example

The following routine shows how to make sure that an item at a given position in a list is visible:

```
void MakePosVisible (Widget list_w, int item_no)
{
    int top, visible;

    XtVaGetValues (list_w, XmNtopItemPosition, &top, XmNvisibleItem-
    Count, &visible, NULL);

    if (item_no < top)
        XmListSetPos (list_w, item_no);
    else if (item_no >= top+visible)
        XmListSetBottomPos (list_w, item_no);
}
```

See Also

`XmListSetBottomItem`(1), `XmListSetHorizPos`(1), `XmListSetItem`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetHorizPos – set the horizontal position of a list.

Synopsis

#include <Xm/List.h>

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

Inputs

widget	Specifies the List widget.
position	Specifies the horizontal position.

Description

`XmListSetHorizPos`() scrolls the list to the specified horizontal *position*. If XmNlistSizePolicy is set to XmCONSTANT or XmRESIZE_IF_POSSIBLE and the horizontal scroll bar is visible, `XmListSetHorizPos`() sets the XmNvalue resource of the horizontal scroll bar to the specified *position* and updates the visible area of the list.

Usage

When a list item is too long to fit horizontally inside the viewing area of a List widget, the widget either expands horizontally or adds a horizontal scroll bar, depending on the value of the XmNlistSizePolicy resource. Calling `XmListSetHorizPos`() is equivalent to the user moving the horizontal scroll bar to the specified location.

See Also

`XmListSetBottomItem`(1), `XmListSetBottomPos`(1), `XmListSetItem`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetItem – set the first visible item in a list.

Synopsis

#include <Xm/List.h>

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

Inputs

widget	Specifies the List widget.
item	Specifies the item that is made the first visible item.

Description

`XmListSetItem`() scrolls the List *widget* so that the first occurrence of the specified *item* appears as the first visible item in the list.

Usage

`XmListSetItem`() provides a way to make sure that a particular *item* is visible in a list. The routine changes the viewable portion of the list so that the specified *item* is displayed at the top of the viewport. Using this routine is equivalent to setting the XmNtopItemPosition resource. If there is more than one occurrence of the *item* in the list, the routine uses the first occurrence. In order to use this routine, you have to create a compound string for the *item*. The routine uses a linear search to locate the *item*.

See Also

`XmListSetBottomItem`(1), `XmListSetBottomPos`(1), `XmListSetHorizPos`(1), `XmListSetPos`(1), `XmList`(2).

Name

XmListSetKbdItemPos – set the position of the location cursor in a list.

Synopsis

#include <Xm/List.h>

Boolean XmListSetKbdItemPos (Widget *widget*, int *position*)

Inputs

widget	Specifies the List widget.
position	Specifies the position where the location cursor is set.

Returns

True on success or False if there is not item at position or the list is empty.

Availability

Motif 1.2 and later.

Description

`XmListSetKbdItemPos`() sets the location cursor at the specified *position*. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list. The routine does not check the selection state of the item at the specified location.

Usage

`XmListSetKbdItemPos`() provides a way to change which item in a list has the keyboard focus. The routine is useful if you need to make sure that particular item has the keyboard focus at a given time, such as when the list first receives the keyboard focus.

See Also

`XmListGetKbdItemPos`(1), `XmListSetAddMode`(1), `XmList`(2).

Name

XmListSetPos – sets the first visible item in a list.

Synopsis

#include <Xm/List.h>

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

Inputs

widget	Specifies the List widget.
position	Specifies the position of the item that is made the first visible item.

Description

`XmListSetPos`() scrolls the List widget so that the item at the specified *position* appears as the first visible item in the list. A *position* value of 1 indicates the first item, a *position* value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list.

Usage

`XmListSetPos`() provides a way to make sure that an item at a particular location is visible in a list. The routine changes the viewable portion of the list so that the item at the specified position is displayed at the top of the viewport. Using this routine is equivalent to setting the XmNtopItemPosition resource. Since you are specifying the position of the item, you do not have to create a compound string for the item. The routine does not have to search for the item, so it avoids the linear search that is used by `XmListSetItem`().

Example

The following routine shows how to make sure that an item at a given position in a list is visible:

```
void MakePosVisible (Widget list_w, int item_no)
{
    int top, visible;

    XtVaGetValues (list_w, XmNtopItemPosition, &top, XmNvisibleItem-
    Count, &visible, NULL);

    if (item_no < top)
        XmListSetPos (list_w, item_no);
    else if (item_no >= top+visible)
        XmListSetBottomPos (list_w, item_no);
}
```

See Also

`XmListSetBottomItem(1), XmListSetBottomPos(1), XmListSetHorizPos(1), XmListSetItem(1), XmList(2).`

Name

XmListUpdateSelectedList – update the list of selected items in a list.

Synopsis

#include <Xm/List.h>

void XmListUpdateSelectedList (Widget *widget*)

Inputs

widget Specified the List widget.

Availability

Motif 1.2 and later.

Description

`XmListUpdateSelectedList`() updates the array of compound strings specified through the XmNselectedItems resource. The routine frees the current selected array, and then traverses the array of compound strings specified by the XmNitems resource, adding each currently selected item to the XmNselectedItems list.

Usage

`XmListUpdateSelectedList`() provides a way to update the list of selected items in a list. This routine is useful if the actual items that are selected are not synchronized with the value of the XmNselectedItems resource. This situation might arise if you are using internal list functions and modifying internal data structures. If you are using the defined list routines, the situation should never occur.

See Also

`XmListDeselectAllItems`(1), `XmListDeselectItem`(1), `XmListDeselectPos`(1), `XmListGetSelectedPos`(1), `XmListPosSelected`(1), `XmListSelectItem`(1), `XmListSelectPos`(1), `XmList`(2).

Name

XmListYToPos – get the position of the item at the specified y-coordinate in a list.

Synopsis

#include <Xm/List.h>

int XmListYToPos (Widget *widget*, Position *y*)

Inputs

widget	Specifies the List widget.
y	Specifies the y-coordinate.

Returns

The position of the item at the specified y-coordinate.

Availability

Motif 1.2 and later.

Description

`XmListYToPos`() retrieves the position of the item at the specified *y*-coordinate in the list. The *y*-coordinate is specified in the coordinate system of the list. A returned value of 1 indicates the first item, a value of 2 indicates the second item, and so on. The value 0 (zero) specifies that there is no item at the specified location.

As of Motif 1.2, a return value of 0 (zero) indicates the first item, a value of 1 indicates the second item, and so on. In Motif 1.2.3 and earlier, the value that is returned may not be a valid position in the list, so an application should check the value with respect to the value of XmNitemCount before using it. In Motif 1.2.4 and later, the returned position may not exceed the value of XmNitemCount.

Usage

`XmListYToPos`() provides a way to translate a y-coordinate into a list position. This routine is useful if you are processing events that report a pointer position and you need to convert the location of the event into an item position.

See Also

`XmListPosToBounds`(1), `XmList`(2).

Name

XmMainWindowSep1, XmMainWindowSep2, XmMainWindowSep3 – get the widget ID of a MainWindow Separator.

Synopsis

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

Widget XmMainWindowSep1 (Widget widget)
Widget XmMainWindowSep2 (Widget widget)
Widget XmMainWindowSep3 (Widget widget)
```

Inputs

widget Specifies the MainWindow widget.

Returns

The widget ID of the particular MainWindow Separator.

Availability

In Motif 2.0 and later, these routines are marked as deprecated.

Description

`XmMainWindowSep1`() returns the widget ID of the MainWindow widget's first Separator, which is located directly below the MenuBar. `XmMainWindowSep2`() returns the widget ID of the second Separator in the Main Window, which is between the Command and ScrolledWindow widgets. `XmMainWindowSep3`() returns the widget ID of the MainWindow's third Separator, which is located just above the message window. The three Separator widgets in a MainWindow are visible only when the XmNshowSeparator resource is set to True.

Usage

`XmMainWindowSep1`(), `XmMainWindowSep2`(), and `XmMainWindowSep3`() provide access to the three Separator widgets that can be displayed by a MainWindow widget. With the widget IDs, you can change the visual attributes of the individual Separators.

In Motif 2.0 and later, the function `XtNameToWidget`() is the preferred method of obtaining the MainWindow components. You should pass *widget* as the first parameter, and "Separator1", "Separator2", or "Separator3" as the second parameter to this procedure.

See Also

`XmMainWindowSetAreas`(1), `XmMainWindow`(2), `XmScrolledWindow`(2).

Name

XmMainWindowSetAreas – specify the children for a MainWindow.

Synopsis

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

void XmMainWindowSetAreas ( Widget  widget,
                            Widget  menu_bar,
                            Widget  command_window,
                            Widget  horizontal_scrollbar,
                            Widget  vertical_scrollbar,
                            Widget  work_region)
```

Inputs

widget	Specifies the MainWindow widget.
menu_bar	Specifies the widget ID of the MenuBar.
command_window	Specifies the widget ID of the command window.
horizontal_scrollbar	Specifies the widget ID of the horizontal ScrollBar.
vertical_scrollbar	Specifies the widget ID of the vertical ScrollBar.
work_region	Specifies the widget ID of the work window.

Availability

In Motif 2.0 and later, the procedure is marked as deprecated.

Description

`XmMainWindowSetAreas`() sets up the standard regions of the MainWindow *widget* for an application. The MainWindow must be created before the routine is called. `XmMainWindowSetAreas`() specifies the MenuBar, the work window, the command window, and the horizontal and vertical ScrollBars for the MainWindow. If an application does not have one of these regions, the corresponding argument can be specified as NULL. Each region may have child widgets, and this routine determines which of those children will be actively managed by the MainWindow.

Usage

Each of the MainWindow regions is associated with a MainWindow resource; `XmMainWindowSetAreas`() sets the associated resources. The associated resources that correspond to the last five arguments to the routine are XmNmenuBar, XmNcommand, XmNhorizontalScrollBar, XmNverticalScrollBar, and XmNworkWindow. `XmMainWindowSetAreas`() does not provide a way to set up the message area; this region must be set up by specifying the XmNmessageWindow resource.

If an application does not call `XmMainWindowSetAreas`(), the widget may still set some of the standard regions. When a MenuBar child is added to a MainWindow, if XmNmenuBar has not been set, it is set to the MenuBar child. When a Command child is added to a MainWindow, if XmNcommand has not been set, it is set to the Command child. If ScrollBars are added as children, the XmNhorizontalScrollBar and XmNverticalScrollBar resources may be set if they have not already been specified. Any child that is not one of these types is used for the XmNworkWindow. If you want to be certain about which widgets are used for the different regions, it is wise to call `XmMainWindowSetAreas`() explicitly.

In Motif 2.0 and later, `XmMainWindowSetAreas`(), is deprecated. The programmer should use `XtSetValues`() in order to specify the XmNcommandWindow, XmNmenuBar, XmNworkWindow, XmNhorizontalScrollBar, and XmNverticalScrollBar resources of the MainWindow. `XmMainWindowSetAreas`() does not handle the XmNmessageWindow resource in any case.

Example

The following code fragment shows how to set some of the regions of a MainWindow:

```
Widget top, main_w, menubar, command_w, text_w, scrolled_text_w;
Arg    args[4];

main_w = XtVaCreateManagedWidget("main_w", xmMainWindowWidgetClass, top, NULL);
menubar = XmCreateMenuBar (main_w, "menubar", NULL, 0);
XtManageChild (menubar);

XtSetArg (args[0], XmNrows, 24);
XtSetArg (args[1], XmNcolumns, 80);
XtSetArg (args[2], XmNeditable, False);
XtSetArg (args[3], XmNeditMode, XmMULTI_LINE_EDIT);
text_w = XmCreateScrolledText (main_w, "text_w", args, 4);
XtManageChild (text_w);

scrolled_text_w = XtParent (text_w);
command_w = XmCreateText (main_w, "command_w", (Arg *) 0, 0);
XtManageChild (command_w);

#if    (XmVERSION > 1)
XtVaSetValues (main_w,
                    XmNmenuBar,              menubar,
                    XmNcommandWindow,        command_w,
                    XmNhorizontalScrollBar,  NULL,
                    XmNverticalScrollBar,    NULL,
```

```
                    XmNworkWindow,          scrolled_text_w,
                    0);
#else  /* XmVERSION > 1 */
XmMainWindowSetAreas (main_w, menubar, command_w, NULL, NULL,
scrolled_text_w);
#endif /* XmVERSION > 1 */
```

See Also

`XmMainWindowSep`(1), `XmMainWindow`(2), `XmScrolledWindow`(2).

Name

XmMapSegmentEncoding – get the compound text encoding format for a font list element tag.

Synopsis

char * XmMapSegmentEncoding (char *_fontlist_tag_)

Inputs

fontlist_tag Specifies the compound string font list element tag.

Returns

A character string that contains a copy of the compound text encoding format or NULL if the font list element tag is not found in the registry.

Availability

Motif 1.2 and later.

Description

`XmMapSegmentEncoding`() retrieves the compound text encoding format associated with the specified *fontlist_tag*. The toolkit stores the mappings between compound text encodings and font list elements tags in a registry. `XmMapSegmentEncoding`() searches the registry for a compound text encoding format associated with the specified *fontlist_tag* and returns a copy of the format. If *fontlist_tag* is not in the registry, the routine returns NULL. `XmMapSegmentEncoding`() allocates storage for the returned character string; the application is responsible for freeing the storage using XtFree().

Usage

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

`XmCvtXmStringToCT`() converts a compound string into compound text by using the font list tag of each compound string segment to select a compound text format from the registry for the segment. `XmMapSegmentEncoding`() provides a way for an application to determine the compound text format that would be used for a particular font list element tag.

See Also

`XmCvtXmStringToCT`(1), `XmRegisterSegmentEncoding`(1).

Name

XmMenuPosition – position a popup menu.

Synopsis

#include <Xm/RowColumn.h>

void XmMenuPosition (Widget *menu*, XButtonPressedEvent **event*)

Inputs

menu Specifies the PopupMenu.

event Specifies the event that was passed to the action procedure managing the PopupMenu.

Description

`XmMenuPosition`() positions a popup menu, using the values of the x_root and y_root fields from the specified *event*. An application must call this routine before managing the popup menu, except when the application is positioning the menu itself.

Usage

The *event* parameter for `XmMenuPosition`() is defined to be of type XButtonPressedEvent *; using another type of event might lead to toolkit problems. The x_root and y_root fields in the *event* structure are used to position the menu at the location of the mouse button press. You can modify these fields to position the menu at another location.

In Motif 2.0 and later, a menu whose XmNpopupEnabled resource is XmPOPUP_AUTOMATIC or XmPOPUP_AUTOMATIC_RECURSIVE has an installed event handler which calls `XmMenuPosition`() directly without the need for an application to intervene in posting the menu.

Example

The following routine shows the use of an event handler to post a popup menu.

```
void PostIt (Widget w, XtPointer client_data, XEvent *event, Boolean *dispatch)
{
    Widget                popup = (Widget) client_data;
    XButtonPressedEvent   *bevent = (XButtonPressedEvent *) event;

    if ((bevent->type != ButtonPress) && (bevent->button != 3))
        return;

    XmMenuPosition (popup, bevent);
    XtManageChild (popup);
}
```

```
...
extern Widget some_widget;      /* Where the menu is posted */
extern Widget my_menu;          /* The menu to post */

XtAddEventHandler(some_widget, ButtonPressMask, False, PostIt, (XtPointer)
my_menu) ;
```

See Also

XmRowColumn(2), XmPopupMenu(2).

Name

XmMessageBoxGetChild – get the specified child of a MessageBox widget.

Synopsis

#include <Xm/MessageB.h>

Widget XmMessageBoxGetChild (Widget *widget*, unsigned char *child*)

Inputs

widget Specifies the MessageBox widget.
child Specifies the child of the MessageBox widget. Pass one of the values from the list below.

Returns

The widget ID of the specified child of the MessageBox.

Availability

As of Motif 2.0, the toolkit abstract child fetch routines are marked for deprecation. You should give preference to `XtNameToWidget`(), except when fetching the MessageBox default button.

Description

`XmMessageBoxGetChild`() returns the widget ID of the specified *child* of the MessageBox *widget*.

Usage

The *child* values XmDIALOG_CANCEL_BUTTON, XmDIALOG_HELP_BUTTON, and XmDIALOG_OK_BUTTON specify the action buttons in the *widget*. A *child* value of XmDIALOG_DEFAULT_BUTTON specifies the current default button. The value XmDIALOG_SYMBOL_LABEL specifies the label used to display the message symbol, while XmDIALOG_MESSAGE_LABEL specifies the message label. XmDIALOG_SEPARATOR specifies the separator that is positioned between the message and the action buttons. For more information on the different children of the MessageBox, see the manual page in Section 2, *Motif and Xt Widget Classes*.

Widget Hierarchy

As of Motif 2.0, most Motif composite child fetch routines are marked as deprecated. However, since it is not possible to fetch the XmDIALOG_DEFAULT_BUTTON child using a public interface except through `XmMessageBoxGetChild`(), the routine should not be considered truly deprecated. For consistency with the preferred new style, when fetching all other child values, consider giving preference to the Intrinsics routine `XtNameToWidget`(), passing one of the following names as the second parameter:

“Cancel”	(XmDIALOG_CANCEL_BUTTON)
“OK”	(XmDIALOG_OK_BUTTON)
“Separator”	(XmDIALOG_SEPARATOR)
“Help”	(XmDIALOG_HELP_BUTTON)
“Symbol”	(XmDIALOG_SYMBOL_LABEL)
“Message”	(XmDIALOG_MESSAGE_LABEL)

Structures

The possible values for child are:

XmDIALOG_CANCEL_BUTTON	XmDIALOG_OK_BUTTON
XmDIALOG_DEFAULT_BUTTON	XmDIALOG_SEPARATOR
XmDIALOG_HELP_BUTTON	
XmDIALOG_SYMBOL_LABEL	
XmDIALOG_MESSAGE_LABEL	

See Also

`XmBulletinBoard`(2), XmBulletinBoardDialog(2), `XmErrorDialog`(2), `XmInformationDialog`(2), `XmManager`(2), `XmMessageBox`(2), `XmMessageDialog`(2), `XmQuestionDialog`(2), `XmTemplateDialog`(2), `XmWarningDialog`(2), `XmWorkingDialog`(2).

Name

XmNotebookGetPageInfo – return information about a Notebook page.

Synopsis

#include <Xm/Notebook.h>

XmNotebookPageStatus XmNotebookGetPageInfo (Widget *widget*,
int *page_number*,
XmNotebookPageInfo **page_info*)

Inputs

widget Specifies the Notebook widget.
page_number Specifies a logical page number.

Outputs

page_info Returns a structure into which the requested page information is placed.

Returns

The status of the search for the requested information.

Availability

Motif 2.0 and later.

Description

`XmNotebookGetPageInfo`() returns information associated with a logical page of the Notebook.

The Notebook searches through the list of its children, looking for those which are associated with the logical page number specified by *page_number*. The Notebook principally searches for page children, but collects data in passing on any status area child with a matching logical number, or major and minor tab children whose logical page number does not exceed *page_number*. The function returns within the *page_info* structure the data collected for each of the child widget types.

If the requested *page_number* is greater than the value of the Notebook XmNlast-PageNumber resource, or less than the Notebook XmNfirstPageNumber value, the function returns XmPAGE_INVALID.

Otherwise, if exactly one matching page child is found, the function returns XmPAGE_FOUND. If more than one matching page child is found, the routine returns XmPAGE_DUPLICATED. For no matching page child, the return value is XmPAGE_EMPTY.

Usage

XmNotebookGetPageInfo performs a linear search through the children of the Notebook for widgets whose XmNpageNumber constraint resource matches the requested *page_number*. If a matching child is found with the XmNnotebookChildType resource set to XmPAGE, the widget ID is stored within the *page_widget* element of the *page_info* structure. If a matching child is of type XmSTATUS_AREA, the widget ID is placed in the *status_area_widget* element. If during the search a child widget is found which is of type XmMAJOR_TAB, and the logical page number of the child does not exceed *page_number*, the widget ID is stored within the *major_tab_widget* element. Again, if a child widget is found of type XmMINOR_TAB, and the logical page number of the child does not exceed *page_number*, the widget ID is stored within the *minor_tab_widget* element of page_info.

The *page_widget*, *status_area_widget*, *major_tab_widget*, and *minor_tab_widget* elements of the *page_info* structure are set during the search as each Notebook child is compared, even if no XmPAGE child is found, or if *page_number* exceeds the Notebook first and last page resources. An element of the *page_info* structure can be NULL if no child of the associated type is found with a logical page number which meets the matching criteria.

The Notebook automatically sorts children into ascending logical page order, and the search is terminated as soon as any child has a logical page number which exceeds the requested *page_number*.

Structures

XmNotebookPageInfo is defined as follows:

```
typedef struct {
    int         page_number;          /* the requested page number      */
    Widget      page_widget;          /* any matching page widget       */
    Widget      status_area_widget;   /* any matching status area widget */
    Widget      major_tab_widget;     /* the nearest major tab widget   */
    Widget      minor_tab_widget;     /* the nearest minor tab widget   */
} XmNotebookPageInfo;
```

A XmNotebookPageStatus can have one of the following values:

XmPAGE_FOUND	XmPAGE_INVALID
XmPAGE_EMPTY	XmPAGE_DUPLICATED

See Also

`XmNotebook`(2).

Name

XmObjectAtPoint – determine the child nearest to a point.

Synopsis

#include <Xm/Xm.h>

Widget XmObjectAtPoint (Widget *widget*, Position *x*, Position *y*)

Inputs

widget	Specifies a composite widget.
x	Specifies an X coordinate relative to the widget left side.
y	Specifies an Y coordinate relative to the widget top side.

Returns

The widget most closely associated with the coordinate x, y.

Availability

Motif 2.0 or later.

Description

`XmObjectAtPoint`() searches the list of children of *widget*, and returns the widget ID of the child associated with the *x, y* coordinate. *x* and *y* are interpreted as pixel values, relative to the top left of the Manager widget.

Usage

`XmObjectAtPoint`() calls the object_at_point method associated with a Manager widget, in order to determine the child of the Manager most closely associated with the coordinate specified by *x* and *y*. Each widget class may override the object_at_point method inherited from Manager, to redefine what is meant by "associated".

The default Manager class method returns the last managed gadget which contains the coordinate.

The DrawingArea overrides the default method, and performs a simple linear search for the first managed child, widget or gadget, which contains the coordinate.

The Container overrides the object_at_point method, by searching through the list of logical child nodes, using any XmQTpointIn trait held by each child to determine a logical match with the coordinate. If no XmQTpointIn is held by the child, the Container simply checks whether the coordinate is within the child dimensions. The IconGadget holds the XmQTpointIn trait, although neither this fact nor the trait itself is otherwise documented.

See Also

`XmContainer`(2), `XmDrawingArea`(2), `XmGadget`(2), `XmIconGadget`(2), `XmManager`(2).

Name

XmOptionButtonGadget – get the CascadeButtonGadget in an option menu

Synopsis

#include <Xm/RowColumn.h>

Widget XmOptionButtonGadget (Widget *option_menu*)

Inputs

option_menu Specifies the option menu.

Returns

The widget ID of the internal CascadeButtonGadget.

Description

`XmOptionButtonGadget`() returns the widget ID for the internal CascadeButtonGadget that is created when the specified *option_menu* widget is created. An option menu is a RowColumn widget containing two gadgets: a CascadeButtonGadget that displays the current selection and posts the submenu and a LabelGadget that displays the XmNlabelString resource.

Usage

`XmOptionButtonGadget`() provides a way for an application to access the internal CascadeButtonGadget that is part of an option menu. Once you have retrieved the gadget, you can alter its appearance. In Motif 1.2, you can also specify resources for the gadget using the widget name OptionButton.

See Also

`XmOptionLabelGadget`(1), `XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmOptionMenu`(2), `XmRowColumn`(2).

Name

XmOptionLabelGadget – get the LabelGadget in an option menu.

Synopsis

#include <Xm/RowColumn.h>

Widget XmOptionLabelGadget (Widget *option_menu*)

Inputs

option_menu Specifies the option menu.

Description

`XmOptionLabelGadget`() returns the widget ID for the internal LabelGadget that is created when the specified *option_menu* widget is created. An option menu is a RowColumn widget containing two gadgets: a LabelGadget that displays the XmNlabelString resource, and a CascadeButtonGadget that displays the current selection and posts the submenu.

Usage

`XmOptionLabelGadget`() provides a way for an application to access the internal LabelGadget that is part of an option menu. Once you have retrieved the gadget, you can alter its appearance. In Motif 1.2, you can also specify resources for the gadget using the widget name OptionLabel.

See Also

`XmOptionButtonGadget`(1), `XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmOptionMenu`(2), `XmRowColumn`(2).

Name

XmParseMappingCreate – create a parse mapping.

Synopsis

XmParseMapping XmParseMappingCreate (Arg **arg_list*, Cardinal *arg_count*)

Inputs

arg_list	Specifies an argument list, consisting of resource name/value pairs.
arg_count	Specifies the number of arguments in arg_list.

Returns

An allocated parse mapping.

Availability

Motif 2.0 and later.

Description

`XmParseMappingCreate`() creates a parse mapping, which is an entry in a parse table. A parse mapping consists minimally of a match pattern, and a substitution pattern or procedure, which can be used by string parsing functions in order to compare against and subsequently transform text. A parse mapping is created through a resource style argument list, where *arg_list* is an array of resource name/value pairs, and *arg_count* is the number of such pairs.

Usage

A parse table is an array of parse mappings. `XmParseMappingCreate`() creates a parse mapping using a resource style parameter list. The parse table can subsequently be passed to `XmStringParseText`() in order to filter or modify an input string.

`XmParseMappingCreate`() allocates storage associated with the returned parse mapping object. It is the responsibility of the programmer to free the allocated memory by a call to `XmParseMappingFree`() at the appropriate moment.

Example

The following code fragment creates a parse mapping which performs a simple swap of occurrences of two characters within an input string:

```
char *swapover ( char   *input,     /* input string */
                 char   *a,         /* only first character in array used */
                 char   *b)         /* only first character in array used */
{
    XmString            tmp;
    XmParseMapping      parse_mapping;
```

```
XmParseTable        parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
Cardinal            parse_table_index = 0;
Arg                 argv[4];
Cardinal            argc   = 0;
char                *output = (char *) 0;

/* create a XmParseMapping object to swap *a with *b */

argc = 0;
tmp = XmStringCreateLocalized (a);
XtSetArg (argv[argc], XmNincludeStatus,     XmINSERT);
argc++;
XtSetArg (argv[argc], XmNsubstitute,        tmp);
argc++;
XtSetArg (argv[argc], XmNpattern,           b);
argc++;
XtSetArg (argv[argc], XmNpatternType,       XmCHARSET_TEXT);
argc++;
parse_mapping = XmParseMappingCreate (argv, argc);
parse_table[parse_table_index++] = parse_mapping;
XmStringFree (tmp);

/* create a XmParseMapping object to swap *b with *a */
argc = 0;
tmp = XmStringCreateLocalized (b);
XtSetArg (argv[argc], XmNincludeStatus,     XmINSERT);
argc++;
XtSetArg (argv[argc], XmNsubstitute,        tmp);
argc++;
XtSetArg (argv[argc], XmNpattern,           a);
argc++;
XtSetArg (argv[argc], XmNpatternType,       XmCHARSET_TEXT);
argc++;
parse_mapping = XmParseMappingCreate (argv, argc);
parse_table[parse_table_index++] = parse_mapping;
XmStringFree (tmp);

/* substitute using the XmParseMapping. */

tmp = XmStringParseText ((XtPointer) input, NULL, NULL,
                         XmCHARSET_TEXT,
                         parse_table, parse_table_index, NULL);
XmParseTableFree (parse_table, parse_table_index);
```

```
    /* convert XmString to String */
    if (tmp != (XmString) 0) {
        output = (char *) XmStringUnparse (tmp, NULL,
                                           XmCHARSET_TEXT,
                                           XmCHARSET_TEXT, NULL,
                                           0, XmOUTPUT_ALL);[1]
        XmStringFree (tmp);
    }

    return output;
}
```

See Also

`XmParseMappingFree(1), XmParseMappingGetValues(1), XmParseMappingSetValues(1), XmParseTableFree(1), XmStringParseText(1), XmStringUnparse(1), XmParseMapping(2).`

1.The code sample in the 2nd edition used `XmStringGetLtoR`() to convert the compound string. `XmStringGetLtoR`() is deprecated as of Motif 2.0.

Name

XmParseMappingFree – free the memory used by a parse mapping.

Synopsis

void XmParseMappingFree (XmParseMapping *parse_mapping*)

Inputs

parse_mapping Specifies a parse mapping.

Availability

Motif 2.0 and later.

Description

`XmParseMappingFree`() deallocates storage used by the specified parse mapping object.

Usage

The XmParseMapping type is opaque, and represents an entry in a parse table, which can be used for transforming text. A parse mapping is created by `XmParseMappingCreate`(), which allocates storage for the object represented by the type, and it is the responsibility of the programmer to reclaim the memory when the parse mapping is no longer required.

It is important to call `XmParseMappingFree`() rather than `XtFree`() upon redundant parse mappings, otherwise compound strings internally referenced by the object are not deallocated.

See Also

`XmParseMappingCreate`(1), `XmParseMappingGetValues`(1), `XmParseMappingSetValues`(1), `XmParseTableFree`(1), `XmStringParseText`(1), `XmParseMapping`(2).

Name

XmParseMappingGetValues – fetch resources from a parse mapping object.

Synopsis

```
void XmParseMappingGetValues (  XmParseMapping    parse_mapping,
                                Arg               *arg_list,
                                Cardinal          arg_count)
```

Inputs

parse_mapping Specifies a parse mapping object.

arg_count Specifies the number of arguments in the list arg_list.

Outputs

arg_list Specifies the argument list of name/value pairs that contain the resource names and addresses into which the resource values are to be stored.

Availability

Motif 2.0 and later.

Description

`XmParseMappingGetValues`() fetches selected attributes from *parse_mapping*. The set of attributes retrieved is specified through the resource list *arg_list*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *arg_count*.

Usage

If the XmNsubstitute attribute of the parse mapping is retrieved, the procedure returns a copy of the internal value. It is the responsibility of the programmer to recover the allocated space at a suitable point by calling `XmStringFree`().

Example

The following code illustrates fetching the values from an XmParseMapping:

```
XtPointer           pattern;
XmTextType          pattern_type;
XmString            substitute;
XmParseProc         parse_proc;
XtPointer           client_data;
XmIncludeStatus     include_status;
Arg                 argv[6];
Cardinal            argc = 0;

/* construct a resource-style argument list for all XmParseMapping values */
XtSetArg (argv[argc], XmNpattern,       &pattern);      argc++;
XtSetArg (argv[argc], XmNpatternType,   &pattern_type); argc++;
```

```
XtSetArg (argv[argc], XmNsubstitute,          &substitute);      argc++;
XtSetArg (argv[argc], XmNinvokeParseProc,     &parse_proc);      argc++;
XtSetArg (argv[argc], XmNclientData,          &client_data);     argc++;
XtSetArg (argv[argc], XmNincludeStatus,       &include_status);  argc++;

/* fetch the values. parse_mapping here is an unspecified XmParseMapping */
XmParseMappingGetValues (parse_mapping, argv, argc);
...
/* XmParseMappingGetValues returns a copy of the XmNsubstitute value */
/* which must be freed when no longer required by the application    */
XmStringFree (substitute);
```

See Also

`XmParseMappingCreate`(1), `XmParseMappingFree`(1), `XmParseMappingSetValues`(1), `XmParseTableFree`(1), `XmParseMapping`(2).

Name

XmParseMappingSetValues – sets resources for a parse mapping object.

Synopsis

```
void XmParseMappingSetValues (  XmParseMapping   parse_mapping,
                                Arg              *arg_list,
                                Cardinal         arg_count)
```

Inputs

parse_mapping	Specifies a parse mapping object.
arg_list	Specifies the list of name/value pairs containing resources to be modified.
arg_count	Specifies the number of arguments in the list *arg_list*.

Availability

Motif 2.0 and later.

Description

`XmParseMappingSetValues`() sets selected attributes within *parse_mapping*. The set of attributes which is modified is specified through the resource list *arg_list*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *arg_count*.

Usage

If the XmNsubstitute attribute of the parse mapping is set, the procedure internally takes a copy of the supplied value. It is the responsibility of the programmer to recover the allocated space at a suitable point by calling `XmStringFree`().

Example

The following skeleton code illustrates changing the values of a parse mapping:

```
XmIncludeStatus map_tab (  XtPointer        *in_out,
                           XtPointer        text_end,       /* unused
*/
                           XmTextType       type,           /* unused
*/
                           XmStringTag      tag,            /* unused
*/
                           XmParseMapping   entry,          /* unused
*/
                           int              pattern_length, /* unused
*/
                           XmString         *str_out,
```

```
                                XtPointer       call_data)      /* unused
*/
{
    /* Insert an XmString Tab component into the output stream */
    *str_out = XmStringComponentCreate (XmSTRING_COMPONENT_TAB,
    0, NULL);
    *in_out = (*in_out + 1);

    return XmINSERT;

}

/* change a parse mapping to invoke the above parse procedure */
void set_parse_tab_mapping (XmParseMapping parse_mapping)
{
    Arg         argv[4];
    Cardinal    argc = 0;

    /* construct resource-style argument list for XmParseMapping values */
    XtSetArg (argv[argc], XmNpattern,           "\t");
    argc++;
    XtSetArg (argv[argc], XmNpatternType,       XmCHARSET_TEXT);
    argc++;
    XtSetArg (argv[argc], XmNincludeStatus,     XmINVOKE);
    argc++;
    XtSetArg (argv[argc], XmNinvokeParseProc,   map_tab);
    argc++;

    /* change the values */
    XmParseMappingSetValues (parse_mapping, argv, argc);
}
```

See Also

`XmParseMappingCreate`(1), `XmParseMappingFree`(1), `XmParseMappingGetValues`(1), `XmParseTableFree`(1), `XmParseMapping`(2),

Name

XmParseTableFree – free the memory used by a parse table.

Synopsis

void XmParseTableFree (XmParseTable *parse_table*, Cardinal *parse_count*)

Inputs

parse_table	Specifies a parse table.
parse_count	Specifies the number of entries in the parse table.

Availability

Motif 2.0 and later.

Description

`XmParseTableFree`() deallocates storage used by the specified *parse_table*. In addition, the function deallocates storage used by any parse mapping elements of the table. *parse_count* indicates the number of mapping elements within the table.

Usage

A parse table is an array of XmParseMapping objects. The XmParseMapping is an opaque type, which is used when transforming text. Each parse mapping object allocates memory in addition to any memory allocated by the parse table array. It is important to call `XmParseTableFree`() rather than `XtFree`() when deallocating storage associated with a parse table, otherwise objects constituent within the array, and compound strings internally referenced by the parse mapping objects, are not deallocated. The function should be called when a parse table is no longer needed.

Example

```
/* Allocate a parse table */
XmParseTable        parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
Cardinal            parse_table_index = 0;
XmParseMapping      parse_mapping;
Arg                 argv[MAX_ARGS];
Cardinal            argc = 0;

/* Create a XmParseMapping object */
argc = 0;
...
parse_mapping = XmParseMappingCreate (argv, argc);

/* Insert into parse table */
parse_table[parse_table_index++] = parse_mapping;
```

```
/* Create another XmParseMapping object */
argc = 0;
...
parse_mapping = XmParseMappingCreate (argv, argc);

/* Insert into parse table */
parse_table[parse_table_index++] = parse_mapping;

/* Use the XmParseTable. */
tmp = XmStringParseText ((XtPointer) input, NULL, NULL,
                         XmCHARSET_TEXT, parse_table,
                         parse_table_index, NULL);

/* Free the parse table: this also frees the parse mappings */
XmParseTableFree (parse_table, parse_table_index);
```

See Also

`XmParseMappingCreate`(1), `XmParseMappingFree`(1), `XmParseMappingGetValues`(1), `XmParseMappingSetValues`(1), `XmParseMapping`(2).

Name

XmPrintPopupPDM – notify the Print Display Manager.

Synopsis

#include <Xm/Print.h>

XtEnum XmPrintPopupPDM (Widget *print_shell*, Widget *video_shell*)

Inputs

print_shell Specifies a PrintShell widget.
video_shell Specifies the widget on whose behalf the PDM dialog is required.

Returns

Returns XmPDM_NOTIFY_SUCCESS if the PDM was notified, XmPDM_NOTIFY_FAIL otherwise.

Availability

Motif 2.1 and later.

Note that not all operating system vendors incorporate the XmPrintShell within the native Motif toolkit.[1]

Description

`XmPrintPopupPDM`() sends a notification to start a Print Display Manager for the application. The notification is issued to either the display associated with *print_shell*, or the display of *video_shell*, depending upon the value of the environment variable XPDMDISPLAY. XPDMDISPLAY can only be set to "print" or "video". If the value is "print", the notification is sent to the display of *print_shell*, and similarly the value "video" sends the notification to the display of *video_shell*. If the notification could be sent, the function returns XmPDM_NOTIFY_SUCCESS, otherwise the return value is XmPDM_NOTIFY_FAIL.

Usage

`XmPrintPopupPDM`() is a convenience function which issues a notification through the X selection mechanisms in order to start a Print Dialog Manager. The notification is issued asynchronously: the return value XmPDM_NOTIFY_SUCCESS indicates that the message has successfully been issued, not that any PDM is now initialized. In order to track the status of the PDM, the programmer registers an XmNpdmNotificationCallback with the widget *print_shell*, which must be an instance of the PrintShell widget class. To ensure that the contents of the video_shell is not modified whilst the PDM is ini-

1.Sun Solaris being a case in point.

tializing, XmPrintPopupPDM() creates an input-only window over the top of *video_shell*, and the window is only removed when the PDM indicates that it is present, or if the selection XmIPDM_START times out. The timeout period is set at two minutes.

See Also

XmPrintSetup(1), XmPrintToFile(1), XmRedisplayWidget(1), XmPrintShell(2).

Name

XmPrintSetup – create a Print Shell widget.

Synopsis

```
#include <Xm/Print.h>

Widget XmPrintSetup (  Widget      video_widget,
                       Screen      *print_screen,
                       String      name
                       ArgList     arg_list,
                       Cardinal    arg_count)
```

Inputs

video_widget	Specifies a widget from which video application data is fetched.
print_screen	Specifies the screen on which the PrintShell is created.
name	Specifies the name of the created PrintShell.
arg_list	Specifies an argument list of name/value pairs that contain resources for the PrintShell.
arg_count	Specifies the number of arguments in the list arg_list.

Returns

The created PrintShell, or NULL if no ApplicationShell can be found from *video_widget*,

Availability

Motif 2.1 and later.

Note that not all operating system vendors incorporate the PrintShell in their native toolkit.[1]

Description

`XmPrintSetup`() creates a PrintShell widget with the given *name* on the screen *print_screen*. The new PrintShell is returned to the application. Resources which configure the new print shell are supplied through an array of structures which contain name/value pairs. The array of resources is *arg_list*, and the number of items in the array is *arg_count*.

Usage

`XmPrintSetup`() creates a new ApplicationShell on the screen specified by *print_screen*, and thereafter creates a PrintShell as a popup child. The new ApplicationShell is created with the same name and class as the ApplicationShell from which *video_widget* is descended. The XmNmappedWhenManaged resource of

1.For example, Sun Solaris includes the headers, but does not compile the widget into the Motif library.

the PrintShell is set to False under the assumption that subsequent notification of the start of a job or page is the correct time to map the widget. The print shell is finally realized, and returned.

See Also

`XmPrintPopupPDM`(1), `XmPrintToFile`(1), `XmRedisplayWidget`(1), `XmPrintShell`(2).

Name

XmPrintToFile – save X Print Server data to file.

Synopsis

```
#include <Xm/Print.h>

XtEnum XmPrintToFile (  Display        *display,
                        String         file_name,
                        XPFinishProc   finish_proc,
                        XPointer       client_data)
```

Inputs

display	Specifies the print connection to the X server.
file_name	Specifies the name of the file to contain the print output.
finish_proc	Specifies a procedure called when printing is finished.
client_data	Specifies application data to be passed to finish_proc.

Returns

True if printing can be initiated, otherwise False.

Availability

Motif 2.1 and later.

Note that not all operating system vendors incorporate the XmPrintShell in their native toolkits.[1]

Description

`XmPrintToFile`() is a convenience function which provides a simple interface onto the X Print mechanisms, in order to save print data to the file *file_name*. Printing takes place asynchronously, and the programmer receives notification of the status of the printing task by supplying *finish_proc*, which is called when the task is finished. The *display* parameter is the print connection to the X server, and is used to deduce an application name and class.

Usage

If `XmPrintToFile`() cannot open the file *file_name* for writing, create a pipe, or fork off a child process, the procedure returns False. An application name and class is deduced using the *display* parameter, and these are used by the child process, which creates a new application context, and opens a new display connection using the same name and class as the application process. Data is retrieved from the X server through a call to `XpGetDocumentData`(). The parent process does not wait for the child to complete, but returns immediately after

1.For example, Sun Solaris supply the widget headers, but do not compile the component into the Motif library.

initiating the child process. The return value True therefore does not mean that the print task is complete, merely that the task is initiated.

The application is notified of task completion by supplying an XPFinishProc. The *status* parameter passed to the finish procedure when the task is completed is set to XPGetDocFinished on successful completion. If for any reason the child process fails to print the data, the file *file_name* is both closed and removed. The file is closed in any case prior to calling the XPFinishProc.

`XpStartJob`() must be called by the application before `XmPrintToFile`() can be called.

Structures

An XPFinishProc is specified as follows:

```
typedef void (*XPFinishProc)( Display          *display,
                              XPContext        context,
                              XPGetDocStatus   status,
                              XPointer         client_data);
```

If status is XPGetDocFinished, the print task has completed successfully.

See Also

`XmPrintPopupPDM`(1), `XmPrintSetup`(1), `XmRedisplayWidget`(1), `XmPrintShell`(2).

Name

XmProcessTraversal – set the widget that has the keyboard focus.

Synopsis

Boolean XmProcessTraversal (Widget *widget*, XmTraversalDirection *direction*)

Inputs

widget	Specifies the widget whose hierarchy is to be traversed.
direction	Specifies the direction in which to traverse the hierarchy. Pass one of the values from the list below.

Returns

True on success or False otherwise.

Description

`XmProcessTraversal`() causes the input focus to change to another widget under application control, rather than as a result of keyboard traversal events from a user. *widget* specifies the widget whose hierarchy is traversed up to the shell widget. If that shell has the keyboard focus, `XmProcessTraversal`() changes the keyboard focus immediately. If that shell does not have the focus, the routine does not have an effect until the shell receives the focus.

The *direction* argument specifies the nature of the traversal to be made. In each case, the routine locates the hierarchy that contains the specified widget and then performs the action that is particular to the *direction*. If the new setting succeeds, `XmProcessTraversal`() returns True. The routine returns False if the keyboard focus policy is not XmEXPLICIT, if no traversable items exist, or if the arguments are invalid.

Usage

For XmTRAVERSE_CURRENT, if the tab group that contains *widget* is inactive, it is made the active tab group. If *widget* is in the active tab group, it is given the keyboard focus; if *widget* is the active tab group, the first traversable item in it is given the keyboard focus. For XmTRAVERSE_UP, XmTRAVERSE_DOWN, XmTRAVERSE_LEFT, and XmTRAVERSE_RIGHT, in the hierarchy that contains *widget*, the item in the specified *direction* from the active item is given the keyboard focus. For XmTRAVERSE_NEXT and XmTRAVERSE_PREV, in the hierarchy that contains *widget*, the next and previous items in child order from the active item are given keyboard focus. For XmTRAVERSE_HOME, in the hierarchy that contains widget, the first traversable item is given the keyboard focus. For XmTRAVERSE_NEXT_TAB_GROUP and XmTRAVERSE_PREV_TAB_GROUP, in the hierarchy that contains *widget*, the next and previous tab groups from the active tab group are given the keyboard focus.

In Motif 2.0 and later, new XmTraversalDirection values XmTRAVERSE_GLOBALLY_FORWARD and XmTRAVERSE_GLOBALLY_BACKWARD are provided in order to implement the XmDisplay resource XmNenableButtonTab. If enabled, for XmTRAVERSE_GLOBALLY_FORWARD navigation proceeds to the next (or downwards, depending upon orientation) item within the current tab group, unless the current location is the last item in the group, when navigation is into the next tab group. Similarly, for XmTRAVERSE_GLOBALLY_BACKWARD navigation proceeds to the previous (or upwards) item in the current tab group, unless the current location is the first item in the group, when navigation is into the previous tab group. The interpretation of the *direction* values XmTRAVERSE_GLOBALLY_FORWARD and XmTRAVERSE_GLOBALLY_BACKWARD is reversed where XmNlayoutDirection is XmRIGHT_TO_LEFT.

`XmProcessTraversal`() does not allow traversal to widgets in different shells or widgets that are not mapped. Calling `XmProcessTraversal`() inside a XmNfocusCallback causes a segmentation fault.

Example

The following code fragments shows the use of `XmProcessTraversal`() as a callback routine for a text widget. When the user presses the Return key, the keyboard focus is advanced to the next input area:

```
Widget form, label, text;

form = XtVaCreateWidget ("form", xmFormWidgetClass, parent,
                         XmNorientation, XmHORIZONTAL,
                         NULL);
label = XtVaCreateManagedWidget ("label", xmLabelGadgetClass, form,
                         XmNleftAttachment,   XmATTACH_FORM,
                         XmNtopAttachment,    XmATTACH_FORM,
                         XmNbottomAttachment, XmATTACH_FORM,
                         NULL);
text = XtVaCreateManagedWidget ("text", xmTextWidgetClass, form,
                         XmNleftAttachment,
                         XmATTACH_WIDGET,
                         XmNleftWidget,       label,
                         XmNtopAttachment,    XmATTACH_FORM,
                         XmNrightAttachment,  XmATTACH_FORM,
                         XmNbottomAttachment, XmATTACH_FORM,
                         NULL);
XtAddCallback (text, XmNactivateCallback,
```

```
                    XmProcessTraversal, (XtPointer)
                    XmTRAVERSE_NEXT_TAB_GROUP);
    XtManageChild (form);
```

Structures

The possible values for direction are:

XmTRAVERSE_CURRENT	XmTRAVERSE_NEXT
XmTRAVERSE_UP	XmTRAVERSE_PREV
XmTRAVERSE_DOWN	XmTRAVERSE_HOME
XmTRAVERSE_LEFT	
XmTRAVERSE_NEXT_TAB_GROUP	
XmTRAVERSE_RIGHT	
XmTRAVERSE_PREV_TAB_GROUP	
XmTRAVERSE_GLOBALLY_FORWARD	
XmTRAVERSE_GLOBALLY_BACKWARD	

See Also

`XmGetFocusWidget`(1), `XmGetTabGroup`(1), `XmGetVisibility`(1), `XmIsTraversable`(1).

Name

XmRedisplayWidget – force widget exposure for printing.

Synopsis

#include <Xm/Print.h>

void XmRedisplayWidget (Widget *widget*)

Inputs

widget Specifies the widget to redisplay.

Availability

Motif 2.1 and later.

Note that not all operating system vendors compile the XmPrintShell into their native Motif toolkits.[1]

Description

`XmRedisplayWidget`() forces widget to redisplay itself by invoking the expose method of the *widget*. The routine is a convenience function which hides the internals of the X11R6 Xp mechanisms, which use *widget* exposure in order to implement printing.

Usage

`XmRedisplayWidget`() constructs a region which corresponds precisely to the location and area occupied by a widget. The expose method of the widget is called directly using the region in order to redisplay the widget. `XmRedisplayWidget`() is synchronous in effect. Asynchronous printing is performed by creating a PrintShell, and specifying XmNstartJobCallback, XmNendJobCallback, and XmNpageSetupCallback procedures which are invoked in response to X Print events as they arrive.

`XmRedisplayWidget`() is not multi-thread safe, nor is the *widget* parameter fully validated: it is implicitly assumed to be the descendant of a PrintShell.

1.Sun Solaris supplied the widget headers, but the widget itself is compiled out of the Motif library.

Example

The following code synchronously prints the contents of a text widget:

```
Widget     app_shell, app_text;
Screen     print_screen;
Display    print_display;
Widget     print_shell, print_form, print_text;
short      rows;
int        lines, pages, page;
char       *data;
...
/* create a connection to the X Print server */
print_shell = XmPrintSetup (app_shell, print_screen, "PrintShell", NULL, 0);

/* create a suitable print hierarchy */
print_form = XmCreateForm (print_shell,...);
print_text = XmCreateText (print_form,...);

/* configure and manage the print hierarchy */
...
/* copy the video text to the print text     */
/* what is copied depends upon whether it is */
/* contents and/or visuals that are printed */
...
data = XmTextGetString (app_text);
XmTextSetString (print_text, data);
XtFree (data);
...
/* start a print job */
print_display = XtDisplay (print_shell);
XpStartJob (print_display, XPSpool);

/* deduce number of logical pages in the print text widget */
XtVaGetValues (print_text, XmNrows, &rows, XmNtotalLines, &lines, 0);

for (page = 0, pages = lines / rows; page < pages; page++) {
    /* start of page notification */
    XpStartPage (print_display, XtWindow (print_shell), False);

    /* force the print text to expose itself */
    XmRedisplayWidget (print_text);

    /* end of page notification */
    XpEndPage (print_display);

    /* scroll to next page */
```

```
        XmTextScroll (print_text, rows);
}

/* end of print job notification */
XpEndJob (print_display);
...
```

See Also

`XmPrintPopupPDM`(1), `XmPrintSetup`(1), `XmPrintToFile`(1), `XmPrintShell`(2).

Name

XmRegisterSegmentEncoding – register a compound text encoding format for a font list element tag.

Synopsis

char *XmRegisterSegmentEncoding (char **fontlist_tag*, char **ct_encoding*)

Inputs

fontlist_tag	Specifies the compound string font list element tag.
ct_encoding	Specifies the compound text character set.

Returns

The old compound text encoding format for a previously-registered font list element tag or NULL for a new font list element tag.

Availability

Motif 1.2 and later.

Description

`XmRegisterSegmentEncoding`() registers the specified compound text encoding format *ct_encoding* for the specified *fontlist_tag*. Both *fontlist_tag* and *ct_encoding* must be NULL-terminated ISO8859-1 strings. If the font list tag is already associated with a compound text encoding format, registering the font list tag again overwrites the previous entry and the routine returns the previous compound text format. If the font list tag is has not been registered before, the routine returns NULL. If *ct_encoding* is NULL, the font list tag is unregistered. If *ct_encoding* is the reserved value XmFONTLIST_DEFAULT_TAG, the font list tag is mapped to the code set of the current locale. `XmRegisterSegmentEncoding`() allocates storage if the routine returns a character string; the application is responsible for freeing the storage using `XtFree`().

Usage

Compound text is an encoding that is designed to represent text from any locale. Compound text strings identify their encoding using embedded escape sequences. The compound text representation was standardized for X11R4 for use as a text interchange format for interclient communication.

`XmCvtXmStringToCT`() converts a compound string into compound text. The routine uses the font list tag of each compound string segment to select a compound text format for the segment. A mapping between font list tags and compound text encoding formats is stored in a registry. `XmRegisterSegmentEncoding`() provides a way for an application to map particular font list element tags to compound text encoding formats.

See Also

`XmCvtXmStringToCT`(1), `XmMapSegmentEncoding`(1).

Name

XmRemoveFromPostFromList – make a menu inaccessible from a widget.

Synopsis

#include <Xm/RowColumn.h>

void XmRemoveFromPostFromList (Widget *menu*, Widget *widget*)

Inputs

menu	Specifies a menu widget
widget	Specifies the widget which no longer posts menu.

Availability

In Motif 2.0 and later, the functional prototype is removed from RowColumn.h, although there is otherwise no indication that the procedure is obsolete. [1]

Description

`XmRemoveFromPostFromList`() is the inverse of the procedure `XmAddToPostFromWidget`(). The menu hierarchy associated with *menu* is made inaccessible from *widget*.

Usage

If the type of menu is XmMENU_PULLDOWN, the XmNsubMenuId resource of widget is set to NULL. If the type of menu is XmMENU_POPUP, event handlers presumably added to widget by XmAddToPostFromWidget() in order to post the menu are removed.

No check is made to ensure that the XmNsubMenuId resource of widget is originally set to menu before clearing the value. Passing the wrong menu into the procedure can therefore have unwanted effects. There are implicit assumptions that widget is a CascadeButton or CascadeButtonGadget when menu is XmMENU_PULLDOWN, and that widget is not a Gadget when menu is XmMENU_POPUP. These are not checked by the procedure.

See Also

`XmAddToPostFromList`(1), `XmGetPostedFromWidget`(1), `XmPopupMenu`(2), `XmPulldownMenu`(2), `XmRowColumn`(2).

1.This is true of Motif 2.1.10, although the header reference is restored in the OpenMotif 2.1.30.

Name

XmRemoveProtocolCallback – remove client callback from a protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmRemoveProtocolCallback (  Widget          shell,
                                 Atom            property,
                                 Atom            protocol,
                                 XtCallbackProc  callback,
                                 XtPointer       closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocol	Specifies the protocol atom.
callback	Specifies the procedure that is to be removed.
closure	Specifies any client data that is passed to the callback.

Description

`XmRemoveProtocolCallback`() removes the specified *callback* from the list of callback procedures that are invoked when the client message corresponding to *protocol* is received.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. To communicate using a protocol, a client sends a ClientMessage event containing a *property* and *protocol*, and the receiving client responds by calling the associated protocol *callback* routine. `XmRemoveProtocolCallback`() allows you to unregister one of these callback routines. The inverse routine is `XmAddProtocolCallback`().

See Also

`XmAddProtocolCallback`(1), `XmInternAtom`(1), `XmRemoveWMProtocolCallback`(1), `VendorShell`(2).

Name

XmRemoveProtocols – remove protocols from the protocol manager.

Synopsis

#include <Xm/Protocols.h>

void XmRemoveProtocols (Widget *shell*, Atom *property*, Atom **protocols*, Cardinal *num_protocols*)

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocols	Specifies a list of protocol atoms.
num_protocols	Specifies the number of atoms in protocols.

Description

`XmRemoveProtocols`() removes the specified *protocols* from the protocol manager and deallocates the internal tables for the protocols. If the specified *shell* is realized and at least one of the *protocols* is active, the routine also updates the handlers and the *property*. The inverse routine is `XmAddProtocols`().

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. `XmRemoveProtocols`() allows you eliminate protocols that can be understood by your application. The inverse routine is XmAddProtocols().

See Also

`XmAddProtocols`(1), `XmInternAtom`(1), `XmRemoveWMProtocols`(1), `VendorShell`(2).

Name

XmRemoveTabGroup – remove a widget from a list of tab groups.

Synopsis

void XmRemoveTabGroup (Widget *tab_group*)

Inputs

tab_group Specifies the widget to be removed.

Availability

In Motif 1.1, `XmRemoveTabGroup`() is obsolete. It has been superseded by setting XmNnavigationType to XmNONE.

Description

`XmRemoveTabGroup`() removes the specified *tab_group* widget from the list of tab groups associated with the widget hierarchy. This routine is retained for compatibility with Motif 1.0 and should not be used in newer applications. If traversal behavior needs to be changed, this should be done by setting the XmNnavigationType resource directly.

Usage

A tab group is a group of widgets that can be traversed using the keyboard rather than the mouse. Users move from widget to widget within a single tab group by pressing the arrow keys. Users move between different tab groups by pressing the Tab or Shift-Tab keys. The inverse routine is XmAddTabGroup().

See Also

`XmAddTabGroup`(1), `XmGetTabGroup`(1), `XmManager`(2), `XmPrimitive`(2).

Name

XmRemoveWMProtocolCallback – remove client callbacks from a XA_WM_PROTOCOLS protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmRemoveWMProtocolCallback (  Widget          shell,
                                   Atom            protocol,
                                   XtCallbackProc  callback,
                                   XtPointer       closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
protocol	Specifies the protocol atom.
callback	Specifies the procedure that is to be removed.
closure	Specifies any client data that is passed to the callback.

Description

`XmRemoveWMProtocolCallback`() is a convenience routine that calls `XmRemoveProtocolCallback`() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. To communicate using a protocol, a client sends a ClientMessage event containing a *property* and *protocol*, and the receiving client responds by calling the associated protocol *callback* routine. `XmRemoveWMProtocolCallback`() allows you to unregister one of these *callback* routines with the window manager *protocol* property. The inverse routine is `XmAddWMProtocolCallback`().

See Also

`XmAddProtocolCallback`(1), `XmAddWMProtocolCallback`(1), `XmInternAtom`(1), `XmRemoveProtocolCallback`(1), `VendorShell`(2).

Name

XmRemoveWMProtocols – remove the XA_WM_PROTOCOLS protocols from the protocol manager.

Synopsis

#include <Xm/Protocols.h>

void XmRemoveWMProtocols (Widget *shell*, Atom **protocols*, Cardinal *num_protocols*)

Inputs

shell	Specifies the widget associated with the protocol property.
protocols	Specifies a list of protocol atoms.
num_protocols	Specifies the number of atoms in protocols.

Description

`XmRemoveWMProtocols`() is a convenience routine that calls `XmRemoveProtocols`() with property set to XA_WM_PROTOCOL, the window manager protocol property. The inverse routine is `XmAddWMProtocols`().

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. `XmRemoveWMProtocols`() allows you to remove this protocol so that it is no longer understood by your application. The inverse routine is `XmAddWMProtocols`().

See Also

`XmAddProtocols`(1), `XmAddWMProtocols`(1), `XmInternAtom`(1), `XmRemoveProtocols`(1), `VendorShell`(2).

Name

XmRenderTableAddRenditions – add renditions to a render table.

Synopsis

```
XmRenderTable XmRenderTableAddRenditions (  XmRenderTable
old_table,
                                            XmRendition
*new_renditions,
                                            Cardinal
new_rendition_count,
                                            XmMergeMode
merge_mode)
```

Inputs

old_table	Specifies a render table.
new_renditions	Specifies an array of renditions to merge with the render table.
new_rendition_count	Specifies the number of renditions in the array.
merge_mode	Specifies the action to take if entries have the same tag.

Returns

The newly allocated merged render table.

Availability

Motif 2.0 and later.

Description

A render table is a set of renditions which can be used to specify the way in which XmStrings are drawn. `XmRenderTableAddRenditions`() creates a new render table by merging the list of renditions specified by *new_renditions* into the renditions contained within *old_table*. If a rendition with the same tag is found in both *old_table* and *new_renditions*, *merge_mode* is used to give precedence. The new render table is returned.

If *old_table* is NULL, a new render table is allocated which contains only the renditions of *new_renditions*. If *new_renditions* is NULL or *new_rendition_count* is zero, the *old_table* is returned unmodified. If a rendition within *old_table* has the same tag as one within *new_renditions*, *merge_mode* determines how to resolve the conflict. If *merge_mode* is XmMERGE_REPLACE, the rendition within *old_table* is ignored, and the rendition within *new_renditions* is added to the new table. If the mode is XmMERGE_SKIP, the new table contains the rendition from *old_table*, and that from *new_renditions* is ignored. If the mode is XmMERGE_NEW, the rendition within *new_renditions* is used, except that

where any resources of the rendition are unspecified, the value is copied from the matching rendition from the *old_table*. A resource is unspecified if the value is XmAS_IS or NULL. Lastly, if the mode is XmMERGE_OLD, it is the *old_table* rendition which is added to the new table, and any unspecified resources are taken from the new rendition.

Usage

The reference count for the original table is decremented and deallocated where necessary, and a newly allocated render table containing the merged data is returned. It is the responsibility of the programmer to reclaim the allocated memory for the returned render table by calling `XmRenderTableFree`() at a suitable point.

Example

The following specimen code creates a set of renditions and merges them into an unspecified render table:

```
XmRendition     new_renditions[2];
XmRenderTable   new_table;
Arg             argv[4];
Cardinal        argc = 0;
Pixel           fg =...;
Pixel           bg =...;

XtSetArg (argv[argc], XmNfontName,      "fixed");
argc++;
XtSetArg (argv[argc], XmNfontType,      XmFONT_IS_FONT);
argc++;
XtSetArg (argv[argc], XmNloadModel,     XmLOAD_DEFERRED);
argc++;
new_renditions[0] = XmRenditionCreate (widget,
XmFONTLIST_DEFAULT_TAG, argv, argc);

argc = 0;
XtSetArg (argv[argc], XmNrenditionBackground, bg); argc++;
XtSetArg (argv[argc], XmNrenditionForeground, fg); argc++;
new_renditions[1] = XmRenditionCreate (widget, "colors", argv, argc);
new_table = XmRenderTableAddRenditions (old_table, new_renditions, 2,
XmMERGE_REPLACE);)
```

See Also

XmRenderTableCopy(1), XmRenderTableFree(1), XmRenderTableGetRendition(1), XmRenderTableGetRenditions(1), XmRenderTableGetTags(1), XmRenderTableRemoveRenditions(1), XmRenditionCreate(1), XmRenditionFree(1), XmRenditionRetrieve(1), XmRenditionUpdate(1), XmRendition(2).

Name

XmRenderTableCopy – copy a render table.

Synopsis

XmRenderTable XmRenderTableCopy (XmRenderTable *old_table*, XmStringTag **tags*, int *tag_count*)

Inputs

old_table	Specifies the table containing the renditions to be copied.
tags	Specifies an array of tags. Renditions with matching tags are copied.
tag_count	Specifies the number of items within the tags array.

Returns

A new render table containing renditions with matching tags, or NULL.

Availability

Motif 2.0 and later.

Description

An XmRenderTable is an array of XmRendition objects, which are used to render compound strings. `XmRenderTableCopy`() creates a newly allocated render table by copying renditions from an existing table, *old_table*. An array of tags can be supplied which acts as a filter: only those renditions from *old_table* which have a matching XmNtag resource are copied. The number of items within any tags array is specified through *tag_count*. If *tags* is NULL, all of the renditions within *old_table* are copied. If *old_table* is NULL, the function returns NULL.

Usage

The function allocates storage for the returned render table, including storage for each of the newly copied renditions. It is the responsibility of the programmer to reclaim the memory at an appropriate point by calling `XmRenderTableFree`().

In Motif 2.0 and later, the XmRenderTable supersedes the XmFontList, which is now considered obsolete. For backwards compatibility, the XmFontList opaque type is implemented through the render table.

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableFree`(1), `XmRenderTableGetRendition`(1), `XmRenderTableGetRenditions`(1), `XmRenderTableGetTags`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRenditionRetrieve`(1), `XmRenditionUpdate`(1), `XmRendition`(2).

Name

XmRenderTableCvtFromProp – convert from a string representation into a render table.

Synopsis

XmRenderTable XmRenderTableCvtFromProp (Widget *widget*, char **property*, unsigned int *length*)

Inputs

widget	Specifies a destination widget in a data transfer.
property	Specifies the render table in string representation format.
length	Specifies the number of bytes in the property string.

Returns

The converted render table.

Availability

Motif 2.0 and later.

Description

`XmRenderTableCvtFromProp`() converts a string representation of a render table into an XmRenderTable. The string representation to be converted is given by *property*, and the size of the string in bytes is *length*.

Usage

Typically, the procedure is used within the destination callback of widget when it is the target of a data transfer. The inverse function `XmRenderTableCvtToProp`() is called by the convert procedures of the source of the data transfer. `XmRenderTableCvtFromProp`() returns allocated memory, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmRenderTableFree`().

See Also

`XmRenderTableCvtToProp`(1), `XmRenderTableFree`(1), `XmRendition`(2).

Name

XmRenderTableCvtToProp – convert a render table into a string representation.

Synopsis

unsigned int XmRenderTableCvtToProp (Widget *widget*,
XmRenderTable *render_table*,
char
***property_return*)

Inputs

widget	Specifies a source widget for the render table.
render_table	Specifies the render table to convert.

Outputs

property_return	Returns the string representation of the converted render table.

Returns

The number of bytes in the converted string representation.

Availability

Motif 2.0 and later.

Description

`XmRenderTableCvtToProp`() converts an XmRenderTable render_table into a string representation at the address specified by *property_return*. The length of the converted string is returned.

Usage

Typically, the procedure is used within the convert callback of widget when it is the source of a data transfer. The procedure returns allocated memory within property_return, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XtFree`().

The standard built-in conversion routines within the Uniform Transfer Model internally call `XmRenderTableCvtToProp`() when asked to convert the _MOTIF_RENDER_TABLE selection.

See Also

`XmRenderTableCvtFromProp`(1), `XmRendition`(2).

Name

XmRenderTableFree – free the memory used by a render table.

Synopsis

void XmRenderTableFree (XmRenderTable *table*)

Inputs

table Specifies the render table to free.

Availability

Motif 2.0 and later.

Description

`XmRenderTableFree`() is a convenience function which deallocates space used by the render table table.

Usage

Render tables, and the renditions which they contain, are reference counted. It is important to call `XmRenderTableFree`() on a render table rather than `XtFree`() so that each rendition in the table is properly deallocated. Motif caches and shares render tables and the renditions which they contain, and so an improper XtFree() would not respect any sharing currently in place. `XmRenderTableFree`() does not actually free the render table until the reference count is zero.

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableCopy`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRendition`(2).

Name

XmRenderTableGetRendition – search a render table for a matching rendition.

Synopsis

XmRendition XmRenderTableGetRendition (XmRenderTable *table*, XmStringTag *tag*)

Inputs

table	Specifies the render table to search.
tag	Specifies the tag with which to find a rendition.

Returns

A Rendition which matches tag, otherwise NULL.

Availability

Motif 2.0 and later.

Description

`XmRenderTableGetRendition`() is a convenience function which searches *table*, and returns the rendition which matches *tag*.

Usage

`XmRenderTableGetRendition`() performs a linear search through the renditions contained within *table*, comparing the XmNtag resource value with the search string given by *tag*. If no match is found, any XmNnoRenditionCallback[1] callbacks registered with the XmDisplay object are invoked, supplying the table as the *render_table* element of the XmDisplayCallbackStruct passed to the callbacks. If the callbacks modify the *render_table* element, the linear search is restarted. A copy of any matching rendition is returned, otherwise NULL.

`XmRenderTableGetRendition`() allocates space for the returned rendition, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmRenditionFree`().

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableGetRenditions`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionFree`(1), `XmRendition`(2).

1.Erroneously given as XmNnoRendition in 2nd edition.

Name

XmRenderTableGetRenditions – search a render table for matching renditions.

Synopsis

```
XmRendition *XmRenderTableGetRenditions (  XmRenderTable   table,
                                           XmStringTag     *tags,
                                           Cardinal        tag_count)
```

Inputs

table	Specifies the render table to search.
tags	Specifies an array of tags for which matching renditions are required.
tag_count	Specifies the number of items in tags.

Returns

The array of renditions which have matching tags.

Availability

Motif 2.0 and later.

Description

`XmRenderTableGetRenditions`() searches *table* for all renditions which have a tag that matches an entry within the list *tags*. If the *table* is NULL, or if *tags* is NULL, or if *tag_count* is zero, the function returns NULL. Otherwise, the function returns an allocated array of matching rendition objects.

Usage

`XmRenderTableGetRenditions`() iterates through a set of *tags*, comparing in turn each tag with the group of renditions contained within a render table.

If no match is found when comparing a tag, any XmNnoRenditionCallback[1] callbacks registered with the XmDisplay object are invoked, supplying the table as the *render_table* element of the XmDisplayCallbackStruct passed to the callbacks. If the callbacks modify the *render_table* element, the linear search is restarted for that tag.

The documentation states that the function returns an allocated array, renditions being copied into the array at the same index of the matching tag within the tags array. For example, if the third tag in *tags* matches a rendition, that rendition is copied into the third element of the returned array. If any tag in the *tags* list does not match any rendition in the table, that slot in the returned array is set to NULL.

The sources, however, do not match the documentation: renditions are copied into the array in the order which they are matched, ignoring any slots which do

1.Erroneously given as XmNnoRendition in 2nd edition.

not match. Thus if the first tag in *tags* results in a NULL match, any rendition found from the second tag is placed into the first slot. If the number of matched renditions is less than the number of supplied *tags*, then memory for the returned array is reallocated to match the number of found renditions. In the absence of a XmNnoRenditionCallback callback, it is not possible to deduce the size of the returned rendition array.

The function allocates space for both the returned rendition array and the constituent renditions, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmRenditionFree`() on each of the elements in the returned array, and subsequently `XtFree`() on the array itself.

Example

The following specimen code illustrates the basic outline of a call to `XmRenderTableGetRenditions`():

```
XmRendition     *match_renditions;
XmStringTag     tags[MAX_TAGS];
int             i;

tags[0] = XmFONTLIST_DEFAULT_TAG;
tags[1] = XmS; /* "" */

...
/* search an unspecified render table */
match_renditions = XmRenderTableGetRenditions (render_table, tags,
MAX_TAGS);

/* use the matched set of renditions */
...

/* free the returned space */
if (match_renditions != NULL) {
    /* ASSUMPTION: XtNumber (match_renditions) == MAX_TAGS */
    /* Not a valid assumption if a tag does not match     */

    for (i = 0; i < MAX_TAGS; i++) {
        XmRenditionFree (match_renditions[i]);
    }

XtFree (match_renditions);
}
```

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableGetRendition`(1), `XmRenderTableRemoveRenditions`(1), `XmRenditionFree`(1), `XmRendition`(2).

Name

XmRenderTableGetTags – fetch the list of rendition tags from a render table.

Synopsis

int XmRenderTableGetTags (XmRenderTable *table*, XmStringTag **_tag_list*)

Inputs

table Specifies the render table.

Outputs

tag_list Returns the list of rendition tags.

Returns

The number of tags within the returned tag_list.

Availability

Motif 2.0 and later.

Description

`XmRenderTableGetTags`() is a convenience function which iterates through a render *table*, collecting all the tags from the individual renditions within the table, and returning them to the programmer. The number of tags placed at the address *tag_list* by the function is returned.

Usage

`XmRenderTableGetTags`() allocates an array, and places in the array a copy of the XmNtag resource for each rendition within the table. The array is returned at the address specified by the *tag_list* parameter. If the *table* is NULL, *tag_list* is initialized to NULL, and the function returns zero. It is the responsibility of the programmer to reclaim the space by calling `XtFree`() on each of the items within the allocated array, and then subsequently calling `XtFree`() on the array itself.

Example

The following specimen code illustrates the basic outline of a call to `XmRenderTableGetTags`():

```
XmStringTag *tags;
int         count, i;

/* fetch the tags from an unspecified render table */
count = XmRenderTableGetTags (render_table, &tags);

/* use the tags */
...

/* free the returned space */
```

```
if (tags != (XmStringTag *) 0) {
    for (i = 0; i < count; i++) {
        XtFree (tags[i]);
    }

    XtFree (tags);
}
```

See Also

XmRenditionFree(1), XmRendition(2).

Name

XmRenderTableRemoveRenditions – copy a render table, excluding specified renditions.

Synopsis

XmRenderTable XmRenderTableRemoveRenditions (XmRenderTable *old_table*,
XmStringTag **tags*,
int *tag_count*)

Inputs

old_table	Specifies a render table.
tags	Specifies an array of rendition tags. Any rendition which matches an item in the array is not copied from old_table.
tag_count	Specifies the number of items in the tags array.

Returns

A new render table with matching renditions removed.

Availability

Motif 2.0 and later.

Description

`XmRenderTableRemoveRenditions`() creates a new render table by copying from *old_table* only those renditions which do not have a tag matching items within the array *tags*. If *tags* is NULL, or if *tag_count* is zero, or if no renditions are removed, the function returns the *old_table* unmodified. Otherwise, *old_table* is deallocated, and the reference counts for any excluded renditions are decremented, before the function returns the newly allocated render table.

Usage

A rendition is not copied into the returned table if it has a XmNtag resource value the same as any item within the tags list. When the returned render table differs from the original *old_table* parameter, the function allocates space for the new table, and it is the responsibility of the programmer to reclaim the space by calling `XmRenderTableFree`().

See Also

`XmRenderTableAddRenditions`(1), `XmRenderTableFree`(1), `XmRendition`(2).

Name

XmRenditionCreate – create a rendition object.

Synopsis

XmRendition XmRenditionCreate (Widget *widget*, XmStringTag *tag*, Arg **arglist*, Cardinal *argcount*)

Inputs

widget	Specifies a widget.
tag	Specifies a tag for the rendition object.
arglist	Specifies an argument list, consisting of resource name/value pairs.
argcount	Specifies the number of arguments in arglist.

Returns

The new rendition object.

Availability

Motif 2.0 and later.

Description

`XmRenditionCreate`() creates a new rendition object, which can be used as an entry in a render table used for rendering XmStrings. *widget* is used to find a connection to the X server and an application context. *tag* is used as the XmNtag resource of the new rendition object. Resources for the new object are supplied in the *arglist* array.

Usage

The implementation of XmRendition is through a pseudo widget: although not a true widget, the object has resources and a resource style interface for setting and fetching values of the rendition. Typically, a rendition is merged into an existing render table through the function `XmRenderTableAddRenditions`(). Compound strings are rendered by successively matching tags within the compound string with the XmNtag resources of renditions in the table, and then using the resources of matched renditions to display the string components.

`XmRenditionCreate`() allocates storage for the returned rendition object. It is the responsibility of the programmer to reclaim the storage at a suitable point by calling `XmRenditionFree`(). Renditions are reference counted, and it is important to call `XmRenditionFree`() rather than `XtFree`() in order to maintain the references.

Example

The following specimen code creates a pair of renditions and merges them into an unspecified render table:

```
XmRendition       new_renditions[2];
```

```
XmRenderTable    new_table;
Arg              argv[4];
Cardinal         argc = 0;
Pixel            fg =...;
Pixel            bg =...;

/* create a rendition with fonts specified */
argc = 0;
XtSetArg (argv[argc], XmNfontName,        "fixed");
argc++;
XtSetArg (argv[argc], XmNfontType,        XmFONT_IS_FONT);
argc++;
XtSetArg (argv[argc], XmNloadModel,       XmLOAD_DEFERRED);
argc++;
new_renditions[0] = XmRenditionCreate (widget,
XmFONTLIST_DEFAULT_TAG, argv, argc);

/* create a rendition with line style specified */
argc = 0;
XtSetArg (argv[argc], XmNrenditionBackground,     bg);
argc++;
XtSetArg (argv[argc], XmNrenditionForeground,     fg);
argc++;
XtSetArg (argv[argc], XmNunderlineType,           XmSINGLE_LINE);
argc++;
XtSetArg (argv[argc], XmNstrikethruType,          XmSINGLE_LINE);
argc++;
new_renditions[1] = XmRenditionCreate (widget, "lineStyle", argv, argc);

/* merge into an unspecified render table */
new_table = XmRenderTableAddRenditions (old_table, new_renditions, 2,
XmMERGE_REPLACE);
```

See Also

`XmRenderTableAddRenditions`(1), `XmRenditionFree`(1), `XmRenditionRetrieve`(1), `XmRenditionUpdate`(1), `XmRendition`(2).

Name

XmRenditionFree – free the memory used by a rendition.

Synopsis

void XmRenditionFree (XmRendition *rendition*)

Inputs

rendition Specifies the rendition that is to be freed.

Availability

Motif 2.0 and later.

Description

`XmRenditionFree`() deallocates storage used by the specified *rendition*. The routine does not free any XFontSet or XFontStruct data structures associated with the rendition object.

Usage

`XmRenditionFree`() frees the storage used by the rendition object, but does not free font data structures associated with the XmNfont resource of the object. It is important to call `XmRenditionFree`() rather than `XtFree`() because Motif reference counts rendition objects. `XmRenditionFree`() decrements the reference count for the rendition; the rendition is not actually freed until the reference count reaches 0 (zero).

See Also

`XmRenditionCreate`(1), `XmRendition`(2).

Name

XmRenditionRetrieve – fetch rendition object resources.

Synopsis

void XmRenditionRetrieve (XmRendition *rendition*, Arg **arg_list*, Cardinal *arg_count*)

Inputs

rendition	Specifies the rendition whose resources are fetched.
arg_count	Specifies the number of arguments in arg_list.

Outputs

arg_list	Specifies an argument list, consisting of resource name/value pairs.

Availability

Motif 2.0 and later.

Description

`XmRenditionRetrieve`() fetches selective resource values of a *rendition* object. The set of resources retrieved is specified through the resource list *arg_list*, each element of the list being a structure containing a name/value pair. The number of elements within the list is given by *arg_count*.

Usage

`XmRenditionRetrieve`() directly returns the values of the rendition resources, and not copies of them. The programmer should not inadvertently modify a returned value, but should take a copy of any pointer-valued resource which is to be changed. For example, the XmNtag and XmNfontName resources should be copied into a separate address space before modifying or manipulating the values.

If the XmNloadModel of the rendition object is XmLOAD_DEFERRED, and the font specified by the XmNfont resource is NULL, but the XmNfontName value is not NULL, and if the programmer has specified that the font is to be retrieved within *arg_list*, then `XmRenditionRetrieve`() automatically changes the load model to XmLOAD_IMMEDIATE and directly calls a procedure to load the font indicated by XmNfontName before returning the requested resource values.

Example

The following specimen code illustrates fetching resources from an unspecified rendition object:

```
Pixel       bg;
Pixel       fg;
XtPointer   font;
String      font_name;
```

```
XmFontType      font_type;
unsigned char   load_model;
unsigned char   strike_type;
XmTabList       tab_list;
XmStringTag     tag;
unsigned char   ul_type;
Arg             av[10];
Cardinal        ac = 0;

XtSetArg (av[ac], XmNrenditionForeground,   &fg);          ac++;
XtSetArg (av[ac], XmNrenditionBackground,   &bg);          ac++;
XtSetArg (av[ac], XmNfont,                  &font);        ac++;
XtSetArg (av[ac], XmNfontName,              &font_name);   ac++;
XtSetArg (av[ac], XmNfontType,              &font_type);   ac++;
XtSetArg (av[ac], XmNloadModel,             &load_model);  ac++;
XtSetArg (av[ac], XmNstrikethruType,        &strike_type); ac++;
XtSetArg (av[ac], XmNtabList,               &tab_list);    ac++;
XtSetArg (av[ac], XmNtag,                   &tag);         ac++;
XtSetArg (av[ac], XmNunderlineType,         &ul_type);     ac++;

XmRenditionRetrieve (rendition, av, ac);
```

See Also

`XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRenditionUpdate`(1), `XmRendition`(2).

Name

XmRenditionUpdate – set rendition object resources.

Synopsis

void XmRenditionUpdate (XmRendition *rendition*, Arg **arg_list*, Cardinal *arg_count*)

Inputs

rendition	Specifies the rendition whose resources are to be changed.
arg_list	Specifies an argument list, consisting of resource name/value pairs.
arg_count	Specifies the number of arguments within arg_list.

Availability

Motif 2.0 and later.

Description

`XmRenditionUpdate`() is a convenience function which sets the resources for a *rendition* object. The attributes to change are specified through an array of name/value pairs, similar to the resource-style interface of `XtSetValues`().

Usage

Modifying the value of the XmNfontName resource initially resets the XmNfont resource to NULL, irrespective of whether the load model for the new font is XmLOAD_IMMEDIATE or XmLOAD_DEFERRED.

Example

The following specimen code illustrates setting resources for an unspecified rendition object:

```
Pixel       bg =...;
Pixel       fg =...;
Arg         av[10];
Cardinal    ac = 0;

XtSetArg (av[ac], XmNrenditionForeground,       fg);
ac++;
XtSetArg (av[ac], XmNrenditionBackground,       bg);
ac++;
XtSetArg (av[ac], XmNfontType,                  XmFONT_IS_FONT);
ac++;
XtSetArg (av[ac], XmNfontName,                  "fixed");
ac++;
XtSetArg (av[ac], XmNloadModel,                 XmLOAD_DEFERRED);
ac++;
```

```
XtSetArg (av[ac], XmNstrikethruType,        XmSINGLE_LINE);
ac++;
XtSetArg (av[ac], XmNunderlineType,         XmSINGLE_LINE);
ac++;

XmRenditionUpdate (rendition, av, ac);
```

See Also

`XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRenditionRetrieve`(1), `XmRendition`(2).

Name

XmRepTypeAddReverse – install the reverse converter for a representation type.

Synopsis

#include <Xm/RepType.h>

void XmRepTypeAddReverse (XmRepTypeId *rep_type_id*)

Inputs

rep_type_id Specifies the ID number of the representation type.

Availability

Motif 1.2 and later.

Description

`XmRepTypeAddReverse`() installs a reverse converter for a previously registered representation type. The reverse converter converts numerical representation type values to string values. The *rep_type_id* argument specifies the ID number of the representation type. If the representation type contains duplicate values, the reverse converter uses the first name in the *value_names* list that matches the specified numeric value.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeAddReverse`() provides a way for an application to install a converter that converts numeric values to their string values.

See Also

`XmRepTypeGetId`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeGetId – get the ID number of a representation type.

Synopsis

#include <Xm/RepType.h>

XmRepTypeId XmRepTypeGetId (String *rep_type*)

Inputs

rep_type Specifies the string name of a representation type.

Returns

The ID number of the representation type or XmREP_TYPE_INVALID if the representation type is not registered.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetId`() retrieves the ID number of the specified representation type *rep_type* from the representation type manager. The *rep_type* string is the string name of a representation type that has been registered with `XmRepTypeRegister`(). `XmRepTypeGetId`() returns the ID number if the representation type has been registered. This value is used in other representation type manager routines to identify a particular type. Otherwise, the routine returns XmREP_TYPE_INVALID.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetId`() provides a way for an application get the ID of a representation type, which can be used to identify the type to other representation manager routine.

See Also

`XmRepTypeGetNameList`(1), `XmRepTypeGetRecord`(1), `XmRepTypeGetRegistered`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeGetNameList – get the list of value names for a representation type.

Synopsis

#include <Xm/RepType.h>

String * XmRepTypeGetNameList (XmRepTypeId *rep_type_id*, Boolean *use_uppercase_format*)

Inputs

rep_type_id Specifies the ID number of the representation type.
use_uppercase_format Specifies whether or not the names are in uppercase characters.

Returns

A pointer to an array of value names.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetNameList`() retrieves the list of value names associated with the specified *rep_type_id*. The routine returns a pointer to a NULL-terminated list of value names for the representation type, where each value name is a NULL-terminated string. If *use_uppercase_format* is True, the value names are in uppercase characters with Xm prefixes. Otherwise, the value names are in lowercase characters without Xm prefixes. `XmRepTypeGetNameList`() allocates storage for the returned data. The application is responsible for freeing the storage using `XtFree`() on each of the elements in the returned array, and subsequently upon the array pointer itself.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetNameList`() provides a way for an application to get the named values for a particular representation type.

See Also

`XmRepTypeGetId`(1), `XmRepTypeGetRecord`(1), `XmRepTypeGetRegistered`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeGetRecord – get information about a representation type.

Synopsis

#include <Xm/RepType.h>

XmRepTypeEntry XmRepTypeGetRecord (XmRepTypeId *rep_type_id*)

Inputs

rep_type_id Specifies the ID number of the representation type.

Returns

A pointer to a representation type entry structure.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetRecord`() retrieves information about the representation type specified by *rep_type_id*. The routine returns a XmRepTypeEntry, which is a pointer to a representation type entry structure. This structure contains information about the value names and values for the enumerated type. `XmRepTypeGetRecord`() allocates storage for the returned data. The application is responsible for freeing the storage using `XtFree`().

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetRecord`() provides a way for an application to retrieve information about a particular representation type.

Structures

The XmRepTypeEntry is defined as follows:

```
typedef struct {
    String          rep_type_name;      /* name of representation type     */
    String          *value_names;       /* array of value names            */
    unsigned char   *values;            /* array of numeric values         */
    unsigned char   num_values;         /* number of values                */
    Boolean         reverse_installed;  /* reverse converter installed flag */
    XmRepTypeId     rep_type_id;        /* representation type ID          */
```

} XmRepTypeEntryRec, *XmRepTypeEntry, XmRepTypeListRec, *XmRep-TypeList;

See Also

`XmRepTypeGetId(1), XmRepTypeGetNameList(1), XmRepTypeGetRegistered(1), XmRepTypeRegister(1).`

Name

XmRepTypeGetRegistered – get the registered representation types.

Synopsis

#include <Xm/RepType.h>

XmRepTypeList XmRepTypeGetRegistered (void)

Returns
A pointer to the registration list of representation types.

Availability

Motif 1.2 and later.

Description

`XmRepTypeGetRegistered`() retrieves the whole registration list for the representation type manager. The routine returns a copy of the registration list, which contains information about all of the registered representation types. The registration list is an array of XmRepTypeList structures, where each structure contains information about the value names and values for a single representation type. The end of the registration list is indicated by a NULL pointer in the *rep_type_name* field. `XmRepTypeGetRegistered` allocates storage for the returned data. The application is responsible for freeing this storage using `XtFree`(). The list of value names (the value of the *value_names* field), the list of values (the value of the *values* field), and the array of structures all need to be freed.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeGetRegistered`() *p*rovides a way for an application to get information about all of the registered representation types.

Example

The following code fragment shows the use of `XmRepTypeGetRegistered`() to print the value names and values of all of the registered representation types:

```
XmRepTypeList replist; int i;
replist = XmRepTypeGetRegistered();
```

```
while (replist->rep_type_name != NULL) {
    printf ("Representation type name: %s\n", replist->rep_type_name);
    printf ("Value names and associated values: \n");

    for (i = 0; i < replist->num_values; i++) {
        printf ("%s: ", replist->value_names[i]);
        printf ("%d\n", replist->values[i]);
    }

    replist++;
    XtFree ((char *)replist->values);
    XtFree ((char *)replist->value_names);
}

XtFree ((char *)replist);
```

Structures

The XmRepTypeList is defined as follows:

```
typedef struct {
    String          rep_type_name;      /* name of representation type
    */
    String          *value_names;       /* array of value names        */
    unsigned char   *values;            /* array of numeric values     */
    unsigned char   num_values;         /* number of values            */
    Boolean         reverse_installed;  /* reverse converter installed flag
    */
    XmRepTypeId     rep_type_id;        /* representation type ID      */
} XmRepTypeEntryRec, *XmRepTypeEntry, XmRepTypeListRec, *XmRep-
TypeList;
```

See Also

`XmRepTypeGetRecord`(1), `XmRepTypeGetNameList`(1), `XmRepTypeRegister`(1).

Name

XmRepTypeInstallTearOffModelConverter – install the resource converter for the RowColumn XmNtearOffModel resource.

Synopsis

#include <Xm/RepType.h>

void XmRepTypeInstallTearOffModelConverter (void)

Availability

Motif 1.2 and later. In Motif 2.0 and later, the converter for the XmNtearOff-Model resource is internally installed, and this function is obsolete.

Description

`XmRepTypeInstallTearOffModelConverter`() installs the resource converter for the RowColumn XmNtearOffModel resource. This resource controls whether or not PulldownMenus and PopupMenus in an application can be torn off. Once the converter is installed, the value of XmNtearOffModel can be specified in a resource file.

Usage

In Motif 1.2, a RowColumn that is configured as a PopupMenu or a Pulldown-Menu supports tear-off menus. When a menu is torn off, it remains on the screen after a selection is made so that additional selections can be made. A menu pane that can be torn off contains a tear-off button at the top of the menu. The XmNtearOffModel resource controls whether or not tear-off functionality is available for a menu. This resource can take the values XmTEAR_OFF_ENABLED or XmTEAR_OFF_DISABLED.

In Motif 1.2, the resource converter for XmNtearOffModel is not installed by default. Some existing applications depend on receiving a callback when a menu is mapped; since torn-off menus are always mapped, these applications might fail if a user is allowed to enable tear-off menus from a resource file. `XmRepTy-peInstallTearOffModelConverter`() registers the converter that allows the resource to be set from a resource file.

See Also

`XmRowColumn`(2).

Name

XmRepTypeRegister – register a representation type resource.

Synopsis

```
#include <Xm/RepType.h>

XmRepTypeId XmRepTypeRegister (  String         rep_type,
                                 String         *value_names,
                                 unsigned char  *values,
                                 unsigned char  num_values)
```

Inputs

rep_type	Specifies the string name for the representation type.
value_names	Specifies an array of value names for the representation type. IP values 1i Specifies an array of values for the representation type.
num_values	Specifies the number of items in value_names and values.

Returns

The ID number of the representation type.

Availability

Motif 1.2 and later.

Description

`XmRepTypeRegister`() registers a representation type with the representation type manager. The representation type manager provides resource conversion facilities for enumerated values. `XmRepTypeRegister`() installs a resource converter that converts string values to numerical representation type values. The strings in the *value_names* array specify the value names for the representation type. The strings are specified in lowercase characters, with underscore characters separating words and without Xm prefixes.

If the *values* argument is NULL, the order of the strings in the *value_names* array determines the numerical values for the enumerated type. In this case, the names are assigned consecutive values starting with 0 (zero). If *values* is non-NULL, it is used to assign values to the names. Each name in the *value_names* array is assigned the corresponding value in the *values* array, so it is possible to have non-consecutive values or duplicate names for the same value.

`XmRepTypeRegister`() returns the ID number that is assigned to the representation type. This value is used in other representation type manager routines to identify a particular type. A representation type can only be registered once. If a type is registered more than once, the behavior of the representation type manager is undefined.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeRegister`() provides a way for an application to register representation types for application-specific resources or for new widget classes.

See Also

`XmRepTypeAddReverse`(1), `XmRepTypeGetId`(1), `XmRepTypeGetNameList`(1), `XmRepTypeGetRecord`(1), `XmRepTypeGetRegistered`(1), `XmRepTypeValidValue`(1).

Name

XmRepTypeValidValue – determine the validity of a numerical value for a representation type.

Synopsis

```
#include <Xm/RepType.h>

Boolean XmRepTypeValidValue (  XmRepTypeId   rep_type_id,
                               unsigned char test_value,
                               Widget        enable_default_warning)
```

Inputs

rep_type_id	Specifies the ID number of the representation type.
test_value	Specifies the value that is to be tested.
enable_default_warning	Specifies a widget that is used to generate a default warning message.

Returns

True if the specified value is valid or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmRepTypeValidValue`() checks the validity of the specified *test_value* for the representation type specified by *rep_type_id*. The routine returns True if the value is valid. Otherwise, it returns False. If the *enable_default_warning* parameter is non-NULL, `XmRepTypeValidValue`() uses the specified widget to generate a default warning message if the value is invalid. If *enable_default_warning* is NULL, no default warning message is provided.

Usage

In Motif 1.2 and later, the representation type manager provides support for handling many of the tasks related to enumerated values. This facility installs resource converters that convert a string value to its numerical representation. The representation type manager can also be queried to get information about the registered types. This facility is especially useful for interface builders and applications like *editres* that allow a user to set resources interactively. `XmRepTypeValidValue`() provides a way for an application to check if a value is valid for a particular representation type.

See Also

`XmRepTypeGetId`(1), `XmRepTypeRegister`(1).

Name

XmResolveAllPartOffsets – ensure upward-compatible widgets and applications.

Synopsis

```
void XmResolveAllPartOffsets (  WidgetClass     widget_class,
                                XmOffsetPtr     *offset,
                                XmOffsetPtr     *constraint_offset)
```

Inputs

widget_class Specifies the widget class pointer.

Outputs

offset Returns the widget offset record.

constraint_offset Returns the constraint offset record.

Description

`XmResolveAllPartOffsets`() ensures that an application or a widget will be upwardly compatible with the records in a widget structure. In other words, if the size of a widget structure changes in the future, this routine can be used to calculate the locations of the new offsets. This routine and `XmResolvePartOffsets`() are similar. During the creation of a widget, both routines modify the widget structure by allocating an array of offset values. XmResolvePartOffsets() affects only the widget instance record, while `XmResolveAllPartOffsets`() affects the widget instance and constraint records.

Usage

If you are subclassing a Motif widget, you should use `XmResolveAllPartOffsets`() and `XmResolvePartOffsets`() to ensure that your widget will be compatible with future releases of the toolkit.

See Also

`XmResolvePartOffsets`(1).

Name

XmResolvePartOffsets – ensure upward-compatible widgets and applications.

Synopsis

void XmResolvePartOffsets (WidgetClass *widget_class*, XmOffsetPtr **offset*)

Inputs

widget_class Specifies the widget class pointer.

Outputs

offset Returns the widget offset record.

Description

`XmResolvePartOffsets`() ensures that an application or a widget will be upwardly compatible with the records in a widget structure. In other words, if the size of a widget structure changes in the future, this routine can be used to calculate the locations of the new offsets. This routine and `XmResolveAllPartOffsets`() are similar. During the creation of a widget, both routines modify the widget structure by allocating an array of offset values. `XmResolvePartOffsets`() affects only the widget instance record, while `XmResolveAllPartOffsets`() affects the widget instance and constraint records.

Usage

If you are subclassing a Motif widget, you should use `XmResolvePartOffsets`() and `XmResolveAllPartOffsets`() to ensure that your widget will be compatible with future releases of the toolkit.

See Also

`XmResolveAllPartOffsets`(1).

Name

XmScaleGetValue – get the slider value for a Scale widget.

Synopsis

#include <Xm/Scale.h>

void XmScaleGetValue (Widget *widget*, int **value_return*)

Inputs

widget Specifies the Scale widget.

Outputs

value_return Returns the current slider position for the Scale.

Description

`XmScaleGetValue`() returns the current position of the slider within the specified Scale *widget*.

Usage

`XmScaleGetValue`() is a convenience routine that returns the value of the XmNvalue resource for the Scale widget. Calling the routine is equivalent to calling `XtGetValues`() for that resource, although `XmScaleGetValue`() accesses the value through the widget instance structure rather than through `XtGetValues`().

See Also

`XmScaleSetValue`(1), `XmScale`(2).

Name

XmScaleSetTicks – set tick marks for a Scale widget.

Synopsis

```
#include <Xm/Scale.h>

void XmScaleSetTicks ( Widget      widget,
                       int         big_every,
                       Cardinal    num_med,
                       Cardinal    num_small,
                       Dimension   size_big,
                       Dimension   size_med,
                       Dimension   size_small)
```

Inputs

widget	Specifies a scale widget.
big_every	Specifies the number of scale values between large ticks.
num_med	Specifies the number of medium-sized ticks between the large tick marks.
num_small	Specifies the number of small-sized ticks between the medium-sized tick marks.
size_big	Specifies the size of the large ticks.
size_med	Specifies the size of the medium ticks.
size_small	Specifies the size of the small ticks.

Availability

Motif 2.0 and later.

Description

XmScaleSetTicks() places tick marks along the edges of a Scale widget. Ticks may be of three types: big, medium, and small, and the size (in pixels) of each type is specified by size_big, size_med, and size_small respectively. The location of each big tick is given by big_every, which simply specifies the number of scale values between each big tick. The number of medium-sized ticks between each big tick is given by num_med, and the number of small-sized ticks between each medium-sized tick is num_small.

Usage

XmScaleSetTicks() is a convenience function which places tick marks along the edge of a Scale by creating a series of SeparatorGadget children at evenly spaced intervals. If size_big is zero, XmScaleSetTicks() simply returns. If size_med or size_small is zero, num_med and num_small are forced to zero respectively. The number of medium and small tick marks required may be zero, but the number of large tick marks must not be less than 2.

SeparatorGadgets are created with the names "BigTic", "MedTic", and "SmallTic", the XmNseparatorType resource of each is forced to XmSINGLE_LINE. XmScaleSetTicks() does not delete any existing ticks when invoked on any particular Scale, neither does the Scale recalculate proper positions for the tick marks if the scale orientation is changed after tick marks are added. In each case, existing tick marks must be erased and subsequently redrawn or re-specified.

Example

The following code ensures that any tick marks are erased before adding new ticks to a Scale:

```
#include <Xm/Scale.h>
#include <Xm/SeparatoG.h>

void ScaleEraseSetTicks ( Widget      scale,
                          int         big_every,
                          Cardinal    num_med,
                          Cardinal    num_small,
                          Dimension   size_big,
                          Dimension   size_med,
                          Dimension   size_med)
{
    WidgetList    children     = (WidgetList) 0;
    Cardinal      num_children = (Cardinal) 0;
    int           i;
    String        name;

    /* fetch scale children. */
    XtVaGetValues (scale, XmNchildren, &children, XmNnumChildren,
    &num_children, 0)

    /* destroy old ticks.                */
    /* some optimization to reuse correctly   */
    /* placed ticks might be in order here... */
    for (i = 0; i < num_children; i++) {
        if (XmIsSeparatorGadget (children[i])) {
            if ((name = XtName (children[i])) != (String) 0) {
                if ((strcmp (name, "BigTic") == 0) ||
                    (strcmp (name, "MedTic") == 0) ||
                    (strcmp (name, "SmallTic") == 0)) {
                    XtDestroyWidget (children[i]);}
            }
        }
    }
```

```
    /* create new ticks. */
    XmScaleSetTicks (scale, big_every, num_med, num_small, size_big,
    size_med, size_small);
}
```

See Also

`XmScaleSetValues`(1). `XmScale`(2), `XmSeparatorGadget`(2).

Name

XmScaleSetValue – set the slider value for a Scale widget.

Synopsis

#include <Xm/Scale.h>

void XmScaleSetValue (Widget *widget*, int *value*)

Inputs

widget	Specifies the Scale widget.
value	Specifies the value of the slider.

Description

`XmScaleSetValue`() sets the current position of the slider to value in the specified Scale *widget*. The *value* must be in the range XmNminimum to XmNmaximum.

Usage

`XmScaleSetValue`() is a convenience routine that sets the value of the XmNvalue resource for the Scale widget. Calling the routine is equivalent to calling `XtSetValues`() for that resource, although `XmScaleSetValue`() accesses the value through the widget instance structure rather than through `XtSetValues`().

See Also

`XmScaleGetValue`(1), `XmScale`(2).

Name

XmScrollBarGetValues – get information about the current state of a ScrollBar widget.

Synopsis

```
#include <Xm/ScrollBar.h>

void XmScrollBarGetValues (  Widget   widget,
                             int      *value_return,
                             int      *slider_size_return,
                             int      *increment_return,
                             int      *page_increment_return)
```

Inputs

widget	Specifies the ScrollBar widget.

Outputs

value_return	Returns the current slider position.
slider_size_return	Returns the current size of the slider.
increment_return	Returns the current increment and decrement level.
page_increment_return	Returns the current page increment and decrement level.

Description

`XmScrollBarGetValues`() returns the current state information for the specified ScrollBar *widget*. This information consists of the position and size of the slider, as well as the increment and page increment values.

Usage

`XmScrollBarGetValues`() is a convenience routine that returns the values of the XmNvalue, XmNsliderSize, XmNincrement, and XmNpageIncrement resources for the ScrollBar widget. Calling the routine is equivalent to calling `XtGetValues`() for those resources, although `XmScrollBarGetValues`() accesses the values through the widget instance structure rather than through `XtGetValues`().

See Also

`XmScrollBarSetValues`(1), `XmScrollBar`(2).

Name

XmScrollBarSetValues – set the current state of a ScrollBar widget.

Synopsis

```
#include <Xm/ScrollBar.h>

void XmScrollBarSetValues (  Widget     widget,
                             int        value,
                             int        slider_size,
                             int        increment,
                             int        page_increment,
                             Boolean    notify)
```

Inputs

widget	Specifies the ScrollBar widget.
value	Specifies the slider position.
slider_size	Specifies the size of the slider.
increment	Specifies the increment and decrement level.
page_increment	Specifies the page increment and decrement level.
notify	Specifies whether or not the value changed callback is invoked.

Description

`XmScrollBarSetValues`() sets the current state of the specified ScrollBar *widget*. The position of the slider is set to *value*, which must be in the range XmNminimum to XmNmaximum minus XmNsliderSize. The size of the slider is set to *slider_size*, which must be between 1 and the size of the scroll region. The increment and page increment values are set to *increment* and *page_increment*, respectively.

If *notify* is True, `XmScrollBarSetValues`() invokes the XmNvalueChangedCallback for the ScrollBar when the state is set.

Usage

`XmScrollBarSetValues`() is a convenience routine that sets the values of the XmNvalue, XmNsliderSize, XmNincrement, and XmNpageIncrement resources for the ScrollBar widget. Calling the routine is equivalent to calling `XtSetValues`() for those resources, although `XmScrollBarSetValues`() accesses the values through the widget instance structure rather than through `XtSetValues`().

The *notify* parameter indicates whether or not the value changed callbacks for the ScrollBar are invoked. You can avoid redundant code by setting this parameter to True. If you are calling `XmScrollBarSetValues`() from a value changed

callback routine, you probably want to set the parameter to False to avoid the possibility of an infinite loop. Calling `XmScrollBarSetValues`() with *notify* set to True causes the callback routines to be invoked in a way that is indistinguishable from a user-initiated adjustment to the ScrollBar.

See Also

`XmScrollBarGetValues`(1), `XmScrollBar`(2).

Name

XmScrolledWindowSetAreas – specify the children for a scrolled window.

Synopsis

```
#include <Xm/ScrolledW.h>

void XmScrolledWindowSetAreas (Widget   widget,
                               Widget   horizontal_scrollbar,
                               Widget   vertical_scrollbar,
                               Widget   work_region)
```

Inputs

widget	Specifies the ScrolledWindow widget.
horizontal_scrollbar	Specifies the widget ID of the horizontal ScrollBar.
vertical_scrollbar	Specifies the widget ID of the vertical ScrollBar.
work_region	Specifies the widget ID of the work window.

Availability

In Motif 2.0 and later, `XmScrolledWindowSetAreas`() is obsolete.

Description

`XmScrolledWindowSetAreas`() sets up the standard regions of a ScrolledWindow widget for an application. The ScrolledWindow must be created before the routine is called. `XmScrolledWindowSetAreas`() specifies the horizontal and vertical ScrollBars and the work window region. If a particular ScrolledWindow does not have one of these regions, the corresponding argument can be specified as NULL.

Usage

Each of the ScrolledWindow regions is associated with a ScrolledWindow resource; `XmScrolledWindowSetAreas`() sets the associated resources. The resources that correspond to the last three arguments to the routine are XmNhorizontalScrollBar, XmNverticalScrollBar, and XmNworkWindow, respectively.

If an application does not call `XmScrolledWindowSetAreas`(), the widget may still set some of the standard regions. If ScrollBars are added as children, the XmNhorizontalScrollBar and XmNverticalScrollBar resources may be set if they have not already been specified. Any child that is not a ScrollBar is used for the XmNworkWindow. If you want to be certain about which widgets are used for the different regions, it is wise to call `XmScrolledWindowSetAreas`() explicitly.

In Motif 2.0 and later, the function is obsolete, and the programmer should specify the XmNhorizontalScrollBar, XmNverticalScrollBar, and XmNworkWindow

resources directly through a call to XtSetValues(). Although ostensibly maintained for backwards compatibility, the implementation of XmScrolledWindowSetAreas() in Motif 2.0 and later is not Motif 1.2 compatible. In Motif 1.2, supplying a NULL value for any of the scrollbar or work window parameters directly sets the internal component to NULL. In Motif 2.0 and later, supplying a NULL value causes that parameter to be ignored, leaving the internal component intact.

Example

The following code fragment shows how to set the regions of a ScrolledWindow:

```
Widget    toplevel, scrolled_w, drawing_a, vsb, hsb;
int       view_width, view_height;

scrolled_w = XtVaCreateManagedWidget ("scrolled_w", xmScrolledWindow-
WidgetClass, toplevel,
                                        XmNscrollingPolicy,
                                        XmAPPLICATION_DEFINED,
                                        XmNvisualPolicy, XmVARIABLE,
                                        NULL);
drawing_a = XtVaCreateManagedWidget ("drawing_a", xmDrawingAreaWidg-
etClass, scrolled_w,
                                        XmNwidth, view_width,
                                        XmNheight, view_height,
                                        NULL);
vsb = XtVaCreateManagedWidget ("vsb", xmScrollBarWidgetClass, scrolled_w,
                                    XmNorientation, XmVERTICAL,
                                    NULL);
hsb = XtVaCreateManagedWidget ("hsb", xmScrollBarWidgetClass, scrolled_w,
                                    XmNorientation, XmHORIZONTAL,
                                    NULL);

XmScrolledWindowSetAreas (scrolled_w, hsb, vsb, drawing_a);
```

See Also

XmScrolledWindow(2).

Name

XmScrollVisible – make an obscured child of a ScrolledWindow visible.

Synopsis

```
#include <Xm/ScrolledW.h>

void XmScrollVisible (  Widget      scrollw_widget,
                        Widget      widget,
                        Dimension   left_right_margin,
                        Dimension   top_bottom_margin)
```

Inputs

scrollw_widget	Specifies the ScrolledWindow widget.
widget	Specifies the widget ID of the widget that is to be made visible.
left_right_margin	Specifies the distance between the widget and the left or right edge of the viewport if the ScrolledWindow is scrolled horizontally.
top_bottom_margin	Specifies the distance between the widget and the top or bottom edge of the viewport if the ScrolledWindow is scrolled vertically.

Availability

Motif 1.2 and later.

Description

`XmScrollVisible`() scrolls the specified ScrolledWindow *scrollw_widget* so that the obscured or partially obscured *widget* becomes visible in the work area viewport. *widget* must be a descendent of *scrollw_widget*. The routine repositions the work area of the ScrolledWindow and sets the margins between the widget and the viewport boundaries based on *left_right_margin* and *top_bottom_margin* if necessary.

Usage

`XmScrollVisible`() provides a way for an application to ensure that a particular child of a ScrolledWindow is visible. In order for the routine to work, the XmNscrollingPolicy of the ScrolledWindow widget must be set to XmAUTOMATIC. This routine is designed to be used in the XmNtraverseObscureCallback for a ScrolledWindow.

See Also

`XmScrolledWindow`(2).

Name

XmSelectionBoxGetChild – get the specified child of a SelectionBox widget.

Synopsis

#include <Xm/SelectioB.h>

Widget XmSelectionBoxGetChild (Widget *widget*, unsigned char *child*)

Inputs

widget Specifies the SelectionBox widget.
child Specifies the child of the SelectionBox widget. Pass one of the values from the list below.

Returns

The widget ID of the specified child of the SelectionBox.

Availability

As of Motif 2.0, the toolkit abstract child fetch routines are marked for deprecation. You should give preference to `XtNameToWidget`(), except when fetching the SelectionBox default button or work area.

Description

`XmSelectionBoxGetChild`() returns the widget ID of the specified child of the SelectionBox widget.

Usage

XmDIALOG_APPLY_BUTTON, XmDIALOG_CANCEL_BUTTON, XmDIALOG_HELP_BUTTON, and XmDIALOG_OK_BUTTON specify the action buttons in the *widget*. XmDIALOG_DEFAULT_BUTTON specifies the current default button. XmDIALOG_LIST and XmDIALOG_LIST_LABEL specify the list and its label. XmDIALOG_TEXT and XmDIALOG_SELECTION_LABEL specify the selection text entry area and its label. XmDIALOG_SEPARATOR specifies the separator and XmDIALOG_WORK_AREA specifies any work area child that has been added to the SelectionBox. For more information on the different children of the SelectionBox, see the manual page in Section 2, *Motif and Xt Widget Classes*.

Widget Hierarchy

As of Motif 2.0, most Motif composite child fetch routines are marked as deprecated. However, since it is not possible to fetch the XmDIALOG_DEFAULT_BUTTON or XmDIALOG_WORK_AREA children using a public interface except through XmSelectionBoxGetChild(), the routine should not be considered truly deprecated. For consistency with the preferred new style, when fetching all other child values, consider giving preference to the

Intrinsics routine XtNameToWidget(), passing one of the following names as the second parameter:

“Apply”	(XmDIALOG_APPLY_BUTTON)
“Cancel”	(XmDIALOG_CANCEL_BUTTON)
“OK”	(XmDIALOG_OK_BUTTON)
“Separator”	(XmDIALOG_SEPARATOR)
“Help”	(XmDIALOG_HELP_BUTTON)
“Symbol”	(XmDIALOG_SYMBOL_LABEL)
“Message”	(XmDIALOG_MESSAGE_LABEL)
“*ItemsList”[1]	(XmDIALOG_LIST)
“Items”	(XmDIALOG_LIST_LABEL)
“Selection”	(XmDIALOG_SELECTION_LABEL)
“Text”	(XmDIALOG_TEXT)

Structures

The possible values for child are:

XmDIALOG_APPLY_BUTTON
XmDIALOG_OK_BUTTON
XmDIALOG_CANCEL_BUTTON
XmDIALOG_SELECTION_LABEL
XmDIALOG_DEFAULT_BUTTON XmDIALOG_SEPARATOR
XmDIALOG_HELP_BUTTON XmDIALOG_TEXT
XmDIALOG_LIST
XmDIALOG_WORK_AREA
XmDIALOG_LIST_LABEL

See Also

`XmPromptDialog`(2), `XmSelectionBox`(2).

1.The “*” is important: the List is not a direct child of the SelectionBox, but of a ScrolledList.

Name

XmSetColorCalculation – set the procedure that calculates default colors.

Synopsis

XmColorProc XmSetColorCalculation (XmColorProc *color_proc*)

Inputs

color_proc Specifies the procedure that is used for color calculation.

Returns

The previous color calculation procedure.

Description

`XmSetColorCalculation`() sets the procedure called by `XmGetColors`()[1] that calculates the default foreground, top and bottom shadow, and selection colors. The procedure calculates these colors based on the background color that has been passed to the procedure. If *color_proc* is NULL, this routine restores the default color calculation procedure. `XmSetColorCalculation`() returns the color calculation procedure that was in use when the routine was called. Both `XmGetColors`() and `XmChangeColor`() use the color calculation procedure.

Usage

Motif widgets rely on the use of shadowed borders to create their three-dimensional appearance. The top and bottom shadow colors are lighter and darker shades of the background color; these colors are reversed to make a component appear raised out of the screen or recessed into the screen. The select color is a slightly darker shade of the background color that indicates that a component is selected. The foreground color is either black or white, depending on which color provides the most contrast with the background color. `XmSetColorCalculation`() sets the procedure that calculates these colors. Use `XmGetColorCalculation`() to get the default color calculation procedure.

In Motif 2.0 and later, per-screen color calculation procedures are supported: if the XmNcolorCalculationProc resource of the XmScreen object associated with a given widget is not NULL, the procedure specified by the resource is used to calculate color in preference to any procedure which may have been specified by `XmSetColorCalculation`().

Procedures

The XmColorProc has the following syntax:

typedef void (*XmColorProc) (XColor *bg_color, /* specifies the background color */

1.Erroneously missing from 1st and 2nd editions.

```
                                XColor      *fg_color,    /* returns the fore-
ground color */
                                XColor      *sel_color,   /* returns the select
color */
                                XColor      *ts_color,    /* returns the top
shadow color */
                                XColor      *bs_color)    /* returns the bottom
shadow color */
```

An XmColorProc takes five arguments. The first argument, bg_color, is a pointer to an XColor structure that specifies the background color. The red, green, blue, and pixel fields in the structure contain valid values. The rest of the arguments are pointers to XColor structures for the colors that are to be calculated. The procedure fills in the red, green, and blue fields in these structures.

See Also

`XmChangeColor`(1), `XmGetColorCalculation`(1), `XmGetColors`(1). `XmScreen`(2).

Name

XmSetFontUnit – set the font unit values.

Synopsis

void XmSetFontUnit (Display **display*, int *font_unit_value*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

font_unit_value Specifies the value for both horizontal and vertical font units.

Availability

In Motif 1.2 and later, `XmSetFontUnit`() is obsolete. It has been superseded by setting the Screen resources XmNhorizontalFontUnit and XmNverticalFontUnit.

Description

`XmSetFontUnit`() sets the value of the horizontal and vertical font units for all of the screens on the display. This routine is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

Font units are a resolution-independent unit of measurement that are based on the width and height characteristics of a particular font. The default horizontal and vertical font unit values are based on the XmNfont resource, which in Motif 1.2, is a resource of the Screen object. An application can override these default values by calling `XmSetFontUnit`(). The values should be set before any widgets that use resolution-independent data are created.

See Also

`XmConvertUnits`(1), `XmSetFontUnits`(1), `XmGadget`(2), `XmManager`(2), `XmPrimitive`(2), `XmScreen`(2).

Name

XmSetFontUnits – set the font unit values.

Synopsis

void XmSetFontUnits (Display **display*, int *h_value*, int *v_value*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

h_value Specifies the value for horizontal font units.

v_value Specifies the value for vertical font units.

Availability

In Motif 1.2 and later, `XmSetFontUnits`() is obsolete. It has been superseded by setting the Screen resources XmNhorizontalFontUnit and XmNverticalFontUnit.

Description

`XmSetFontUnits`() sets the value of the horizontal and vertical font units to *h_value* and *v_value* respectively. The routine sets the font units for all of the screens on the display. This routine is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

Font units are a resolution-independent unit of measurement that are based on the width and height characteristics of a particular font. The default horizontal and vertical font unit values are based on the XmNfont resource, which in Motif 1.2 and later, is a resource of the Screen object. An application can override these default values by calling `XmSetFontUnits`(). The values should be set before any widgets that use resolution-independent data are created.

See Also

`XmConvertUnits`(1), `XmSetFontUnit`(1), `XmGadget`(2), `XmManager`(2), `XmPrimitive`(2), `XmScreen`(2).

Name

XmSetMenuCursor – set the current menu cursor.

Synopsis

void XmSetMenuCursor (Display **display*, Cursor *cursorId*)

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

cursorId Specifies the cursor ID for the menu cursor.

Availability

In Motif 1.2 and later, `XmSetMenuCursor`() is obsolete. It has been superseded by setting the Screen resource XmNmenuCursor.

Description

`XmSetMenuCursor`() sets the menu cursor for an application. The routine sets the cursor for all screens on the specified *display.* The specified cursor is shown whenever the application is using a Motif menu on the specified *display.* This routine is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

The menu cursor is the pointer shape that is used whenever a menu is posted. This cursor can be different from the normal pointer shape. In Motif 1.2 and later, the new Screen object has a resource, XmNmenuCursor, that specifies the menu cursor. `XmSetMenuCursor`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

See Also

`XmGetMenuCursor`(1), `XmScreen`(2).

Name

XmSetProtocolHooks – set prehooks and posthooks for a protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmSetProtocolHooks ( Widget          shell,
                          Atom            property,
                          Atom            protocol,
                          XtCallbackProc  prehook,
                          XtPointer       pre_closure,
                          XtCallbackProc  posthook,
                          XtPointer       post_closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
property	Specifies the property that holds the protocol data.
protocol	Specifies the protocol atom.
prehook	Specifies the procedure to invoke before the client callbacks.
pre_closure	Specifies any client data that is passed to the prehook.
posthook	Specifies the procedure to invoke after the client callbacks.
post_closure	Specifies any client data that is passed to the posthook.

Description

`XmSetProtocolHooks`() allows pre- and post-procedures to be invoked in addition to the regular callback procedures that are performed when the Motif window manager sends a protocol message. The prehook procedure is invoked before calling the procedures on the client's callback list, whereas the posthook procedure is invoked after calling the procedures on the client's callback list. This routine gives shells more control flow, since callback procedures aren't necessarily executed in any particular order.

Usage

A protocol is a communication channel between applications. Protocols are simply atoms, stored in a property on the top-level shell window for the application. To communicate using a protocol, a client sends a ClientMessage event containing a *property* and *protocol*, and the receiving client responds by calling the associated protocol callback routine. `XmSetProtocolHooks`() gives an application more control over the flow of callback procedures, since callbacks are not necessarily invoked in any particular order.

See Also

`XmAddProtocolCallback`(1), `XmRemoveProtocolCallback`(1), `XmSetWMProtocolHooks`(1), `VendorShell`(2).

Name

XmSetWMProtocolHooks – set prehooks and posthooks for the XA_WM_PROTOCOLS protocol.

Synopsis

```
#include <Xm/Protocols.h>

void XmSetWMProtocolHooks ( Widget          shell,
                            Atom            protocol,
                            XtCallbackProc  prehook,
                            XtPointer       pre_closure,
                            XtCallbackProc  posthook,
                            XtPointer       post_closure)
```

Inputs

shell	Specifies the widget associated with the protocol property.
protocol	Specifies the protocol atom.
prehook	Specifies the procedure to invoke before the client callbacks.
pre_closure	Specifies any client data that is passed to the prehook.
posthook	Specifies the procedure to invoke after the client callbacks.
post_closure	Specifies any client data that is passed to the posthook.

Description

`XmSetWMProtocolHooks`()[1] is a convenience routine that calls `XmSetProtocolHooks`() with property set to XA_WM_PROTOCOL, the window manager protocol property.

Usage

The property XA_WM_PROTOCOLS is a set of predefined protocols for communication between clients and window managers. To communicate using a protocol, a client sends a ClientMessage event containing a property and *protocol*, and the receiving client responds by calling the associated protocol callback routine. `XmSetWMProtocolHooks`() gives an application more control over the flow of callback procedures, since callbacks are not necessarily invoked in any particular order.

See Also

`XmAddWMProtocolCallback`(1), `XmInternAtom`(1), `XmRemoveWMProtocolCallback`(1), `XmSetProtocolHooks`(1), `VendorShell`(2).

1.Erroneously given as XmSetXmProtocolHooks() in 1st and 2nd editions.

Name

XmSimpleSpinBoxAddItem – add an item to a SimpleSpinBox.

Synopsis

#include <Xm/SSpinB.h>

void XmSimpleSpinBoxAddItem (Widget *widget*, XmString *item*, int *position*)

Inputs

widget	Specifies a SimpleSpinBox widget.
item	Specifies an item to add.
position	Specifies the position at which to add the new item.

Availability

Motif 2.1 and later.

Description

`XmSimpleSpinBoxAddItem`() adds an *item* to a SimpleSpinBox *widget* at a given *position* within the list of values which the widget may display. If *position* is zero, or if *position* is greater than the number of items in the list, the *item* is appended to the list of values.

Usage

`XmSimpleSpinBoxAddItem`() is a convenience routine that adds an *item* to the list of items which a SimpleSpinBox may display. In order to add an item to the SimpleSpinBox, a compound string must be created. `XmSimpleSpinBox-AddItem`() adds the *item* to the SimpleSpinBox by manipulating the XmNvalues, XmNnumValues, and XmNposition resources of the widget. If the XmNspinBoxChildType resource of the widget is not XmSTRING, or if the item is NULL, the procedure simply returns without modifying the array of values.

The SimpleSpinBox widget takes a copy of the supplied item; the programmer is responsible for freeing the compound string at an appropriate point by calling `XmStringFree`().

Example

The following procedure simply appends an item onto the end of a SimpleSpinBox list:

```
void SimpleSpinBoxAppend (Widget spinb, char *item)
{
   XmString xms = XmStringGenerate ((XtPointer)
   value,
                                    XmFONTLIST_DEF
                                    AULT_TAG,
```

```
                                                XmCHARSET_TEXT,
                                                NULL);
        XmSimpleSpinBoxAddItem (spinb, xms, 0);
        XmStringFree (xms);
    }
```

See Also

XmSimpleSpinBoxDeletePos(1), XmSimpleSpinBoxSetItem(1), XmStringFree(1), XmSimpleSpinBox(2).

Name

XmSimpleSpinBoxDeletePos – delete an item at the specified position from a SimpleSpinBox.

Synopsis

#include <Xm/SSpinB.h>

void XmSimpleSpinBoxDeletePos (Widget *widget*, int *position*)

Inputs

widget	Specifies a SimpleSpinBox widget.
position	Specifies the position at which to delete an item.

Availability

Motif 2.1 and later.

Description

`XmSimpleSpinBoxDeletePos`() deletes an item at a given *position* from a SimpleSpinBox widget. A value of 1 indicates the first item, 2 is the second item, and so on. The last item in the list can be specified by passing a *position* of zero.

Usage

`XmSimpleSpinBoxDeletePos`() is a convenience function which deletes an item from the set of values associated with a SimpleSpinBox. The function directly manipulates the XmNvalues, XmNnumValues, and XmNposition resources of the widget. If the XmNspinBoxChildType resource of the widget is not XmSTRING, the function simply returns without modifying the array of values.

See Also

`XmSimpleSpinBoxAddItem`(1), `XmSimpleSpinBoxSetItem`(1), `XmSimpleSpinBox`(2).

Name

XmSimpleSpinBoxSetItem – set an item in a SimpleSpinBox.

Synopsis

#include <Xm/SSpinB.h>

void XmSimpleSpinBoxSetItem (Widget *widget*, XmString *item*)

Inputs

widget	Specifies a SimpleSpinBox widget.
item	Specifies the item to set.

Availability

Motif 2.1 and later.

Description

`XmSimpleSpinBoxSetItem`() makes an item in a SimpleSpinBox widget the current value.

Usage

`XmSimpleSpinBoxSetItem`() is a convenience routine that selects one of the SimpleSpinBox values. The *item* must exist within the XmNvalues array of the *widget*, otherwise a warning message is displayed. The function modifies the XmNposition resource of the widget if the item is found. No check is performed to ensure that the XmNspinBoxChildType resource of the SimpleSpinBox is XmSTRING.

See Also

`XmSimpleSpinBoxAddItem`(1), `XmSimpleSpinBoxDeletePos`(1), `XmSimpleSpinBox`(2).

Name

XmSpinBoxValidatePosition – validate the current value of a SpinBox.

Synopsis

#include <Xm/SpinB.h>

int XmSpinBoxValidatePosition (Widget *text_field*, int **position_value*)

Inputs

text_field Specifies a text field child of a SpinBox widget.

Outputs

position_value Returns the position of the current value.

Returns

The status of the validation.

Availability

Motif 2.1 and later.

Description

`XmSpinBoxValidatePosition`() checks that the *text_field* child of a SpinBox has a valid position value, and places the validated value of the text_field at the address *position_value*. If the position is valid, the function returns XmVALID_VALUE. Otherwise the function returns XmCURRENT_VALUE, XmMAXIMUM_VALUE, XmMINIMUM_VALUE, or XmINCREMENT_VALUE, depending upon a comparison of the current position and other constraint resources of the *text_field*.

Usage

`XmSpinBoxValidatePosition`() can be used to ensure that the user has entered a valid value into an editable textual child of a SpinBox. If *text_field* is NULL, or if *text_field* does not hold the XmQTaccessTextual trait, or if the XmNspinBoxChildType of this widget is not XmNUMERIC the function returns XmCURRENT_VALUE. The current value of the text field is fetched as a floating point number, then converted into an integer using the XmNdecimalPoints resource: digits after the decimal place are simply truncated. The current value is subsequently compared against the XmNminimumValue and XmNmaximumValue resources: if less than XmNminimumValue, *position_value* is set to the value of XmNminimumValue, and the function returns XmMINIMUM_VALUE, or if the current value is more than XmNmaximumValue, *position_value* is set to the value of XmNmaximumValue, and the function returns XmMAXIMUM_VALUE. Lastly, the function checks that the current value falls between XmNminimumValue and XmNmaximumValue on an interval specified

by the XmNincrementValue resource. That is, the current value is a member of the set:

```
{
        XmNminimumValue,
        XmNminimumValue + XmNincrementValue,
        XmNminimumValue + (2 * XmNincrementValue),
        XmNminimumValue + (3 * XmNincrementValue),
        ...
        XmNminimumValue + (n * XmNincrementValue),
        ...
        XmNmaximumValue
}
```

If the current value does not fall within the set, the *position_value* is set to the nearest item in the set which is not more than the current value, and the function returns XmINCREMENT_VALUE. If all checks pass, the *position_value* is set to the current value, and the function returns XmVALID_VALUE.

The SpinBox does not modify the contents of *text_field* when performing the validation.

Structures

The returned status has the following values:

XmCURRENT_VALUE	XmINCREMENT_VALUE
XmMAXIMUM_VALUE	XmMINIMUM_VALUE
XmVALID_VALUE	

See Also

`XmSpinBox`(2).

Name

XmStringBaseline – get the baseline spacing for a compound string.

Synopsis

Dimension XmStringBaseline (XmFontList *fontlist*, XmString *string*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Returns

The distance, in pixels, from the top of the character box to the baseline of the first line of text.

Availability

In Motif 2.0 and later, the XmFontList is obsolete. It is superseded by the XmRenderTable, to which it has become an alias.

Description

`XmStringBaseline`() returns the distance, in pixels, from the top of the character box to the baseline of the first line of text in *string*. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringBaseline`() provides information that is useful if you need to render a compound string. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringComponentCreate`(1), `XmStringExtent`(1), `XmStringHeight`(1), `XmStringWidth`(1), `XmRendition`(2).

Name

XmStringByteCompare – compare two compound strings byte-by-byte.

Synopsis

Boolean XmStringByteCompare (XmString *string1*, XmString *string2*)

Inputs

string1	Specifies a compound string.
string2	Specifies another compound string.

Returns

True if the two compound strings are byte-by-byte identical or False otherwise.

Description

`XmStringByteCompare`() compares the compound strings *string1* and *string2* byte by byte. If the strings are equivalent, it returns True; otherwise it returns False. If two compound strings are created with `XmStringCreateLocalized`() in the same language environment, using the same character string, the strings are byte-for-byte equal. Similarly, if two compound strings are created with `XmStringCreate`() using the same font list element tag and character string, the strings are equal.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringByteCompare`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

When a compound string is placed into a widget, the string is sometimes converted to an internal format, which provides faster processing but strips out redundant information. As a result, when an application retrieves the compound string from the widget by calling `XtGetValues`(), the returned string does not necessarily match the original string byte-for-byte. This situation occurs most often with Label widgets and its subclasses.

See Also

`XmStringComponentCreate`(1), `XmStringCompare`(1).

Name

XmStringByteStreamLength – calculates the length of a byte stream.

Synopsis

unsigned int XmStringByteStreamLength (unsigned char *_string_)

Inputs

string Specifies a string in byte stream format.

Returns

The length, in bytes, of the string.

Availability

Motif 2.0 and later.

Description

`XmStringByteStreamLength`() calculates and returns the length of a byte stream *string* in bytes, including any header information. The *string* is presumed to be a compound string which has been converted into byte stream format.

Usage

Since the returned value includes the size of the stream header, the function returns a non-zero value even if *string* is NULL. The function is primarily used as part of data transfer operations, for example in transferring compound string tables to and from the clipboard or other widgets.

See Also

`XmCvtXmStringToByteStream`(1),
`XmCvtByteStreamToXmString`(1).

Name

XmStringCompare – compare two compound strings.

Synopsis

Boolean XmStringCompare (XmString *string1*, XmString *string2*)

Inputs

string1	Specifies a compound string.
string2	Specifies another compound string.

Returns

True if the two compound strings are semantically equivalent or False otherwise.

Description

`XmStringCompare`() compares the compound strings *string1* and *string2* semantically. If the strings are equivalent, it returns True; otherwise it returns False. `XmStringCompare`() is similar to `XmStringByteCompare`() but less restrictive. Two compound string are semantically equivalent if they have the same text components, font list element tags, directions, and separators.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCompare`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringcomponentCreate`(1), `XmStringByteCompare`(1).

Name

XmStringComponentCreate – create a compound string consisting of a single component.

Synopsis

XmString XmStringComponentCreate (	XmStringComponentType	*type*,
	unsigned int	*length*,
	XtPointer	*value*)

Inputs

type	Specifies the type of component to create.
length	Specifies the length, in bytes, of value.
value	Specifies the value of the component.

Returns

A new compound string, or NULL.

Availability

Motif 2.0 and later.

Description

`XmStringComponentCreate`() creates a new compound string consisting of a component of the type specified by *type*, which contains the given *value*.

Usage

If *type* is not a valid component type, or if *length* is greater than zero and *value* is NULL, then the function returns NULL. Otherwise, the function returns an allocated compound string. It is the responsibility of the programmer to reclaim the utilized space at an appropriate point by calling `XmStringFree`().

Structures

The string component *type* can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_TAB
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN
XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP

XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

Example

The following code illustrates basic compound string creation by concatenating elements from an array of strings:

```
XmString create_xmstring_from_array (char **array, int count, Boolean tab)
{
    XmString                txt, sep;
    XmString                xms = (XmString) 0;
    XmStringComponentType   sep_type;
    int                     i;

    if (tab) {
        sep_type = XmSTRING_COMPONENT_TAB;
    }
    else {
        sep_type = XmSTRING_COMPONENT_SEPARATOR;
    }

    for (i = 0; i < count; i++) {
        txt = XmStringComponentCreate
                                        (XmSTRING_COMPONENT_TE
                                        XT, strlen (array[i]), (XtPointer)
                                        array[i]);
        xms = XmStringConcatAndFree (xms, txt);

        if (i < count) {
            /* another item after this... */
            sep = XmStringComponentCreate (sep_type, 0, NULL);
            xms = XmStringConcatAndFree (xms, sep);
        }
    }

    /* caller must free this */
    return xms;
}
```

See Also

`XmStringConcatAndFree`(1), `XmStringFree`(1).

Name

XmStringConcat – concatenate two compound strings.

Synopsis

XmString XmStringConcat (XmString *string1*, XmString *string2*)

Inputs

string1	Specifies a compound string.
string2	Specifies another compound string.

Returns

A new compound string.

Description

`XmStringConcat`() returns the compound string formed by appending *string2* to *string1*, leaving the original compound strings unchanged. Storage for the result is allocated within the routine and should be freed by calling `XmString-Free`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringConcat`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringComponentCreate`(1), `XmStringCopy`(1), `XmStringNConcat`(1), `XmStringNCopy`(1).

Name

XmStringConcatAndFree – concatenate two compound strings.

Synopsis

XmString XmStringConcatAndFree (XmString *string1*, XmString *string2*)

Inputs

string1 Specifies a compound string.
string2 Specifies another compound string.

Returns

A new compound string.

Availability

Motif 2.0 and later.

Description

`XmStringConcatAndFree`() is similar to `XmStringConcat`() in that each returns a compound string formed by appending *string2* to *string1*. `XmStringConcatAndFree`() differs from `XmStringConcat`() by freeing the original compound strings. Storage for the result is allocated within the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, and locale components.

Example

The following code constructs a simple compound string out of piecemeal sub-components:

```
XmString xms;
XmString xms_temp;

xms = XmStringGenerate((XtPointer) “Multiple”,
                       XmFONTLIST_DEFAULT_TAG,
                       XmCHARSET_TEXT,
                       NULL);
xms_temp = XmStringComponentCreate
                                    (XmSTRING_COMP
                                    ONENT_TAB, 0,
                                    NULL);
xms = XmStringConcatAndFree(xms, xms_temp);
```

```
xms_temp = XmStringGenerate((XtPointer) “Column”,
                            XmFONTLIST_DEFAULT_TAG
                            ,
                            XmCHARSET_TEXT,
                            NULL);
xms = XmStringConcatAndFree (xms, xms_temp);
xms_temp = XmStringComponentCreate
                                  (XmSTRING_COMP
                                  ONENT_TAB, 0,
                                  NULL);
xms = XmStringConcatAndFree (xms, xms_temp);
xms_temp = XmStringGenerate((XtPointer) “Format”,
                            XmFONTLIST_DEFAULT_TAG
                            ,
                            XmCHARSET_TEXT,
                            NULL);
xms = XmStringConcatAndFree (xms, xms_temp);
```

See Also

XmStringComponentCreate(1), XmStringCopy(1), XmStringConcat(1), XmStringNConcat(1), XmStringNCopy(1).

Name

XmStringCopy – copy a compound string.

Synopsis

XmString XmStringCopy (XmString *string*)

Inputs

string Specifies a compound string.

Returns

A new compound string.

Description

`XmStringCopy`() copies the compound string *string* and returns the copy, leaving the original compound string unchanged. Storage for the result is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCopy`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringComponentCreate`(1), `XmStringConcat`(1), `XmStringNConcat`(1), `XmStringNCopy`(1).

Name

XmStringCreate – create a compound string.

Synopsis

XmString XmStringCreate (char *_text_, XmStringCharSet _tag_)

Inputs

text	Specifies the text component of the compound string.
tag	Specifies the font list element tag.

Returns

A new compound string.

Description

`XmStringCreate`() creates a compound string containing two components: a text component composed of *text* and the font list element tag specified by *tag*. *text* must be a NULL-terminated string. *tag* can have the value XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCreate`() allows you to create a compound string composed of a font list element tag and a text component.

In Motif 1.1, compound strings use character set identifiers rather than font list element tags. The character set identifier for a compound string can have the value XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

`XmStringCreate`() creates a compound string with no specified direction. The default direction may be taken from the XmNstringDirection resource of the parent of the widget that contains the compound string. If you need a string with a direction other than the default direction, use `XmStringDirectionCreate`() to create a direction string and concatenate it with the compound string containing the text.

Example

The following code fragment shows how to create compound strings using `XmStringCreate`():

```
Widget        toplevel;
XmString      s1, s2, s3, text, tmp;
String        string1 = "This is a string", string2 = "that contains three", string3 =
"separate fonts.";

s1 = XmStringCreate (string1, "tag1");
s2 = XmStringCreate (string2, "tag2");
s3 = XmStringCreate (string3, XmFONTLIST_DEFAULT_TAG);

tmp = XmStringConcatAndFree (s1, s2);
text = XmStringConcatAndFree (tmp, s3);

XtVaCreateManagedWidget ("widget_name", xmLabelWidgetClass, toplevel,
                                        XmNlabelString, text, NULL);
XmStringFree (text);
```

See Also

`XmStringBaseline`(1), `XmStringByteCompare`(1), `XmStringCompare`(1), `XmStringConcat`(1), `XmStringComponentCreate`(1), `XmStringCopy`(1), `XmStringCreateLocalized`(1), `XmStringCreateLtoR`(1), `XmStringCreateSimple`(1), `XmStringDirectionCreate`(1), `XmStringDraw`(1), `XmStringDrawImage`(1), `XmStringDrawUnderline`(1), `XmStringEmpty`(1), `XmStringExtent`(1), `XmStringFree`(1), `XmStringFreeContext`(1), `XmStringGetLtoR`(1), `XmStringGetNextComponent`(1), `XmStringGetNextSegment`(1), `XmStringHasSubstring`(1), `XmStringHeight`(1), `XmStringInitContext`(1), `XmStringLength`(1), `XmStringLineCount`(1), `XmStringNConcat`(1), `XmStringNCopy`(1), `XmStringPeekNextComponent`(1), `XmStringSegmentCreate`(1), `XmStringSeparatorCreate`(1), `XmStringWidth`(1).

Name

XmStringCreateLocalized – create a compound string in the current locale.

Synopsis

XmString XmStringCreateLocalized (String *text*)

Inputs

text Specifies the text component of the compound string.

Returns

A new compound string.

Availability

Motif 1.2 and later.

Description

`XmStringCreateLocalized`() creates a compound string containing two components: a text component composed of *text* and the font list element tag XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. *text* must be a NULL-terminated string. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCreateLocalized`() creates the identical compound string that would result from calling XmStringCreate with XmFONTLIST_DEFAULT_TAG as the font list entry tag.

Example

The following program shows how to create a compound string in the current locale and use it as the label for a PushButton:

```
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>

String fallbacks[] = { "*fontList:9x15=tag", NULL };

main (int argc, char *argv[])
{
    Widget          toplevel, rowcol;
    XtAppContext    app;
```

```
    XmString        text;

    XtSetLanguageProc (NULL, (XtLanguageProc) NULL, NULL);

    toplevel = XtVaAppInitialize (&app, argv[0], NULL, 0, &argc, argv, fall-
    backs, NULL);
    rowcol = XtVaCreateWidget ("rowcol", xmRowColumnWidgetClass,
    toplevel, NULL);

    text = XmStringCreateLocalized ("Testing, testing...");
    XtVaCreateManagedWidget ("pb", xmPushButtonWidgetClass, rowcol,
                                        XmNlabelString, text, NULL);

    XmStringFree (text);
    XtManageChild (rowcol);
    XtRealizeWidget (toplevel);
    XtAppMainLoop (app);
}
```

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringFree`(1).

Name

XmStringCreateLtoR – create a compound string.

Synopsis

XmString XmStringCreateLtoR (char **text*, XmStringCharSet *tag*)

Inputs

text	Specifies the text component of the compound string.
tag	Specifies the font list element tag.

Returns

A new compound string.

Availability

In Motif 2.0 and later, this function is obsolete, and is replaced by the function `XmStringGenerate`().

Description

`XmStringCreateLtoR`() creates a compound string containing two components: a text component composed of *text* and the font list element tag specified by *tag*. text must be a NULL-terminated string. In addition, `XmStringCreateLtoR`() searches for newline characters (\n) in *text*. Each time a newline is found, the characters up to the newline are placed into a compound string segment followed by a separator component. The routine does not add a separator component to the end of the compound string. The default direction of the string is left to right and the assumed encoding is 8-bit characters rather than 16-bit characters.

tag can have the value XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element *tag*, a string direction, and a *text* component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringCreateLtoR`() allows you to create a compound string composed of a font list element *tag* and a multi-line *text* component.

In Motif 1.1, compound strings use character set identifiers rather than font list element tags. The character set identifier for a compound string can have the value XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringFree`(1). `XmStringGenerate`(1).

Name

XmStringCreateSimple – create a compound string in the current language environment.

Synopsis

XmString XmStringCreateSimple (char *_text_)

Inputs

text Specifies the text component of the compound string.

Returns

A new compound string.

Availability

In Motif 1.2, `XmStringCreateSimple`() is obsolete. It has been superseded by `XmStringCreateLocalized`().

Description

`XmStringCreateSimple`() creates a compound string containing two components: a text component composed of _text_ and a character set identifier derived from the LANG environment variable or from a vendor-specific default, which is usually ISO8859-1. _text_ must be a NULL-terminated string. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. In Motif 1.1, compound strings use character set identifiers rather than font list element tags. `XmStringCreateSimple`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringCreateLocalized`(1), `XmStringFree`(1).

Name

XmStringDirectionCreate – create a compound string containing a direction component.

Synopsis

XmString XmStringDirectionCreate (XmStringDirection *direction*)

Inputs

direction Specifies the value of the direction component. Pass either XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L or XmSTRING_DIRECTION_DEFAULT.

Returns

A new compound string.

Description

`XmStringDirectionCreate`() creates a compound string containing a single component, which is a direction component with the specified *direction* value. If the *direction* is XmSTRING_DIRECTION_DEFAULT, the widget where the compound string is rendered controls the direction. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDirectionCreate`() allows you to create a string direction component that can be concatenated with a compound string containing other components.

See Also

`XmStringComponentCreate`(1), `XmStringCreate`(1), `XmStringFree`(1).

Name

XmStringDirectionToDirection – converts a string direction to a direction.

Synopsis

XmDirection XmStringDirectionToDirection (XmStringDirection *string_direction*)

Inputs

string_direction Specifies the string direction to be converted.

Returns

The converted direction.

Availability

Motif 2.0 and later.

Description

`XmStringDirectionToDirection`() converts an XmStringDirection value specified by *string_direction* into an XmDirection value.

Usage

`XmStringDirectionToDirection`() converts between the XmStringDirection and XmDirection data types. If *string_direction* is XmSTRING_DIRECTION_LEFT_TO_RIGHT, the function returns XmLEFT_TO_RIGHT. If *string_direction* is XmSTRING_DIRECTION_RIGHT_TO_LEFT, the function returns XmRIGHT_TO_LEFT. Otherwise, the function returns XmDIRECTION_DEFAULT.

See Also

`XmStringDirectionCreate`(1).

Name

XmStringDraw – draw a compound string.

Synopsis

```
void XmStringDraw ( Display        *display,
                    Window         window,
                    XmFontList     fontlist,
                    XmString       string,
                    GC             gc,
                    Position       x,
                    Position       y,
                    Dimension      width,
                    unsigned char  alignment,
                    unsigned char  layout_direction,
                    XRectangle     *clip)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies the window where the string is drawn.
fontlist	Specifies the font list for drawing the string.
string	Specifies a compound string.
gc	Specifies the graphics context that is used to draw the string.
x	Specifies the x-coordinate of the rectangle that will contain the string.
y	Specifies the y-coordinate of the rectangle that will contain the string.
width	Specifies the width of the rectangle that will contain the string.
alignment	Specifies the alignment of the string in the rectangle. Pass one of the following values: XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, or XmALIGNMENT_END.
layout_direction	Specifies the layout direction of the string segments. Pass XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L, or XmSTRING_DIRECTION_DEFAULT.
clip	Specifies an clip rectangle that restricts the area where the string will be drawn.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringDraw`() draws the compound string specified by string by rendering the foreground pixels for each character. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDraw`() provides a means of rendering a compound string that is analogous to the Xlib string rendering routines. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget.

In Motif 1.2 or later, if a segment of a compound string is associated with a font list entry that is a font set, the font member of the *gc* is left in an undefined state by the underlying call to `XmbDrawString`(). If a segment of the compound string is not associated with a font set, the gc must contain a valid font member. The *gc* must be created using `XtAllocateGC`(); graphics contexts created with `XtGetGC`() are not valid.

See Also

`XmStringDrawImage`(1), `XmStringDrawUnderline`(1), `XmRendition`(2).

Name

XmStringDrawImage – draw a compound string.

Synopsis

```
void XmStringDrawImage (  Display        *display,
                          Window         window,
                          XmFontList     fontlist,
                          XmString       string,
                          GC             gc,
                          Position       x,
                          Position       y,
                          Dimension      width,
                          unsigned char  alignment,
                          unsigned char  layout_direction,
                          XRectangle     *clip)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies the window where the string is drawn.
fontlist	Specifies the font list for drawing the string.
string	Specifies a compound string.
gc	Specifies the graphics context that is used to draw the string.
x	Specifies the x-coordinate of the rectangle that will contain the string.
y	Specifies the y-coordinate of the rectangle that will contain the string.
width	Specifies the width of the rectangle that will contain the string.
alignment	Specifies the alignment of the string in the rectangle. Pass one of the following values: XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, or XmALIGNMENT_END.
layout_direction	Specifies the layout direction of the string segments. Pass XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L, or XmSTRING_DIRECTION_DEFAULT.
clip	Specifies an clip rectangle that restricts the area where the string will be drawn.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringDrawImage`() draws the compound string specified by *string* by painting the foreground and background pixels for each character. If string is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDrawImage`() provides a means of rendering a compound string that is analogous to the Xlib string rendering routines. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget.

In Motif 1.2 or later, if a segment of a compound string is associated with a font list entry that is a font set, the font member of the *gc* is left in an undefined state by the underlying call to `XmbDrawImageString`(). If a segment of the compound string is not associated with a font set, the *gc* must contain a valid font member. The *gc* must be created using `XtAllocateGC`(); graphics contexts created with `XtGetGC`() are not valid.

See Also

`XmStringDraw`(1), `XmStringDrawUnderline`(1), `XmRendition`(2).

Name

XmStringDrawUnderline – draw a compound string with an underlined sub-string.

Synopsis

```
void XmStringDrawUnderline (  Display        *display,
                              Window         window,
                              XmFontList     fontlist,
                              XmString       string,
                              GC             gc,
                              Position       x,
                              Position       y,
                              Dimension      width,
                              unsigned char  alignment,
                              unsigned char  layout_direction,
                              XRectangle     *clip,
                              XmString       underline)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
window	Specifies the window where the string is drawn.
fontlist	Specifies the font list for drawing the string.
string	Specifies a compound string.
gc	Specifies the graphics context that is used to draw the string.
x	Specifies the x-coordinate of the rectangle that will contain the string.
y	Specifies the y-coordinate of the rectangle that will contain the string.
width	Specifies the width of the rectangle that will contain the string.
alignment	Specifies the alignment of the string in the rectangle. Pass one of the following values: XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, or XmALIGNMENT_END.
layout_direction	Specifies the layout direction of the string segments. Pass XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L, or XmSTRING_DIRECTION_DEFAULT.
clip	Specifies an clip rectangle that restricts the area where the string will be drawn.

underline Specifies the substring that is to be underlined.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringDrawUnderline`() is similar to `XmStringDraw`(), but it also draws an underline beneath the first matching substring *underline* that is contained within *string*. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font associated with the character set from the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringDrawUnderline`() provides a means of rendering a compound string and underlining a substring within it. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget.

In Motif 1.2 and later, if a segment of a compound string is associated with a font list entry that is a font set, the font member of the *gc* is left in an undefined state by the underlying call to `XmbDrawString`(). If a segment of the compound string is not associated with a font set, the *gc* must contain a valid font member. The *gc* must be created using `XtAllocateGC`(); graphics contexts created with `XtGetGC`() are not valid.

See Also

`XmStringDraw`(1), `XmStringDrawImage`(1), `XmRendition`(2).

Name

XmStringEmpty – determine whether there are text segments in a compound string.

Synopsis

Boolean XmStringEmpty (XmString *string*)

Inputs

string Specifies a compound string.

Returns

True if there are no text segments in the string or False otherwise.

Description

`XmStringEmpty`() returns True if no text segments exist in the specified *string* and False otherwise. If the routine is passed NULL, it returns True.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringEmpty`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringLength`(1), `XmStringLineCount`(1).

Name

XmStringExtent – get the smallest rectangle that contains a compound string.

Synopsis

void XmStringExtent (XmFontList *fontlist*, XmString *string*, Dimension **width*, Dimension **height*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Outputs

width	Returns the width of the containing rectangle.
height	Returns the height of the containing rectangle.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringExtent`() calculates the size of the smallest rectangle that can enclose the specified compound *string* and returns the *width* and *height* of the rectangle in pixels. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font from the character set of the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringExtent`() provides information that is useful if you need to render a compound string. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringBaseline`(1), `XmStringHeight`(1), `XmStringWidth`(1), `XmRendition`(2).

Name

XmStringFree – free the memory used by a compound string.

Synopsis

void XmStringFree (XmString *string*)

Inputs

string Specifies the compound string.

Description

`XmStringFree`() frees the memory used by the specified compound string.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. All of the routines that return a compound string allocate memory for the string. An application is responsible for this storage; `XmStringFree`() provides a way to free the memory.

When `XtGetValues`() is called for a resource that contains an XmString, a copy of the compound string is returned. The allocated storage is again the responsibility of the application and can be freed using `XmStringFree`().

Example

The following code fragment shows the use of `XmStringFree`():

```
Widget      toplevel, rowcol, pb;
XmString    str;
char        *text;

rowcol = XtVaCreateWidget ("rowcol", xmRowColumnWidgetClass, toplevel,
NULL);

str = XmStringCreateLocalized ("Testing, testing...");
pb = XtVaCreateManagedWidget ("pb", xmPushButtonWidgetClass, rowcol,
                                    XmNlabelString, str, NULL);
XmStringFree (str);
...
XtVaGetValues (pb, XmNlabelString, &str, NULL);
text = (char *) XmStringUnparse (str, NULL,
                                    XmCHARSET_TEXT,
                                    XmCHARSET_TEXT,
                                    NULL, 0, XmOUTPUT_ALL)[1];
```

```
printf ("PushButton's label is %s\n", text);
XmStringFree (str);
XtFree (text);
```

See Also

`XmStringCreate`(1), `XmStringCreateLocalized`(1), `XmStringCreateLtoR`(1), `XmStringCreateSimple`(1), `XmStringDirectionCreate`(1), `XmStringSegmentCreate`(1), `XmStringSeparatorCreate`(1).

1.Erroneously given as XmStringGetLtoR() in 2nd edition. XmStringGetLtoR() is deprecated from Motif 2.0 onwards.

Name

XmStringFreeContext – free a string context.

Synopsis

void XmStringFreeContext (XmStringContext *context*)

Inputs

context Specifies the string context that is to be freed.

Description

`XmStringFreeContext`() deallocates the string context structure specified by context.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. XmStringFreeContext() is the last of the string context routines that an application should call when processing a compound string, as it frees the string context data structure. An application begins by calling XmStringInitContext() to create a string context and then makes repeated calls to either XmStringGetNextComponent() or XmStringGetNextSegment() to cycle through the compound string.

The most common use of these routines is in converting a compound string to a regular character string when the compound string uses multiple fontlist element tags or it has a right-to-left orientation.

Example

The following code fragment shows how to convert a compound string into a character string:

```
XmString            str;
XmStringContext     context;
char                *text, buf[128], *p;
XmStringCharSet     tag;
XmStringDirection   direction;
Boolean             separator;

XtVaGetValues (widget, XmNlabelString, &str, NULL);

if (!XmStringInitContext (&context, str)) {
    XmStringFree (str);
    XtWarning ("Can’t convert compound string.");
    return;
```

```
}

/* p keeps a running pointer thru buf as text is read */ p = buf;
while (XmStringGetNextSegment (context, &text, &tag, &direction, &separator)) {
    /* copy text into p and advance to the end of the string */
    p += (strlen (strcpy (p, text)));

    if (separator == True) {
        /* if there's a separator... */
        *p++ = '\n';
        *p = 0;  /* add newline and null-terminate */
    }
    XtFree (text); /* we're done with the text; free it */
}

XmStringFreeContext (context);
XmStringFree (str);

printf ("Compound string:\n%s\n", buf);
```

See Also

`XmStringInitContext`(1), `XmStringGetNextSegment`(1), `XmStringGetNextComponent`(1), `XmStringPeekNextComponent`(1).

Name

XmStringGenerate – generate a compound string.

Synopsis

```
XmString XmStringGenerate ( XtPointer     text,
                            XmStringTag   tag,
                            XmTextType    type,
                            XmStringTag   rendition)
```

Inputs

text	Specifies the data forming the value of the compound string.
tag	Specifies the tag used in creating the compound string.
type	Specifies the type of text.
rendition	Specifies a rendition tag.

Returns

A new compound string.

Availability

Motif 2.0 and later.

Description

`XmStringGenerate`() is a convenience function which invokes `XmString-ParseText`() using a default parse table in order to convert *text* into a compound string. The default parse table maps tab characters to XmSTRING_COMPONENT_TAB, and newline characters to XmSTRING_COMPONENT_SEPARATOR components of the compound string. If a *rendition* tag is specified, the resulting compound string is placed within matching components of type XmSTRING_RENDITION_BEGIN and XmSTRING_RENDITION_END which contain the *rendition*. The type of the input *text* is specified by *type*, and is one of XmCHARSET_TEXT, XmWIDECHAR_TEXT, or XmMULTIBYTE_TEXT. *type* also specifies the type of the *tag* which is used in creating the compound string. If *tag* is NULL and the input *text* is of type XmCHARSET_TEXT, then the compound string is created with the *tag* set to XmFONTLIST_DEFAULT_TAG. If *tag* is NULL and the input *text* is of type XmWIDECHAR_TEXT or XmMULTIBYTE_TEXT, then the *tag* used is constructed from the value of _MOTIF_DEFAULT_LOCALE.

Usage

The function returns allocated storage, and it is the responsibility of the programmer to reclaim the space by calling `XmStringFree`() at an appropriate point.

In Motif 2.0 and later, in common with other objects, the compound string is manipulated as a reference counted data structure. XmString functions prior to Motif 2.0 handle ASN.1 strings, and the data structures are only used internally.

Example

The following code converts data taken from a Text widget into a compound string:

```
XmString convert_text (Widget text)
{
    /* ignoring widechar text values */
    char        *value = XmTextGetString (text);
    XmString    xms   = (XmString) 0;

    if (value) {
        xms = XmStringGenerate ((XtPointer) value,
                                XmFONTLIST_DEFAULT_TAG,
                                XmCHARSET_TEXT, NULL);
        XtFree (value);
    }
    /* caller must free this */
    return xms;
}
```

See Also

`XmStringFree(1), XmStringPutRendition`(1), `XmRendition`(2).

Name

XmStringGetLtoR – get a text segment from a compound string.

Synopsis

Boolean XmStringGetLtoR (XmString *string*, XmStringCharSet *tag*, char *******text*)

Inputs

string	Specifies the compound string.
tag	Specifies the font list element tag.

Outputs

text	Returns the NULL-terminated character string.

Returns

True if there is a matching text segment or False otherwise.

Availability

In Motif 2.0 and later, the function is obsolete, and is replaced by `XmStringUnparse`().

Description

`XmStringGetLtoR`() looks for a text segment in *string* that matches the font list element tag specified by *tag*. *tag* can have the value XmFONTLIST_DEFAULT_TAG, which identifies a locale-encoded text segment. The routine returns True if a text segment is found. *text* returns a pointer to the NULL-terminated character string that contains the text from the segment. Storage for the returned character string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringGetLtoR`() allows you to retrieve a character string from a compound string, so that you can use the string with the standard C string manipulation functions.

In Motif 1.1, compound strings use character set identifiers rather than font list element tags. The character set identifier for a compound string can have the value XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

`XmStringGetLtoR`() gets the first text segment from the compound string that is associated with the specified tag. If the *string* contains multiple font list element tags, you must cycle through the compound string and retrieve each segment individually in order to retrieve the entire string. The routine only gets strings with a left-to-right orientation.

See Also

`XmStringCreate`(1), `XmStringCreateLtoR`(1), `XmStringGetNextSegment`(1), `XmStringUnparse`(1).

Name

XmStringGetNextComponent – retrieves information about the next compound string
component.

Synopsis

```
XmStringComponentType
XmStringGetNextComponent (  XmStringContext          context,
                            char                     **text,
                            XmStringCharSet          *tag,
                            XmStringDirection        *direction,
                            XmStringComponentType    *unknown_tag,
                            unsigned short
*unknown_length,
                            unsigned char
**unknown_value)
```

Inputs

context	Specifies the string context for the compound string.

Outputs

text	Returns the NULL-terminated string for a text component.
tag	Returns the font list element tag for a tag component.
direction	Returns the string direction for a direction component.
unknown_tag	Returns the tag of an unknown component.
unknown_length	Returns the length of an unknown component.
unknown_value	Returns the value of an unknown component.

Returns

The type of the compound string component. The type is one of the values described below.

Availability

In Motif 2.0 and later, `XmStringGetNextComponent`() is obsolete, and is replaced by `XmStringGetNextTriple`().

Description

`XmStringGetNextComponent`() reads the next component in the compound string specified by *context* and returns the type of component found. The return value indicates which, if any, of the output parameters are valid. Storage for the returned values is allocated by the routine and must be freed by the application using `XtFree`().

For the type XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG, the font list element tag is returned in *tag*. In Motif 2.0 and later, the type

XmSTRING_COMPONENT_CHARSET is obsolete and is retained for compatibility with Motif 1.2. The type indicates that the character set identifier is returned in *tag*. XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG replaces XmSTRING_COMPONENT_CHARSET.

For the string component types XmSTRING_COMPONENT_TEXT and XmSTRING_COMPONENT_LOCALE_TEXT, the text string is returned in *text*. For XmSTRING_COMPONENT_DIRECTION, the direction is returned in *direction*. Only one of *tag*, *text*, and *direction* can be valid at any one time.

The type XmSTRING_COMPONENT_SEPARATOR indicates that the next component is a separator, while XmSTRING_COMPONENT_END specifies the end of the compound string. For type XmSTRING_COMPONENT_UNKNOWN, the tag, length, and value of the unknown component are returned in the corresponding arguments.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is called first to create the string context. `XmStringGetNextComponent`() cycles through the components in the compound string. When an application is done processing the string, it should call `XmStringFreeContext`() with the same context to free the allocated data.

Structures

A XmStringComponentType can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN

In Motif 2.0 and later, the following additional types are defined:

XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

See Also

XmStringFreeContext(1), XmStringGetNextTriple(1), XmStringGetNextSegment(1), XmStringInitContext(1), XmStringPeekNextComponent(1).

Name

XmStringGetNextSegment – retrieves information about the next compound string segment.

Synopsis

```
Boolean XmStringGetNextSegment (  XmStringContext   context,
                                  char              **text,
                                  XmStringCharSet   *charset,
                                  XmStringDirection *direction,
                                  Boolean           *separator)
```

Inputs

context	Specifies the string context for the compound string.

Outputs

text	Returns the NULL-terminated string for the segment.
tag	Returns the font list element tag for the segment.
direction	Returns the string direction for the segment.
separator	Returns whether or not the next component is a separator.

Returns

True if a valid segment is located or False otherwise.

Availability

In Motif 2.0 and later, the function is obsolete, and is replaced by `XmStringGetNextTriple`().

Description

`XmStringGetNextSegment`() retrieves the text string, font list element tag, and direction for the next segment of the compound string specified by *context*. The routine returns True if a valid segment is retrieved; otherwise, it returns False. Storage for the returned text is allocated by the routine and must be freed by the application using `XtFree`().

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is called first to create the string *context*. `XmStringGetNextSegment`() cycles through the segments in the compound string. The Boolean *separator* can be used to determine whether or not the next component in the compound string is a separator. When an application is done processing the string, it should call `XmStringFreeContext`() with the same context to free the allocated data.

The most common use of these routines is in converting a compound string to a regular character string when the compound string uses multiple fontlist element tags or it has a right-to-left orientation.

See Also

XmStringFreeContext(1), XmStringGetLtoR(1), XmStringGetNextComponent(1), XmStringGetNextTriple(1), XmStringInitContext(1), XmStringPeekNextComponent(1), XmStringPeekNextTriple(1).

Name

XmStringGetNextTriple – retrieve information about the next component.

Synopsis

XmStringComponentType
XmStringGetNextTriple (XmStringContext *context*, unsigned int **length*, XtPointer **value*)

Inputs

context Specifies the string context for the compound string.

Outputs

length Returns the length of the value of the component.
value Returns the value of the component.

Returns

The type of the component.

Availability

Motif 2.0 and later.

Description

`XmStringGetNextTriple`() is a convenience function which returns the type, *length*, and *value* of the next component within the compound string associated with *context*. The context is an opaque structure used for walking along compound strings one component at a time, and is initialized through a call to `XmStringInitContext`().

Usage

If either of *value* or *length* are NULL pointers, the function immediately returns XmSTRING_COMPONENT_END without fetching the next string segment. Otherwise, *value* is initially set to point to NULL, and *length* is reset to zero, and the next segment is processed. The function allocates memory for the returned value, which should be reclaimed at an appropriate point by calling `XtFree`().

Structures

An XmStringComponentType can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN

In Motif 2.0 and later, the following additional types are defined:

XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

Example

The following code fragment shows how to convert a compound string into a character string:

```
XmString                str;
XmStringContext         context;
char                    *text, buf[128], *p;
XmStringComponentType   type;
unsigned int            len;

/* Fetch the Compound String from somewhere */
XtVaGetValues (widget, XmNlabelString, &str, NULL);

if (!XmStringInitContext (&context, str)) {
    XmStringFree (str);
    XtWarning ("Can't convert compound string.");
    return;
}

/* p keeps a running pointer through buf as text is read */
p = buf;

/* Ignoring locale or widechar text for simplicity */
while ((type = XmStringGetNextTriple (context, &len, &text)) !=
XmSTRING_COMPONENT_END)
{
    switch (type) {
        case XmSTRING_COMPONENT_TAB             :
                        *p++ = '\t';
                        break;
        case XmSTRING_COMPONENT_SEPARATOR       :
                        *p++ = '\n';
                        *p   = '\0';
                        break;
        case XmSTRING_COMPONENT_TEXT            :
                        (void) strcpy (p, text);
```

```
                        p += len;
                        break;
        }
        XtFree (text);
}

XmStringFreeContext (context);
XmStringFree (str);
printf ("Compound string:\n%s\n", buf);
```

See Also

`XmStringFreeContext`(1), `XmStringGetNextComponent`(1), `XmStringGetNextSegment`(1), `XmStringInitContext`(1), `XmStringPeekNextComponent`(1), `XmStringPeekNextTriple`(1).

Name

XmStringHasSubstring – determine whether a compound string contains a substring.

Synopsis

Boolean XmStringHasSubstring (XmString *string*, XmString *substring*)

Inputs

string	Specifies the compound string.
substring	Specifies the substring.

Returns

True if *string* contains *substring* or False otherwise.

Description

`XmStringHasSubstring`() determines whether the compound string *substring* is contained within any single segment of the compound string *string*. *substring* must have only a single segment. The routine returns True if the *string* contains the *substring* and False otherwise.

If two compound strings are created with `XmStringCreateLocalized`() in the same language environment and they satisfy the above condition, `XmStringHasSubstring`() returns True. If two strings are created with `XmStringCreate`() using the same character set and they satisfy the condition, the routine also returns True. When comparing a compound string created by `XmStringCreate`() with a compound string created by `XmStringCreateSimple`() the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringHasSubstring`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringEmpty`(1), `XmStringLength`(1), `XmStringLineCount`(1).

Name

XmStringHeight – get the line height of a compound string.

Synopsis

Dimension XmStringHeight (XmFontList *fontlist*, XmString *string*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Returns

The height of the compound string.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringHeight`() returns the height, in pixels, of the specified compound *string*. If *string* contains multiple lines, where a separator component delimits each line, then the total height of all of the lines is returned. If *string* is created with `XmStringCreateSimple`(), then *fontlist* must begin with the font from the character set of the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringHeight`() provides information that is useful if you need to render a compound string. Motif widgets render compound string automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringBaseline`(1), `XmStringExtent`(1), `XmStringWidth`(1), `XmRendition`(2).

Name

XmStringInitContext – create a string context.

Synopsis

Boolean XmStringInitContext (XmStringContext **context,* XmString *string*)

Inputs

string Specifies the compound string.

Outputs

context Returns the allocated string context structure.

Returns

True if the string context is allocated or False otherwise.

Description

`XmStringInitContext`() creates a string context for the specified compound *string*. This string context allows an application to access the contents of a compound string.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is the first of the three string context routines that an application should call when processing a compound string, as it creates the string context data structure. The *context* is passed to `XmStringGetNextTriple`() to cycle through the compound string. When an application is done processing the string, it should call `XmStringFreeContext`() with the same *context* to free the allocated data.

The most common use of these routines is in converting a compound string to a regular character string when the compound string uses multiple fontlist element tags or it has a right-to-left orientation.

Example

The following code fragment shows how to convert a compound string into a character string:

```
XmString                str;
XmStringContext         context;
char                    *text, buf[128], *p;
XmStringComponentType   type;
unsigned int            len;

/* Fetch the Compound String from somewhere */
```

```
XtVaGetValues (widget, XmNlabelString, &str, NULL);

if (!XmStringInitContext (&context, str)) {
    XmStringFree (str);
    XtWarning ("Can't convert compound string.");
    return;
}

/* p keeps a running pointer through buf as text is read */
p = buf;

/* Ignoring locale or widechar text for simplicity */
while ((type = XmStringGetNextTriple (context, &len, &text)) !=
XmSTRING_COMPONENT_END)
{
    switch (type) {
        case XmSTRING_COMPONENT_TAB             :
            *p++ = '\t';
            break;
        case XmSTRING_COMPONENT_SEPARATOR       :
            *p++ = '\n';
            *p   = '\0';
            break;
        case XmSTRING_COMPONENT_TEXT            :
            (void) strcpy (p, text);
            p += len;
            break;
    }

    XtFree (text);
}

XmStringFreeContext (context);
XmStringFree (str);
printf ("Compound string:\n%s\n", buf);
```

See Also

`XmStringFreeContext`(1), `XmStringGetNextComponent`(1), `XmStringGetNextTriple`(1), `XmStringGetNextSegment`(1), `XmStringPeekNextComponent`(1), `XmStringPeekNextTriple`(1).

Name

XmStringIsVoid – determine whether there are valid segments in a compound string.

Synopsis

Boolean XmStringIsVoid (XmString *string*)

Inputs

string Specifies a compound string.

Returns

True if there are no segments in the string or False otherwise.

Availability

`XmStringIsVoid`() is available from Motif 2.0 or later.

Description

`XmStringIsVoid`() checks to see whether there any text, tab, or separator segments within the specified *string*. If the routine is passed NULL, it returns True. If the *string* contains text, tab or separator components, it returns False. Otherwise, it returns True.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, and locale components.

See Also

`XmStringEmpty`(1), `XmStringLength`(1), `XmStringLineCount`(1).

Name

XmStringLength – get the length of a compound string.

Synopsis

int XmStringLength (XmString *string*)

Inputs

string Specifies the compound string.

Returns

The length of the compound string.

Availability

In Motif 2.0 and later, the function is obsolete, and is replaced by `XmString-ByteStreamLength`().

Description

`XmStringLength`() returns the length, in bytes, of the specified compound *string*. The calculation includes the length of all tags, direction indicators, and separators. The routine returns 0 (zero) if the structure of *string* is invalid.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringLength`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings. However, this routine cannot be used to get the length of the text represented by the compound string; it is not the same as strlen().

See Also

`XmStringByteStreamLength`(1), `XmStringEmpty`(1), `XmStringLineCount`(1).

Name

XmStringLineCount – get the number of lines in a compound string.

Synopsis

int XmStringLineCount (XmString *string*)

Inputs

string Specifies the compound string.

Returns

The number of lines in the compound string.

Description

XmStringLineCount() returns the number of lines in the specified compound *string*. The line count is determined by adding 1 to the number of separators in the string.

Usage

In Motif 1.2 and later, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringLineCount`() provides information that is useful in laying out components that display compound strings.

Example

The following routine shows how to read the contents of a file into a buffer and then convert the buffer into a compound string. The routine also returns the number of lines in the compound string:

```
XmString ConvertFileToXmString (char *filename, int *lines)
{
    struct stat    statb;
    int            fd, len, lines;
    char           *text;
    XmString       str;

    *lines = 0;

    if ((fd = open (filename, O_RDONLY)) < 0) {
        XtWarning ("internal error -- can't open file");
        return (XmString) 0;
    }

    if ((fstat (fd, &statb) == -1) || !(text = XtMalloc ((len = statb.st_size) + 1))) {
        XtWarning("internal error -- can't show text");
```

```
        (void) close (fd);
        return (XmString) 0;
    }

    (void) read (fd, text, len);
    text[len] = '\0';
    str = XmStringGenerate ((XtPointer) text, XmFONTLIST_DEFAULT_TAG,
                            XmCHARSET_TEXT, NULL);[1]
    XtFree (text);
    (void) close (fd);
    *lines = XmStringLineCount (str);
    return str;
}
```

See Also

`XmStringEmpty`(1), `XmStringLength`(1).

1.Erroneously given as XmStringCreateLtoR() in 2nd edition. XmStringCreateLtoR() is deprecated from Motif 2.0 onwards.

Name

XmStringNConcat – concatenate a specified portion of a compound string to another compound string.

Synopsis

XmString XmStringNConcat (XmString *string1*, XmString *string2*, int *num_bytes*)

Inputs

string1	Specifies a compound string.
string2	Specifies the compound string that is appended.
num_bytes	Specifies the number of bytes of string2 that are appended.

Returns

A new compound string.

Availability

In Motif 2.0 and later, the function is obsolete, and is only maintained for backwards compatibility.

Description

`XmStringNConcat`() returns the compound string formed by appending bytes from *string2* to the end of *string1*, leaving the original compound strings unchanged. *num_bytes* of string are appended, which includes tags, directional indicators, and separators. Storage for the result is allocated within this routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

If *num_bytes* is less than the length of *string2*, the resulting string could be invalid. In this case, `XmStringNConcat`() appends as many bytes as possible, up to a maximum of *num_bytes*, to ensure the creation of a valid string.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringNConcat`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringConcat`(1), `XmStringCopy`(1), `XmStringNCopy`(1).

Name

XmStringNCopy – copy a specified portion of a compound string.

Synopsis

XmString XmStringNCopy (XmString *string*, int *num_bytes*)

Inputs

string	Specifies a compound string.
num_bytes	Specifies the number of bytes of string that are copied.

Returns

A new compound string.

Availability

In Motif 2.0 and later, the function is obsolete, and is only maintained for backwards compatibility.

Description

`XmStringNCopy`() copies *num_bytes* bytes from the compound string *string* and returns the resulting copy, leaving the original string unchanged. The number of bytes copied includes tags, directional indicators, and separators. Storage for the result is allocated within this routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

If *num_bytes* is less than the length of string, the resulting string could be invalid. In this case, `XmStringNCopy`() copies as many bytes as possible, up to a maximum of *num_bytes* to ensure the creation of a valid string.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringNCopy`() is one of a number of routines that allow an application to manipulate compound strings as it would regular character strings.

See Also

`XmStringConcat`(1), `XmStringCopy`(1), `XmStringNConcat`(1).

Name

XmStringParseText – convert a string to a compound string.

Synopsis

```
XmString XmStringParseText (  XtPointer      text,
                              XtPointer      *text_end,
                              XmStringTag    tag,
                              XmTextType     type,
                              XmParseTable   parse_table,
                              Cardinal       parse_count,
                              XtPointer      client_data)
```

Inputs

text	Specifies a string to be converted.
text_end	Specifies a pointer into text where parsing is to finish.
tag	Specifies the tag to be used in creating the compound string.
type	Specifies the type of the text and the tag.
parse_table	Specifies a table used for matching characters in the input text.
parse_count	Specifies the number of items in the parse_table.
client_data	Specifies application data to pass to any parse procedures within the parse_table.

Outputs

text_end	Returns a location within the text where parsing finished.

Returns

The converted compound string.

Availability

Motif 2.0 and later.

Description

`XmStringParseText`() converts the string specified by *text* into a compound string. A *parse_table* can be specified which consists of a set of mappings to control the conversion process. The contents of the string to be converted can be in one of a number of formats: simple characters, multibyte, or wide characters. The *type* parameter specifies the type of the input *text*, and is also used to interpret the *tag* which is used in creating text components within the returned compound string. *text_end* is both an input and an output parameter: as an input parameter, it specifies a location within *text* where parsing is to terminate; as an output parameter, it points to a location within *text* where parsing actually finished. Supplying NULL for *text_end* is interpreted to mean that parsing should stop at the occurrence of a null byte.

Usage

If *type* is XmCHARSET_TEXT, the input *text* is assumed to consist of a simple array of characters, and the *tag* is interpreted as the name of a charset to use in constructing the returned compound string. If *tag* is NULL, a default charset using XmFONTLIST_DEFAULT_TAG is used.

If the *type* is XmMULTIBYTE_TEXT or XmWIDECHAR_TEXT, the input *text* is assumed to be in multibyte or widechar text format respectively, and the *tag* is interpreted as a locale specifier. The *tag* should either be specified as NULL or _MOTIF_DEFAULT_LOCALE: if NULL, a locale component with a value of _MOTIF_DEFAULT_LOCALE is created in any case.

A parse table can be specified for controlling the conversion process. A parse table consists of a set of XmParseMapping objects, which have match pattern, substitution pattern and parse procedure components. The head of the input stream is compared against elements within the parse table, and if there is a correspondence between the input and a parse mapping match pattern, the parse mapping object is used to construct the output compound string at that point in the conversion, either by directly inserting the substitution pattern, or by invoking the parse procedure of the mapping object. The parse mapping specifies how the input pointer is advanced, and the process is repeated, comparing the head of the input against the parse table. At the end of the conversion, the *text_end* parameter is set to point to the location within the input *text* where parsing actually terminated. Depending upon the way in which the parse table interacts with the input *text*, the returned *text_end* may not be the same location which the programmer specified if more or less of the input *text* is consumed.

An implicit automatic conversion takes place where there is no matching parse mapping object for the head of the input. In other words, it is not necessary to provide a parse table to convert everything: parse tables are only required where specific inputs need to be handled specially. `XmStringParseText`() uses an internal parse mapping which handles changes in string direction in the absence of a supplied mapping for the task. A parse mapping handles string direction changes if the XmNpattern resource of the object is equal to XmDIRECTION_CHANGE.

`XmStringParseText`() allocates memory for the returned compound string. It is the responsibility of the programmer to reclaim the space through a call to `XmStringFree`() at an appropriate point.

Example

The following specimen code converts an input string containing tab and newline characters into a compound string:

```
XmParseTable    parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
XmString        tmp;
XmString        output;
Arg             av[4];
Cardinal        ac;

/* map \t to a tab component */
tmp = XmStringComponentCreate (XmSTRING_COMPONENT_TAB, 0,
NULL);
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\t");         ac++;
parse_table[0] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);

/* map \n to a separator component */
tmp = XmStringSeparatorCreate();
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\n");         ac++;
parse_table[1] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);

/* convert the (unspecified) input string into a compound string *
 output = XmStringParseText (input, NULL, NULL, XmCHARSET_TEXT,
parse_table, 2, NULL);
XmParseTableFree (parse_table);
```

See Also

`XmStringFree(1), XmStringGenerate`(1), `XmStringUnparse`(1). `XmParseMapping`(2).

Name

XmStringPeekNextComponent – returns the type of the next compound string component.

Synopsis

XmStringComponentType XmStringPeekNextComponent (XmStringContext *context*)

Inputs

context Specifies the string context for the compound string.

Returns

The type of the compound string component. The type is one of the values described below.

Availability

In Motif 2.0 and later, the function is obsolete, and `XmStringPeekNextTriple`() is preferred.

Description

`XmStringPeekNextComponent`() checks the next component in the compound string specified by *context* and returns the type of the component found. The routine shows what would be returned by a call to `XmStringGetNextComponent`(), without actually updating *context*.

The returned type XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG indicates that the next component is a font list element tag. In Motif 1.2, the type XmSTRING_COMPONENT_-CHARSET is obsolete and is retained for compatibility with Motif 1.1. The type indicates that the next component is a character set identifier. XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG replaces XmSTRING_COMPONENT_CHARSET.

The types XmSTRING_COMPONENT_TEXT and XmSTRING_COMPONENT_LOCALE_TEXT specify that the next component is text. XmSTRING_COMPONENT_DIRECTION indicates that the next component is a string direction component.

The type XmSTRING_COMPONENT_SEPARATOR indicates that the next component is a separator, while XmSTRING_COMPONENT_END specifies the end of the compound string. The type XmSTRING_COMPONENT_UNKNOWN, indicates that the type of the next component is unknown.

Usage

The XmString type is opaque, so if an application needs to perform any processing on a compound string, it has to use special functions to cycle through the string. These routines use a XmStringContext to maintain an arbitrary position in a compound string. `XmStringInitContext`() is called first to create the string context. `XmStringPeekNextComponent`() peeks at the next component in the compound string without cycling through the component. When an application is done processing the string, it should call `XmStringFreeContext`() with the same context to free the allocated data.

Structures

A XmStringComponentType can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN

In Motif 2.0 and later, the following additional types are defined:

XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

See Also

`XmStringFreeContext`(1), `XmStringGetNextComponent`(1), `XmStringGetNextSegment`(1), `XmStringPeekNextTriple`(1), `XmStringInitContext`(1).

Name

XmStringPeekNextTriple – retrieve the type of the next component.

Synopsis

XmStringComponentType XmStringPeekNextTriple (XmStringContext *context*)

Inputs

context Specifies the string context for the compound string.

Returns

The type of the next component.

Availability

Motif 2.0 and later.

Description

`XmStringPeekNextTriple`() returns the type of the next component without updating the compound string *context*.

Usage

An XmStringContext is an opaque data type which is used for walking along a compound string one component at a time. It is initialized by a call to XmStringInitContext. Each successive call to `XmStringGetNextComponent`() adjusts the string context to point to the next component. `XmStringPeekNextTriple`() returns the type of the next component without adjusting the *context*, and thus it can be used to look ahead into the compound string.

Structures

The string component type can have one of the following values:

XmSTRING_COMPONENT_CHARSET
XmSTRING_COMPONENT_TEXT
XmSTRING_COMPONENT_LOCALE_TEXT
XmSTRING_COMPONENT_DIRECTION
XmSTRING_COMPONENT_SEPARATOR
XmSTRING_COMPONENT_END
XmSTRING_COMPONENT_UNKNOWN
XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
XmSTRING_COMPONENT_WIDECHAR_TEXT
XmSTRING_COMPONENT_LAYOUT_PUSH
XmSTRING_COMPONENT_LAYOUT_POP
XmSTRING_COMPONENT_RENDITION_BEGIN
XmSTRING_COMPONENT_RENDITION_END

See Also

XmStringFreeContext(1), XmStringGetNextComponent(1), XmStringGetNextSegment(1), XmStringInitContext(1), XmStringPeekNextComponent(1).

Name

XmStringPutRendition – add rendition components to a compound string.

Synopsis

XmString XmStringPutRendition (XmString *string*, XmStringTag *rendition*)

Inputs

string	Specifies a compound string which requires rendition components.
rendition	Specifies a tag used to create the rendition components.

Returns

A newly allocated compound string with rendition components.

Availability

Motif 2.0 and later.

Description

`XmStringPutRendition`() is a convenience function which places XmSTRING_COMPONENT_RENDITION_BEGIN and XmSTRING_COMPONENT_RENDITION_END components containing *rendition* around a compound string. The *string* is not modified by the procedure, which takes a copy.

Usage

`XmStringPutRendition`() allocates space for the returned compound string, and it is the responsibility of the programmer to reclaim the space at an appropriate point by calling `XmStringFree`().

See Also

`XmStringFree`(1).

Name

XmStringSegmentCreate – create a compound string segment.

Synopsis

```
XmString XmStringSegmentCreate ( char                *text,
                                 XmStringCharSet     tag,
                                 XmStringDirection   direction,
                                 Boolean             separator)
```

Inputs

text	Specifies the text component of the compound string segment.
tag	Specifies the font list element tag.
direction	Specifies the value of the direction component. Pass either XmSTRING_DIRECTION_L_TO_R or XmSTRING_DIRECTION_R_TO_L.
separator	Specifies whether or not a separator is added to the compound string.

Returns

A new compound string.

Availability

In Motif 2.0 and later, the function is deprecated. A combination of `XmStringComponentCreate`() and `XmStringConcat`() is preferred.

Description

`XmStringSegmentCreate`() creates a compound string segment that contains the specified *text*, *tag*, and *direction*. If *separator* is True, a separator is added to the segment, following the *text*. If *separator* is False, the compound string segment does not contain a separator. Storage for the returned compound string is allocated by the routine and should be freed by calling `XmStringFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringSegmentCreate`() allows you to create a single segment that can be concatenated with a compound string containing other segments.

See Also

`XmStringCreate`(1), `XmStringFree`(1).

Name

XmStringSeparatorCreate – create a compound string containing a separator component.

Synopsis

XmString XmStringSeparatorCreate (void)

Returns

A new compound string.

Description

`XmStringSeparatorCreate`() creates and returns a compound string containing a separator as its only component.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringSeparatorCreate`() allows you to create a separator component that can be concatenated with a compound string containing other components.

See Also

`XmStringCreate`(1), `XmStringFree`(1), `XmStringSegmentCreate`(1).

Name

XmStringTableParseStringArray – convert an array of strings into a compound string table.

Synopsis

```
XmStringTable XmStringTableParseStringArray (  XtPointer       *strings,
                                               Cardinal        count,
                                               XmStringTag     tag,
                                               XmTextType      type,
                                               XmParseTable
    parse_table,
                                               Cardinal
    parse_count,
                                               XtPointer
    client_data)
```

Inputs

strings	Specifies an array of strings.
count	Specifies the number of items in strings.
tag	Specifies the tag used to create the resulting compound string table.
type	Specifies the type of each input string, and the tag.
parse_table	Specifies a parse table to control the conversion process.
parse_count	Specifies the number of parse mappings in parse_table.
client_data	Specifies application data to pass to parse procedures within the parse_table.

Returns

An array of compound strings.

Availability

Motif 2.0 and later.

Description

`XmStringTableParseStringArray`() converts an array of strings into an array of compound strings. `XmStringTableParseStringArray`() is no more than a convenience function which allocates space for an table of compound strings, and subsequently calls `XmStringParseText`() iteratively on each item within the *strings* array to convert the item into a compound string. Each converted item is placed within the allocated table at a corresponding location to its position in the *strings* array. A *parse_table* can be specified which consists of a set of mappings to control the conversion process. The contents of each of the strings to be converted can be in one of a number of formats: simple characters, multibyte, or wide characters. The *type* parameter specifies the type of

the input strings, and is also used to interpret the *tag* which is used in creating text components within the returned compound string array.

Usage

The function calls `XmStringParseText`() passing NULL as the *text_end* (second) parameter: each item within the array of strings is converted until the occurrence of a terminating null byte. `XmStringTableParseStringArray`() returns allocated storage: the elements within the returned table are compound strings allocated by the internal call to `XmStringParseText`(), and these should each be freed at an appropriate point through `XmStringFree`(). `XmStringTableParseStringArray`() also allocates space for the table itself, and this should subsequently be freed using `XtFree`().

Structures

The XmTextType *type* parameter can take one of the following values:

XmCHARSET_TEXT
XmMULTIBYTE_TEXT
XmWIDECHAR_TEXT

See Also

`XmStringFree`(1), `XmStringGenerate`(1), `XmStringParseText`(1), `XmStringTableUnparse`(1), `XmParseMapping`(2).

Name

XmStringTableProposeTablist – create a tab list for a compound string table.

Synopsis

```
XmTabList XmStringTableProposeTablist (  XmStringTable   strings,
                                         Cardinal        string_count,
                                         Widget          widget,
                                         float           padding,
                                         XmOffsetModel
offset_model)
```

Inputs

strings	Specifies an array of compound strings.
string_count	Specifies the number of items in *strings*.
widget	Specifies a widget from which rendition information is calculated.
padding	Specifies a separation between columns.
offset_model	Specifies whether tabs are created at absolute or relative offsets.

Returns

A new XmTabList.

Availability

Motif 2.0 and later.

Description

`XmStringTableProposeTablist`() creates an XmTabList value which can be used to specify how an array of tabbed compound *strings* is aligned into columns.

A compound string is tabbed if it contains an XmSTRING_COMPONENT_TAB component between textual components: each text component forms an individual column entry. The strings are rendered with respect to a tab list: each tab contains a floating point offset which specifies the starting location of a column. `XmStringTableProposeTablist`() creates a tab list appropriate for laying out the given strings in a multi-column format.

The XmNunitType resource of widget is used to calculate the units in which the tab calculation is performed. Extra spacing between each column is specified by the *padding* parameter, and this is also interpreted in terms of the unit type of *widget*. The *offset_model* determines whether the floating point positions calculated for each tab in the returned XmTabList are at absolute locations (XmABSOLUTE), or relative to the previous tab (XmRELATIVE).

Usage

The tab list created by XmStringTableProposeTablist() can be applied to the render table of the widget where the strings are to be displayed by modifying the XmNtabList resource of an existing rendition through the procedure XmRenditionUpdate(). Alternatively, a new rendition can be created using XmRenditionCreate(), and thereafter merged into the widget render table using XmRenderTableAddRenditions().

XmStringTableProposeTablist() returns allocated storage, and it is the responsibility of the programmer to reclaim the allocated space at a suitable point by calling XmTabListFree().

If no render table is associated with *widget*, XmStringTableProposeTablist() invokes internal routines to deduce a default render table: these routines are not multi-thread safe.

See Also

XmTabCreate(1), XmTabFree(1), XmTabListCopy(1), XmTabListFree(1), XmRenderTableAddRenditions(1), XmRenditionCreate(1), XmRenditionUpdate(1), XmRendition(2).

Name

XmStringTableToXmString – convert compound string table to compound string.

Synopsis

XmString
XmStringTableToXmString (XmStringTable *table*, Cardinal *count*, XmString *break_component*)

Inputs

table	Specifies an array of compound strings.
count	Specified the number of items in the table.
break_component	Specifies a compound string used to separate converted table items.

Returns

A compound string.

Availability

Motif 2.0 and later.

Description

`XmStringTableToXmString`() is a convenience function which converts a *table* of compound strings into a single compound string. A `break_component` can be inserted between each component converted from the *table* in order to separate each.

Usage

`XmStringTableToXmString`() simply walks along the array of items within the *table*, concatenating a copy of each item to the result, along with a copy of the *break_component*. The *break_component* can be NULL, although a component of type XmSTRING_COMPONENT_TAB or XmSTRING_COMPONENT_SEPARATOR is a suitable choice. The function returns allocated storage, and it is the responsibility of the programmer to reclaim the space by calling `XmStringFree`() at a suitable point.

Example

The following code illustrates a basic call to `XmStringTableToXm-String`():

```
extern XmString     table table ;
extern int          table_count ;
XmString            xms;
XmString            break_component;

/* create a break component */
```

```
break_component = XmStringComponentCreate
                                    (XmSTRING_COMPONEN
                                    T_SEPARATOR, 0,
                                    NULL)[1];

/* convert an (unspecified) compound string table */
xms = XmStringTableToXmString (table, table_count, break_component);

/* use the compound string */
...

/* free the allocated space */
XmStringFree (xms);
```

See Also

`XmStringConcat`(1), `XmStringCopy`(1), `XmStringFree`(1), `XmStringToXmStringTable`(1).

1.Erroneously given as XmComponentCreate() in 2nd edition. XmStringSeparatorCreate() would do here equally well.

Name

XmStringTableUnparse – convert a compound string table to an array of strings.

Synopsis

```
XtPointer *XmStringTableUnparse (  XmStringTable   table,
                                   Cardinal        count,
                                   XmStringTag     tag,
                                   XmTextType      tag_type,
                                   XmTextType      output_type,
                                   XmParseTable    parse_table,
                                   Cardinal        parse_count,
                                   XmParseModel    parse_model)
```

Inputs

table	Specifies the compound string table to unparse.
count	Specifies the number of compound strings in table.
tag	Specifies which text segments to unparse.
tag_type	Specifies the type of tag.
output_type	Specifies the type of conversion required.
parse_table	Specifies a parse table to control the conversion.
parse_count	Specifies the number of parse mappings in parse_table.
parse_model	Specifies how non-text components are converted.

Returns

An allocated string array containing the unparsed contents of the compound strings.

Availability

Motif 2.0 and later.

Description

`XmStringTableUnparse`() is a convenience function which unparses an array of compound strings. The XmStringTable *table* is converted into a string array, whose contents is determined by *output_type*, which can be XmCHARSET_TEXT, XmWIDECHAR_TEXT, or XmMULTIBYTE_TEXT. Only those text components within the *table* which match *tag* are converted: NULL converts all text components. An XmParseTable can be supplied which acts as a filter: each parse mapping in the table contains a pattern which must match a text component of the compound string if that component is to be converted.

parse_model determines how non-text components of string are handled when matching against *parse_table*. If the value is XmOUTPUT_ALL, all components are taken into account in the conversion process. If the value is

XmOUTPUT_BETWEEN, any non-text components which are not between two text components are ignored. If the value is XmOUTPUT_BEGINNING, any non-text components which do not precede a text component are ignored. Similarly, the value XmOUTPUT_END specifies that any non-text components which do not follow a text component are ignored. XmOUTPUT_BOTH is a combination of XmOUTPUT_BEGINNING and XmOUTPUT_END. The number of items within the returned string array is identical to the supplied number of compound strings. Each converted string occupies the same index in the returned array as the index of the source compound string in table.

Usage

`XmStringTableUnparse`() is the logical inverse of the routine `XmStringTableParseStringArray`(). It simply invokes `XmStringUnparse`() on each of the elements in the supplied *table*. The function returns allocated storage, both for the returned array, and for each element within the array. It is the responsibility of the programmer to reclaim the storage at an appropriate point by calling `XtFree`() on each element in the array and upon the array itself.

Structures

An XmParseModel has the following possible values:

XmOUTPUT_ALL
XmOUTPUT_BEGINNING
XmOUTPUT_BETWEEN
XmOUTPUT_BOTH
XmOUTPUT_END

See Also

`XmStringFree`(1), `XmStringParseText`(1), `XmStringUnparse`(1), `XmParseMapping`(2).

Name

XmStringToXmStringTable – convert a compound string to a compound string table.

Synopsis

Cardinal XmStringToXmStringTable (XmString *string*, XmString *break_comp*, XmStringTable **table*)

Inputs

string	Specifies a compound string.
break_comp	Specifies a component which indicates where to split string into an individual table element.

Outputs

table	Returns the converted compound string table.

Returns

The number of elements in the converted compound string table.

Availability

Motif 2.0 and later.

Description

`XmStringToXmStringTable`() is a convenience function which converts an XmString *string* into an array of compound strings. *break_comp* is a component which is used to determine how to split the string into individual items: components in string are considered to form contiguous sequences delimited by *break_comp*, each sequence forming a separate entry in the converted string table. Any component from *string* which matches the delimiting *break_comp* is not copied into the converted table. If *break_comp* is NULL, the returned compound string table contains a single entry: a copy of the original string.

Usage

`XmStringToXmStringTable`() is the inverse function to `XmStringTableToXmString`(). A *break_comp* component of type XmSTRING_COMPONENT_TAB or XmSTRING_COMPONENT_SEPARATOR is the most useful choice. The function returns allocated storage, and it is the responsibility of the programmer to reclaim the space by calling `XmStringFree`() on each element in the table, and then `XtFree`() on the table itself.

Example

The following code illustrates a basic call to `XmStringToXmStringTable`():

```
extern XmString     xms;
```

```
Cardinal        count;
XmStringTable   table;
XmString        break_component;
int             i;

/* create a break component */
break_component = XmStringComponentCreate
                                        (XmSTRING_COMPONENT_SEPARATOR, 0,
                                        NULL)[1];

/* convert an (unspecified) compound string */
count = XmStringToXmStringTable (xms, break_component, &table);

/* use the table */
...

/* free the allocated space */
for (i = 0; i < count; i++) {
    XmStringFree (table[i]);
}

XtFree ((char *) table);
```

See Also

`XmStringFree`(1), `XmStringTableToXmString`(1).

1.Erroneously given as XmComponentCreate() in 2nd edition. XmStringSeparatorCreate() could be used equally well.

Name

XmStringUnparse – convert a compound string into a string.

Synopsis

```
XtPointer XmStringUnparse (  XmString       string,
                             XmStringTag    tag,
                             XmTextType     tag_type,
                             XmTextType     output_type,
                             XmParseTable   parse_table,
                             Cardinal       parse_count,
                             XmParseModel   parse_model)
```

Inputs

string	Specifies the compound string to unparse.
tag	Specifies which text segments of string to unparse.
tag_type	Specifies the type of tag.
output_type	Specifies the type of conversion required.
parse_table	Specifies a parse table to control the conversion.
parse_count	Specifies the number of parse mappings in parse_table.
parse_model	Specifies how non-text components are converted.

Returns

An allocated string containing the unparsed contents of a compound string.

Availability

Motif 2.0 and later.

Description

`XmStringUnparse`() is a convenience function which unparses a compound string. The XmString *string* is converted into a string, whose contents is determined by *output_type*, which can be XmCHARSET_TEXT, XmWIDECHAR_TEXT, or XmMULTIBYTE_TEXT. Only those text components within the string which match *tag* are converted: NULL converts all text components. An XmParseTable can be supplied which acts as a filter: each parse mapping in the table contains a pattern which must match a text component of the compound string if that component is to be converted. *parse_model* determines how non-text components of string are handled when matching against *parse_table*. If the value is XmOUTPUT_ALL, all non-text components are used in the conversion process. If the value is XmOUTPUT_BETWEEN, any non-text components which fall between text components are used. If the value is XmOUTPUT_BEGINNING, non-text components which precede a text component are utilized. Similarly, XmOUTPUT_END uses non-text components which follow a text component. XmOUTPUT_BOTH is a combination of XmOUTPUT_BEGINNING and XmOUTPUT_END.

Usage

`XmStringUnparse`() is the logical inverse of the routine `XmStringParse-Text`(). The function returns allocated storage, and it is the responsibility of the programmer to reclaim the storage by calling `XtFree`() at an appropriate point.

Structures

An XmParseModel has the following possible values:

XmOUTPUT_ALL
XmOUTPUT_BEGINNING
XmOUTPUT_BETWEEN
XmOUTPUT_BOTH
XmOUTPUT_END

Example

The following specimen code outlines a basic call to `XmStringUnparse`(), which converts separator and tab components into the output:

```
XmParseTable    parse_table = (XmParseTable) XtCalloc (2, sizeof
(XmParseMapping));
XmString        tmp;
XmString        output;
char            *string;
Arg             av[4];
Cardinal        ac;

/* map tab component to \t */
tmp = XmStringComponentCreate (XmSTRING_COMPONENT_TAB, 0,
NULL);
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\t");         ac++;
parse_table[0] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);

/* map separator component to \n */
tmp = XmStringSeparatorCreate();
ac = 0;
XtSetArg (av[ac], XmNincludeStatus,     XmINSERT);     ac++;
XtSetArg (av[ac], XmNsubstitute,        tmp);          ac++;
XtSetArg (av[ac], XmNpattern,           "\n");         ac++;
parse_table[1] = XmParseMappingCreate (av, ac);
XmStringFree (tmp);
```

```
/* convert the (unspecified) compound string */
string = (char *) XmStringUnparse (xms, NULL, XmCHARSET_TEXT,
                                   XmCHARSET_TEXT, parse_table, 2,
                                   XmOUTPUT_ALL);

/* use the converted string */
....

/* free the allocated space */
XtFree (string);

/* Free the parse table: this also frees the parse mappings */
XmParseTableFree (parse_table, 2);
```

See Also

`XmStringFree`(1), `XmStringParseText`(1), `XmStringTableUnparse`(1), `XmParseMapping`(2).

Name

XmStringWidth – get the width of the longest line of text in a compound string.

Synopsis

Dimension XmStringWidth (XmFontList *fontlist*, XmString *string*)

Inputs

fontlist	Specifies the font list for the compound string.
string	Specifies the compound string.

Returns

The width of the compound string.

Availability

In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable, to which it is an alias.

Description

`XmStringWidth`() returns the width, in pixels, of the longest line of text in the specified compound *string*. Lines in the compound string are delimited by separator components. If *string* is created with `XmStringCreateSimple`(), then fontlist must begin with the font from the character set of the current language environment, otherwise the result is undefined.

Usage

In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. In Motif 2.0 and later, the set of available segments is extended to include, amongst other items, tab, rendition, direction, locale components. `XmStringWidth`() provides information that is useful if you need to render a compound string. Motif widgets render compound strings automatically, so you only need to worry about rendering them yourself if you are writing your own widget. The routine is also useful if you want to get the dimensions of a compound string rendered with a particular font.

See Also

`XmStringBaseline`(1), `XmStringExtent`(1), `XmStringHeight`(1).

Name

XmTabCreate – create a tab stop.

Synopsis

```
XmTab XmTabCreate ( float          value,
                    unsigned char  units,
                    XmOffsetModel  offset_model,
                    unsigned char  alignment,
                    char           *decimal)
```

Inputs

value Specifies the value to be used in calculating the location of a tab stop.

units Specifies the units in which value is expressed.

offset_model Specifies whether the tab is at an absolute position, or relative to the previous tab.

alignment Specifies how text should be aligned to this tab stop.

decimal Specifies the multibyte character in the current locale to be used as a decimal point.

Returns

An XmTab object.

Availability

Motif 2.0 and later.

Description

`XmTabCreate`() creates a tab stop at the position specified by *value*, which is expressed in terms of *units*. If value is less than 0 (zero), a warning is displayed, and *value* is reset to 0.0. The *offset_model* determines whether the tab position is an absolute location (XmABSOLUTE) calculated from the start of any rendering, or relative to the previous tab stop (XmRELATIVE). *alignment* specifies how text is aligned with respect to the tab location: at present only XmALIGNMENT_BEGINNING is supported. *decimal* is a multibyte character which describes the decimal point character in the current locale.

Usage

A tab stop can be created, and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget. The *decimal* parameter is currently unused.

`XmTabCreate`() allocates storage for the object which it returns. It is the responsibility of the programmer to free the memory at an appropriate point by a call to `XmTabFree`().

Structures

The XmOffsetModel type has the following possible values:

XmABSOLUTE	XmRELATIVE

Valid values for the units parameter are:

XmPIXELS	
Xm100TH_MILLIMETERS	XmMILLIMETERS
Xm1000TH_INCHES	XmINCHES
Xm100TH_FONT_UNITS	XmFONT_UNITS
Xm100TH_POINTS	XmPOINTS
XmCENTIMETERS	

Example

The following code creates a multi-column arrangement of compound strings; it creates a set of XmTab objects, which are then inserted into an XmRendition object. The XmRendition object is added to a render table:

```
extern Widget widget;
int            i;
Arg            argv[3];
XmTab          tabs[MAX_COLUMNS];
XmTabList      tab_list = (XmTabList) 0;
XmRendition    rendition;
XmRenderTable  render_table;

/* Create the XmTab objects */
for (i = 0 ; i < MAX_COLUMNS ; i++) {
   tabs[i] = XmTabCreate ((float) 1.5,
                          XmINCHES,
                          ((i == 0) ? XmABSOLUTE :
                          XmRELATIVE),
                          XmALIGNMENT_BEGINNING,
                          ".");
}

/* Add them to an XmTabList */
tab_list = XmTabListInsertTabs (NULL, tabs,
MAX_COLUMNS, 0);

/* Put the XmTabList into an XmRendition object */
```

```
i = 0;
XtSetArg (argv[i], XmNtabList, tab_list); i++;
XtSetArg (argv[i], XmNfontName, “fixed”); i++;
XtSetArg (argv[i], XmNfontType, XmFONT_IS_FONT);
i++;
rendition = XmRenditionCreate (widget,NULL,argv,
i);

/* Add the XmRendition object into an XmRenderTable
*/
render_table = XmRenderTableAddRenditions (NULL,
                                           &rendi-
                                           tion,
                                           1,
                                           XmMERGE
                                           _NEW);

/* Apply to the widget */
XtVaSetValues (widget, XmNrenderTable,
render_table, NULL);
...

/* Free Up - render table applied to widget takes a
reference
** counted copy of everything
*/
for (i = 0 ; i < MAX_COLUMNS ; i++) {
   XmTabFree (tabs[i]);
}
XmTabListFree (tab_list);
XmRenditionFree (rendition);
XmRenderTableFree (render_table);
```

See Also

XmTabFree(1), XmTabGetValues(1), XmTabListInsertTabs(1), XmTabSetValue(1), XmRenditionCreate(1), XmRenderTableAddRenditions(1), XmRendition(2).

Name

XmTabFree – free the memory used by an XmTab object.

Synopsis

void XmTabFree (XmTab *tab*)

Inputs

tab Specifies an XmTab object.

Availability

Motif 2.0 and later.

Description

`XmTabFree`() reclaims the memory associated with tab, previously allocated by a call to `XmTabCreate`().

Usage

A tab stop can be created using `XmTabCreate`(), and inserted into an XmTab-List through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

See Also

`XmTabCreate`(1), `XmRendition`(2).

Name

XmTabGetValues – fetch the value of an XmTab object.

Synopsis

```
float XmTabGetValues (  XmTab           tab,
                        unsigned char   *units,
                        XmOffsetModel   *offset_model,
                        unsigned char   *alignment,
                        char            **decimal)
```

Inputs

tab	Specifies an XmTab object.

Outputs

units	Returns the units in which the tab stop is calculated.
offset_model	Returns the offset model.
alignment	Returns the text alignment with respect to the tab stop.
decimal	Returns the multibyte character used to represent the decimal point.

Returns

The distance, expressed in units, which the tab is offset.

Availability

Motif 2.0 and later.

Description

`XmTabGetValues`() retrieves the data associated with a tab stop object. The function returns the position value of a tab stop, which is expressed in terms of units. The *offset_model* determines whether the *tab* position is an absolute location (XmABSOLUTE) calculated from the start of any rendering, or relative to the previous tab stop (XmRELATIVE). *alignment* specifies how text is aligned with respect to the tab location: at present only XmALIGNMENT_BEGINNING is supported. *decimal* is a multibyte character which describes the decimal point character in the current locale.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget. The decimal value is currently unused.

Structures

The XmOffsetModel type has the following possible values:

XmABSOLUTE	XmRELATIVE

Valid values for units are:

XmPIXELS	
Xm100TH_MILLIMETERS	XmMILLIMETERS
Xm1000TH_INCHES	XmINCHES
Xm100TH_FONT_UNITS	XmFONT_UNITS
Xm100TH_POINTS	XmPOINTS
XmCENTIMETERS	

See Also

`XmTabCreate`(1), `XmTabSetValue`(1), `XmRendition`(2).

Name

XmTabListCopy – copy an XmTabList object.

Synopsis

XmTabList XmTabListCopy (XmTabList *tab_list*, int *offset*, Cardinal *count*)[1]

Inputs

tab_list	Specifies the tab list to copy.
offset	Specifies an index into the tab list from which to start copying.
count	Specifies the number of tab stops to copy.

Returns

A new tab list containing tabs copied from *tab_list*.

Availability

Motif 2.0 and later.

Description

`XmTabListCopy`() is a convenience function which creates a new XmTabList by copying portions of an existing *tab_list*. If *tab_list* is NULL, the function simply returns NULL. Otherwise, a new tab list is allocated, and selected tabs are copied, depending on the *offset* and *count* parameters. *count* specifies the number of tabs to copy, and *offset* determines where in the original list to start. If *offset* is zero, tabs are copied from the beginning of tab_list. If *offset* is negative, tabs are copied in reverse order from the end of *tab_list*. If *count* is zero, all tabs from *offset* to the end of *tab_list* are copied.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

`XmTabListCopy`() allocates storage for the object which it returns. It is the responsibility of the programmer to free the memory at an appropriate point by a call to `XmTabListFree`().

See Also

`XmTabCreate`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

1.Erroneously given as XmTabListTabCopy() in 2nd edition.

Name

XmTabListFree – free the memory used by a tab list.

Synopsis

void XmTabListFree (XmTabList *tab_list*)

Inputs

tab_list Specified the tab list to free.

Availability

Motif 2.0 and later.

Description

`XmTabListFree`() reclaims the space used by an XmTabList object, *tab_list*.

Usage

In common with other objects in Motif 2.0 and later, the tab (XmTab) and tab list (XmTabList) are dynamically allocated data structures which must be freed when no longer required. For example, `XmStringTableProposeTablist`() dynamically creates a tab list for rendering a multi-column compound string table which must be subsequently deallocated.

It is important to call `XmTabListFree`() instead of `XtFree`() because `XmTabListFree`() also reclaims storage for each of the elements in the tab list.

See Also

`XmStringTableProposeTablist`(1), `XmTabCreate`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListGetTab – retrieve a tab from a tab list

Synopsis

XmTab XmTabListGetTab (XmTabList *tab_list*, Cardinal *position*)

Inputs

tab_list	Specifies a tab list.
position	Specifies the position of the required tab within the tab_list.

Returns

A copy of the required tab.

Availability

Motif 2.0 and later.

Description

`XmTabListGetTab`() returns a copy of a tab from the XmTabList specified by *tab_list*. *position* determines where in the list of tabs the required tab is copied from. The first tab within *tab_list* is at *position* zero, the second tab at *position* 1, and so on. If *position* is not less than the number of tabs within the list, `XmTab-ListGetTab`() returns NULL.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

`XmTabListGetTab`() returns a copy of the tab within the original tab list, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabFree`().

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListInsertTabs – insert tabs into a tab list.

Synopsis

XmTabList XmTabListInsertTabs (XmTabList *tab_list*, XmTab **tabs*, Cardinal *tab_count*, int *position*)

Inputs

tab_list	Specifies the tab list into which tabs are added.
tabs	Specifies an array of tabs to add.
tab_count	Specifies the number of tabs within the tabs array.
position	Specifies an index into tab_list at which to insert the tabs.

Returns

A newly allocated tab list.

Availability

Motif 2.0 and later.

Description

`XmTabListInsertTabs`() creates a new tab list by merging a set of *tabs* into a *tab_list* at a given *position*. If *tabs* is NULL, or *tab_count* is zero, the function returns the original *tab_list*. If *position* is zero, the new tabs are inserted at the head of the new tab list, if *position* is 1, they are inserted after the first tab, and so forth. If *position* is greater than the number of tabs in *tab_list*, the new tabs are appended. If *position* is negative, the new tabs are inserted in reverse order at the end of the original list, so that the first new tab becomes the last tab in the new list.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

`XmTabListInsertTabs`() returns allocated storage, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabListFree`().

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListRemoveTabs – copy a tab list, excluding specified tabs.

Synopsis

```
XmTabList XmTabListRemoveTabs ( XmTabList   old_list,
                                Cardinal    *position_list,
                                Cardinal    position_count)
```

Inputs

old_list	Specifies the tab list to copy.
position_list	Specifies an array of tab positions to exclude.
position_count	Specifies the length of position_list.

Returns

A copy of *old_list*, with the specified positions excluded.

Availability

Motif 2.0 and later.

Description

`XmTabListRemoveTabs`() removes tabs from a *old_list* by allocating a new tab list which contains all the tabs of the original list except for those at specific positions within a *position_list*. The first tab within a tab list is at position zero, the second tab at position 1, and so on. If *old_list* is NULL, or if *position_list* is NULL, or if *position_count* is zero, the function returns *old_list* unmodified. Otherwise *old_list* is freed, as are any excluded tab elements, before the newly allocated tab list is returned.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

When the returned tab list differs from the original old_list parameter, `XmTabListRemoveTabs`() returns allocated storage, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabListFree`().

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTabListReplacePositions – copy a tab list, replacing tabs at specified positions.

Synopsis

```
XmTabList XmTabListReplacePositions ( XmTabList    old_list,
                                      Cardinal     *position_list,
                                      XmTab        *tabs,
                                      Cardinal     tab_count)
```

Inputs

old_list	Specifies the tab list to modify.
position_list	Specifies an array of positions at which to replace the tabs.
tabs	Specifies the tabs which replace those in old_list.
tab_count	Specifies the number of tabs and positions.

Returns

A new tab list with tabs replaced at specified positions.

Availability

Motif 2.0 and later.

Description

`XmTabListReplacePositions`() creates a newly allocated tab list which contains all the original tabs in *old_list*, except that at any position contained within *position_list*, the corresponding tab from tabs is used instead of the original. That is, at the first position specified within *position_list*, the original tab is replaced by the first tab within *tabs*. The first tab within a tab list is at position zero, the second tab at position 1, and so on. If *old_list* is NULL, or if *tabs* is NULL, or if *position_list* is NULL, or if *tab_count* is zero, the function returns *old_list* unmodified. Otherwise *old_list* is freed, as are any replaced tab elements, before the newly allocated tab list is returned.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

When the returned tab list differs from the original old_list parameter, `XmTabListReplacePositions`() returns allocated storage, and it is the responsibility of the programmer to reclaim the space at a suitable point by calling `XmTabListFree`()

See Also

XmTabCreate(1), XmTabFree(1), XmTabListFree(1), XmTabListInsertTabs(1), XmTabListRemoveTabs(1), XmRendition(2).

Name

XmTabListTabCount – count the number of tabs in a tab list.

Synopsis

Cardinal XmTabListTabCount (XmTabList *tab_list*)

Inputs

tab_list The tab list to count.

Returns

The number of tab stops in tab_list.

Availability

Motif 2.0 and later.

Description

`XmTabListTabCount`()[1] is a convenience function which counts the number of XmTab objects contained within *tab_list*. If *tab_list* is NULL, the function returns zero.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

See Also

`XmTabCreate`(1), `XmTabFree`(1), `XmTabListFree`(1), `XmTabListInsertTabs`(1), `XmTabListRemoveTabs`(1), `XmRendition`*(s2.

1.Erroneously given as XmTabListCount() in 2nd edition.

Name

XmTabSetValue – set the value of a tab stop.

Synopsis

void XmTabSetValue (XmTab *tab*, float *value*)

Inputs

tab	Specifies the tab to modify.
value	Specifies the new value for the tab stop.

Availability

Motif 2.0 and later.

Description

`XmTabSetValue`() sets the value associated with an XmTab object. The *value* is a floating point quantity which either represents an absolute distance from the start of the current rendition, or a value relative to the previous tab stop. The offset model specified when the tab is created determines whether value is relative or absolute. If *value* is less than 0 (zero), a warning message is displayed and the function returns without modifying the tab.

Usage

A tab stop can be created using the function `XmTabCreate`(), and inserted into an XmTabList through the function `XmTabListInsertTabs`(). The tab list is used to render a multi-column layout of compound strings by specifying the list as the XmNtabList resource of a rendition object within a render table associated with a widget.

See Also

`XmTabCreate`(1), `XmTabListInsertTabs`(1), `XmRendition`(2).

Name

XmTargetsAreCompatible – determine whether or not the target types of a drag source and a drop site match.

Synopsis

```
#include <Xm/DragDrop.h>

Boolean XmTargetsAreCompatible (  Display     *display,
                                  Atom        *export_targets,
                                  Cardinal    num_export_targets,
                                  Atom        *import_targets,
                                  Cardinal    num_import_targets)
```

Inputs

display	Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().
export_targets	Specifies the list of target atoms to which the drag source can convert the data.
num_export_targets	Specifies the number of items in export_targets.
import_targets	Specifies the list of target atoms that are accepted by the drop site.
num_import_targets	Specifies the number of items in import_targets.

Returns

True if there is a compatible target or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmTargetsAreCompatible`() determines whether or not the import targets of a drop site match any of the export targets of a drag source. The routine returns True if the two objects have at least one target in common; otherwise it returns False.

Usage

Motif 1.2 and later supports the drag and drop model of selection actions. In a widget that acts as a drag source, a user can make a selection and then drag the selection, using BTransfer, to other widgets that are registered as drop sites. These drop sites can be in the same application or another application. In order for a drag and drop operation to succeed, the drag source and the drop site must both be able to handle data in the same format. `XmTargetsAreCompatible`() provides a way for an application to check if a drag source and a drop site support compatible formats.

See Also

`XmDragContext`(1), `XmDropSite`(1).

Name

XmTextClearSelection, XmTextFieldClearSelection – clear the primary selection.

Synopsis

```
#include <Xm/Text.h>

void XmTextClearSelection (Widget widget, Time time)

#include <Xm/TextF.h>

void XmTextFieldClearSelection (Widget widget, Time time)
```

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Description

`XmTextClearSelection`() and `XmTextFieldClearSelection`() clear the primary selection in the specified *widget*. `XmTextClearSelection`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldClearSelection`() only works for a TextField widget. For each routine, *time* specifies the server time of the event that caused the request to clear the selection.

Usage

`XmTextClearSelection`() and `XmTextFieldClearSelection`() provide a convenient way to deselect the text selection in a Text or TextField widget. If no text is selected, the routines do nothing. Any text that is stored in the clipboard or selection properties remains; the routines affect the selected text in the widget only. If you are calling one of these routines from a callback routine, you probably want to use the time field from the event pointer in the callback structure as the value of the *time* parameter. You can also use the value CurrentTime, but there is no guarantee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, and **Clear**) shows the use of XmTextClearSelection()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                 num = (int) client_data;
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *) call_data;
```

```
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbutton.time);
        break;
        case 3:   result = XmTextPaste (text_w);break
        case 4:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

`XmTextCopy`(1), `XmTextCopyLink`(1), `XmTextCut`(1), `XmTextGetSelection`(1), `XmTextGetSelectionPosition`(1), `XmTextGetSelectionWcs`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextCopy, XmTextFieldCopy – copy the primary selection to the clipboard.

Synopsis

#include <Xm/Text.h>

Boolean XmTextCopy (Widget *widget*, Time *time*)

#include <Xm/TextF.h>

Boolean XmTextFieldCopy (Widget *widget*, Time *time*)

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Returns

True on success or False otherwise.

Description

`XmTextCopy`() and `XmTextFieldCopy`() copy the primary selection in the specified *widget* to the clipboard. `XmTextCopy`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldCopy`() only works for a TextField widget. For each routine, *time* specifies the server time of the event that caused the request to copy the selection. Both routines return True if successful. If the primary selection is NULL, if it is not owned by the specified widget, or if the function cannot obtain ownership of the clipboard selection, the routines return False.

In Motif 2.0 and later, `XmTextCopy`() interfaces with the Uniform Transfer Model by indirectly invoking the XmNconvertCallback procedures of the widget.

Usage

`XmTextCopy`() and `XmTextFieldCopy`() copy the text that is selected in a Text or TextField widget and place it on the clipboard. If you are calling one of these routines from a callback routine, you probably want to use the *time* field from the event pointer in the callback structure as the value of the time parameter. You can also use the value CurrentTime, but there is no guarantee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of `XmTextCopy`()**:**

```
Widget text_w, status;
```

```
void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                     num = (int) client_data;
    XmAnyCallbackStruct*cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbutton.time);
        break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

`XmTextClearSelection`(1), `XmTextCopyLink`(1), `XmTextCut`(1), `XmTextGetSelection`(1), `XmTextGetSelectionWcs`(1), `XmTextPaste`(1), `XmTextPasteLink`(1), `XmTextRemove`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextCopyLink, XmTextFieldCopyLink – copy the primary selection to the clipboard.

Synopsis

#include <Xm/Text.h>

Boolean XmTextCopyLink (Widget *widget*, Time *time*)

#include <Xm/TextF.h>

Boolean XmTextFieldCopyLink (Widget *widget*, Time *time*)

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Returns

True on success or False otherwise.

Availability

Motif 2.0 and later.

Description

`XmTextCopyLink`() and `XmTextFieldCopyLink`() copy a link to the primary selection in the specified *widget* to the clipboard. `XmTextCopyLink`() works when widget is a Text widget or a TextField widget, while `XmTextFieldCopyLink`() only works for a TextField widget. For each routine, *time* specifies the server time of the event that caused the request to copy the selection. Both routines return True if successful. If the primary selection is NULL, if it is not owned by the specified widget, or if the function cannot obtain ownership of the clipboard selection, the routines return False.

`XmTextCopyLink`() and `XmTextFieldCopyLink`() interface with the Uniform Transfer Model by indirectly invoking XmNconvertCallback procedures for the widget. The Text widget itself does not copy links: convert procedures which the programmer provides are responsible for copying the link to the clipboard.

Usage

`XmTextCopyLink`() and `XmTextFieldCopyLink`() copy links to the text that is selected in a Text or TextField widget and place it on the clipboard. If you are calling one of these routines from a callback routine, you probably want to use the time field from the event pointer in the callback structure as the value of the *time* parameter. You can also use the value CurrentTime, but there is no guarantee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of `XmTextCopyLink`()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
    {
    int                     num = (int) client_data;
    XmAnyCallbackStruct     *cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbutton.time);
        break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

`XmTextClearSelection`(1), `XmTextCopy`(1), `XmTextCut`(1), `XmTextGetSelection`(1), `XmTextGetSelectionWcs`(1), `XmTextPaste`(1), `XmTextPasteLink`(1), `XmTextRemove`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextCut, XmTextFieldCut – copy the primary selection to the clipboard and remove the selected text.

Synopsis

#include <Xm/Text.h>

Boolean XmTextCut (Widget *widget*, Time *time*)

#include <Xm/TextF.h>

Boolean XmTextFieldCut (Widget *widget*, Time *time*)

Inputs

widget	Specifies the Text or TextField widget.
time	Specifies the time of the event that caused the request.

Returns

True on success or False otherwise.

Description

`XmTextCut`() and `XmTextFieldCut`() copy the primary selection in the specified *widget* to the clipboard and then delete the primary selection. `XmTextCut`() works when widget is a Text widget or a TextField widget, while `XmTextFieldCut`() only works for a TextField widget. For each routine, *time* specifies the server time of the event that caused the request to cut the selection. Both routines return True if successful. If the widget is not editable, if the primary selection is NULL or if it is not owned by the specified widget, or if the function cannot obtain ownership of the clipboard selection, the routines return False.

`XmTextCut`() and `XmTextFieldCut`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

In Motif 2.0 and later, `XmTextCut`() interfaces with the Uniform Transfer Model by indirectly invoking the XmNconvertCallback procedures of the widget, firstly to transfer the selection to the clipboard, and secondly to delete the selection.

Usage

`XmTextCut`() and `XmTextFieldCut`() copy the text that is selected in a Text or TextField widget, place it on the clipboard, and then delete the selected text. If you are calling one of these routines from a callback routine, you probably want

to use the time field from the event pointer in the callback structure as the value of the *time* parameter. You can also use the value CurrentTime, but there is no guarantee that using this time prevents race conditions between multiple clients that are trying to use the clipboard.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, and **PasteLink**, **Clear**) shows the use of `XmTextCut`**():**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                 num = (int) client_data;
    XmAnyCallbackStruct*cbs = (XmAnyCallbackStruct *) call_data;
    Boolean             result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbutton.time);
        break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

`XmTextClearSelection`(1), `XmTextCopy`(1), `XmTextCopyLink`(1), `XmTextGetSelection`(1), `XmTextGetSelectionWcs`(1), `XmTextPaste`(1), `XmTextPasteLink`(1), `XmTextRemove`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextDisableRedisplay – prevent visual update of a Text widget.

Synopsis

#include <Xm/Text.h>

void XmTextDisableRedisplay (Widget *widget*)

Inputs

widget Specifies the Text widget.

Availability

Motif 1.2 and later.

Description

`XmTextDisableRedisplay`() temporarily inhibits visual update of the specified Text *widget*. Even if the visual attributes of the widget have been modified, the appearance remains unchanged until `XmTextEnableRedisplay`() is called.

Usage

`XmTextDisableRedisplay`() and `XmTextEnableRedisplay`() allow an application to make multiple changes to a Text widget without immediate visual updates. When multiple changes are made with redisplay enabled, visual flashing often occurs. These routines eliminate this problem.

See Also

`XmTextEnableRedisplay`(1), `XmUpdateDisplay`(1), `XmText`(2).

Name

XmTextEnableRedisplay – allow visual update of a Text widget.

Synopsis

#include <Xm/Text.h>

void XmTextEnableRedisplay (Widget *widget*)

Inputs

widget Specifies the Text widget.

Availability

Motif 1.2 and later.

Description

`XmTextEnableRedisplay`() allows the specified Text *widget* to update its visual appearance. This routine is used in conjunction with `XmTextDisable-Redisplay`(), which prevents visual update of the Text widget. When `XmTex-tEnableRedisplay`() is called, the widget modifies its visuals to reflect all of the changes since the last call to `XmTextDisableRedisplay`(). All future changes that affect the visual appearance are displayed immediately.

Usage

`XmTextDisableRedisplay`() and `XmTextEnableRedisplay`() allow an application to make multiple changes to a Text widget without immediate visual updates. When multiple changes are made with redisplay enabled, visual flashing often occurs. These routines eliminate this problem.

See Also

`XmTextDisableRedisplay`(1), `XmUpdateDisplay`(1), `XmText\*`(s2.

Name

XmTextFindString – find the beginning position of a text string.

Synopsis

```
#include <Xm/Xm.h>

Boolean XmTextFindString (  Widget          widget,
                            XmTextPosition  start,
                            char            *string,
                            XmTextDirection direction,
                            XmTextPosition  *position)
```

Inputs

widget	Specifies the Text widget.
start	Specifies the position from which the search begins.
string	Specifies the string for which to search.
direction	Specifies the direction of the search. Pass either XmTEXT_FORWARD or XmTEXT_BACKWARD.

Outputs

position	Returns the position where the search string starts.

Returns

True if the string is found or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmTextFindString`() finds the beginning position of the specified *string* in the Text widget. Depending on the value of *direction*, the routine searches forward or backward from the specified *start* position for the first occurrence of *string*. If `XmTextFindString`() finds a match, it returns True and *position* specifies the position of the first character of the string as the number of characters from the beginning of the text, where the first character position is 0 (zero). If a match is not found, the routine returns False and the value of *position* is undefined.

Usage

`XmTextFindString`() is a convenience routine that searches the text in a Text widget for a particular *string*. Without the routine, the search must be performed using the standard string manipulation routines.

Example

The following routine shows the use of `XmTextFindString`() to locate a string in a text editing window. The search string is specified by the user in a single-line Text widget:

```
Widget text_w, search_w;

void search_text (void)
{
    char            *search_pat = (char *) 0;
    XmTextPosition  pos, search_pos;
    Boolean         found = False;

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    /* find next occurrence from current position -- wrap if necessary */
    pos = XmTextGetCursorPosition (text_w);
    found = XmTextFindString (text_w, pos, search_pat,
    XmTEXT_FORWARD, &search_pos);

    if (!found)
        found = XmTextFindString (text_w, 0, search_pat,
        XmTEXT_FORWARD, &search_pos);

    if (found)
        XmTextSetInsertionPosition (text_w, search_pos);

    XtFree (search_pat);
}
```

See Also

`XmTextFindStringWcs`(1), `XmTextGetSubstring`(1), `XmTextGetSubstringWcs`(1), `XmText`(2).

Name

XmTextFindStringWcs – find the beginning position of a wide-character string in a Text widget.

Synopsis

```
#include <Xm/Text.h>

Boolean XmTextFindStringWcs (   Widget          widget,
                                XmTextPosition  start,
                                wchar_t         *wcstring,
                                XmTextDirection direction,
                                XmTextPosition  *position)
```

Inputs

widget	Specifies the Text widget.
start	Specifies the position from which the search begins.
wcstring	Specifies the wide-character string for which to search.
direction	Specifies the direction of the search. Pass either XmTEXT_FORWARD or XmTEXT_BACKWARD.

Outputs

position	Returns the position where the search string starts.

Returns

True if the string is found or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmTextFindStringWcs`() finds the beginning position of the specified wide-character *wcstring* in the Text *widget*. Depending on the value of *direction*, the routine searches forward or backward from the specified *start* position for the first occurrence of *wcstring*. If `XmTextFindStringWcs`() finds a match, it returns True and *position* specifies the position of the first character of the string as the number of characters from the beginning of the text, where the first character position is 0 (zero). If a match is not found, the routine returns False and the value of *position* is undefined.

Usage

In Motif 1.2, the Text widget supports wide-character strings. `XmTextFind-StringWcs`() is a convenience routine that searches the text in a Text widget for a particular wide-character string. The routine converts the wide-character string into a multi-byte string and then performs the search. Without the routine, the search must be performed using the standard string manipulation routines.

See Also

XmTextFindString(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmText(2).

Name

XmTextGetBaseline, XmTextFieldGetBaseline – get the position of the baseline.

Synopsis

```
#include <Xm/Text.h>

int XmTextGetBaseline (Widget widget)

#include <Xm/TextF.h>

int XmTextFieldGetBaseline (Widget widget)
```

Inputs

widget Specifies the Text or TextField widget.

Returns

The baseline position.

Description

`XmTextGetBaseline`() returns the y coordinate of the baseline of the first line of text in the specified Text *widget*, while `XmTextFieldGetBaseline`() returns the y coordinate of the baseline for the text in the specified TextField *widget*. `XmTextGetBaseline`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetBaseline`() only works for a TextField widget. For each routine, the returned value is relative to the top of the *widget* and it accounts for the margin height, shadow thickness, highlight thickness, and font ascent of the first font in the font list.

Usage

`XmTextGetBaseline`() and `XmTextFieldGetBaseline`() provide information that is useful when you are laying out an application and trying to align different components.

See Also

`XmTextGetCenterline`(1), `XmWidgetGetBaselines`(1), `XmWidgetGetDisplayRect`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetCenterline – get the height of vertical text.

Synopsis

#include <Xm/Text.h>

int XmTextGetCenterline (Widget *widget*)

Inputs

widget Specifies the Text widget.

Returns

The center line x position.

Availability

Motif 2.1 and later.

Description

`XmTextGetCenterline`() calculates the x coordinate of the centerline in a Text widget containing vertical text. If the layout direction of the Text widget does not match XmTOP_TO_BOTTOM_RIGHT_TO_LEFT the function returns zero. Otherwise the procedure calculates the x position of the centerline relative to the left of the Text. The margin width, shadow thickness, and the width of the font are taken into consideration when performing the calculation.

Usage

`XmTextGetCenterline`() provides information that is useful when you are laying out an application and trying to align different components.

See Also

`XmTextGetBaseline`(1), `XmWidgetGetCenterlines`(1), `XmWidgetGetDisplayRect`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetCursorPosition, XmTextFieldGetCursorPosition – get the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetCursorPosition (Widget *widget*)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetCursorPosition (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The value of the XmNcursorPosition resource.

Description

`XmTextGetCursorPosition`() and `XmTextFieldGetCursorPosition`() return the value of the XmNcursorPosition resource for the specified *widget*. `XmTextGetCursorPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetCursorPosition`() only works for a TextField widget. For each routine, the value specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

Usage

`XmTextGetCursorPosition`() and `XmTextFieldGetCursorPosition`() are convenience routines that return the value of the XmNcursorPosition resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextGetInsertionPosition`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetEditable, XmTextFieldGetEditable – get the edit permission state.

Synopsis

#include <Xm/Text.h>

Boolean XmTextGetEditable (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldGetEditable (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The state of the XmNeditable resource.

Description

`XmTextGetEditable`() and `XmTextFieldGetEditable`() return the value of the XmNeditable resource for the specified Text or TextField *widget*. `XmTextGetEditable`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetEditable`() only works for a TextField widget.

Usage

By default, the XmNeditable resource is True, which means that a user can edit the text string. Setting the resource to False makes a text area read-only. `XmTextGetEditable`() and `XmTextFieldGetEditable`() are convenience routines that return the value of the XmNeditable resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextSetEditable`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetInsertionPosition, XmTextFieldGetInsertionPosition – get the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetInsertionPosition (Widget *widget*)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetInsertionPosition (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

The value of the XmNcursorPosition resource.

Description

The functions, `XmTextGetInsertionPosition`() and `XmTextFieldGetInsertionPosition`(), return the value of the XmNcursorPosition resource for the specified *widget*. `XmTextGetInsertionPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetInsertionPosition`() only works for a TextField widget. For each routine, the value specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

Usage

The functions, `XmTextGetInsertionPosition`() and `XmTextFieldGetInsertionPosition`(), are convenience routines that return the value of the XmNcursorPosition resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetCursorPosition`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetLastPosition, XmTextFieldGetLastPosition – get the position of the last character of text.

Synopsis

```
#include <Xm/Text.h>

XmTextPosition XmTextGetLastPosition (Widget widget)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldGetLastPosition (Widget widget)
```

Inputs

widget Specifies the Text or TextField widget.

Returns

The position of the last text character.

Description

`XmTextGetLastPosition`() and `XmTextFieldGetLastPosition`() return the position of the last character of text in the specified *widget*. `XmText-GetLastPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetLastPosition`() only works for a Text-Field widget. For each routine, the returned value specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero).

Usage

`XmTextGetLastPosition`() and `XmTextFieldGetLastPosition`() are convenience routines that return the number of characters of text in a Text or TextField widget.

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetTopCharacter`(1), `XmTextScroll`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetTopCharacter`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetMaxLength, XmTextFieldGetMaxLength – get the maximum possible length of a text string.

Synopsis

```
#include <Xm/Text.h>

int XmTextGetMaxLength (Widget widget)

#include <Xm/TextF.h>

int XmTextFieldGetMaxLength (Widget widget)
```

Inputs

widget Specifies the Text or TextField widget.

Returns

The value of the XmNmaxLength resource.

Description

`XmTextGetMaxLength`() and `XmTextFieldGetMaxLength`() return the value of the XmNmaxLength resource for the specified Text or TextField *widget*. `XmTextGetMaxLength`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetMaxLength`() only works for a TextField widget. For each routine, the returned value specifies the maximum allowable length of a text string that a user can enter from the keyboard.

Usage

`XmTextGetMaxLength`() and `XmTextFieldGetMaxLength`() are convenience routines that return the value of the XmNmaxLength resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextSetMaxLength`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSelection, XmTextFieldGetSelection – get the value of the primary selection.

Synopsis

#include <Xm/Text.h>

char * XmTextGetSelection (Widget *widget*)

#include <Xm/TextF.h>

char * XmTextFieldGetSelection (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

A string containing the primary selection.

Description

`XmTextGetSelection`() and `XmTextFieldGetSelection`() return a pointer to a character string containing the primary selection in the specified *widget*. `XmTextGetSelection`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSelection`() only works for a TextField widget. For each routine, if no text is selected in the widget, the returned value is NULL. Storage for the returned string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

`XmTextGetSelection`() and `XmTextFieldGetSelection`() provide a convenient way to get the current selection from a Text or TextField widget.

See Also

`XmTextGetSelectionPosition`(1), `XmTextGetSelectionWcs`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSelectionPosition, XmTextFieldGetSelectionPosition – get the position of the primary selection.

Synopsis

```
#include <Xm/Text.h>

Boolean XmTextGetSelectionPosition (    Widget          widget,
                                        XmTextPosition  *left,
                                        XmTextPosition  *right)

#include <Xm/TextF.h>

Boolean XmTextFieldGetSelectionPosition (   Widget          widget,
                                            XmTextPosition  *left,
                                            XmTextPosition  *right)
```

Inputs

widget Specifies the Text or TextField widget.

Outputs

left Returns the position of the left boundary of the primary selection.

right Returns the position of the right boundary of the primary selection.

Returns

True if widget owns the primary selection or False otherwise.

Description

The functions, `XmTextGetSelectionPosition`() and `XmTextFieldGetSelectionPosition`() return the *left* and *right* boundaries of the primary selection for the specified *widget*. `XmTextGetSelectionPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSelectionPosition`() only works for a TextField widget. Each boundary value specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). Each routine returns True if the specified Text or TextField *widget* owns the primary selection; otherwise, the routine returns False and the values of *left* and *right* are undefined.

Usage

The functions, `XmTextGetSelectionPosition`() and `XmTextFieldGetSelectionPosition`(), provide a convenient way to get the position of the current selection from a Text or TextField widget.

See Also

`XmTextGetSelection`(1), `XmTextGetSelectionWcs`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSelectionWcs, XmTextFieldGetSelectionWcs – get the wide-character value of the primary selection.

Synopsis

```
#include <Xm/Text.h>

wchar_t * XmTextGetSelectionWcs (Widget widget)

#include <Xm/TextF.h>

wchar_t * XmTextFieldGetSelectionWcs (Widget widget)
```

Inputs

widget Specifies the Text or TextField widget.

Returns

A wide-character string containing the primary selection.

Availability

Motif 1.2 and later.

Description

`XmTextGetSelectionWcs`() and `XmTextFieldGetSelectionWcs`() return a pointer to a wide-character string containing the primary selection in the specified *widget*. `XmTextGetSelectionWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSelectionWcs`() only works for a TextField widget. For each routine, if no text is selected in the widget, the returned value is NULL. Storage for the returned wide-character string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. `XmTextGetSelectionWcs`() and `XmTextFieldGetSelectionWcs`() provide a convenient way to get the current selection in wide-character format from a Text or TextField widget.

See Also

`XmTextGetSelection`(1), `XmTextGetSelectionPosition`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSource – get the text source.

Synopsis

#include <Xm/Text.h>

XmTextSource XmTextGetSource (Widget *widget*)

Inputs

widget Specifies the Text widget.

Returns

The source of the Text widget.

Description

`XmTextGetSource`() returns the source of the specified Text *widget*. Every Text widget has an XmTextSource data structure associated with it that functions as the text source and sink.

Usage

Multiple text widgets can share the same text source, which means that editing in one of the widgets is reflected in all of the others. `XmTextGetSource`() retrieves the source for a widget; this source can then be used to set the source of another Text widget using `XmTextSetSource`(). `XmTextGetSource`() is a convenience routine that returns the value of the XmNsource resource for the Text widget. Calling the routine is equivalent to calling `XtGetValues`() for the resource, although the routine accesses the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextSetSource`(1), `XmText`(2).

Name

XmTextGetString, XmTextFieldGetString – get the text string.

Synopsis

#include <Xm/Text.h>

char * XmTextGetString (Widget *widget*)

#include <Xm/TextF.h>

char * XmTextFieldGetString (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

A string containing the value of the Text or TextField widget.

Description

`XmTextGetString`() and `XmTextFieldGetString`() return a pointer to a character string containing the value of the specified *widget*. `XmTextGetString`() works when widget is a Text widget or a TextField widget, while `XmTextFieldGetString`() only works for a TextField widget. For each routine, if the string has a length of 0 (zero), the returned value is the empty string. Storage for the returned string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

`XmTextGetString`() and `XmTextFieldGetString`() are convenience routines that return the value of the XmNvalue resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you set the XmNvalueWcs resource, you can still use `XmTextGetString`() or `XmTextFieldGetString`() to retrieve the value of the widget, since the value is stored internally as a multi-byte string.

Example

The following routine shows the use of XmTextGetString() to retrieve the text from one Text widget and use the text to search for the string in another Text widget:

```
Widget text_w, search_w;

void search_text (void)
{
    char            *search_pat;
    XmTextPosition  pos, search_pos;
    Boolean         found = False;

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    /* find next occurrence from current position -- wrap if necessary */
    pos = XmTextGetCursorPosition (text_w);
    found = XmTextFindString (text_w, pos, search_pat,
    XmTEXT_FORWARD, &search_pos);

    if (!found)
        found = XmTextFindString (text_w, 0, search_pat,
        XmTEXT_FORWARD, &search_pos);

    if (found)
        XmTextSetInsertionPosition (text_w, search_pos);
        XtFree (search_pat);
}
```

See Also

XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextSetString(1), XmTextSetStringWcs(1), XmText(2), XmTextField(2).

Name

XmTextGetStringWcs, XmTextFieldGetStringWcs – get the wide-character text string.

Synopsis

```
#include <Xm/Text.h>

wchar_t * XmTextGetStringWcs (Widget widget)

#include <Xm/TextF.h>

wchar_t * XmTextFieldGetStringWcs (Widget widget)
```

Inputs

widget Specifies the Text or TextField widget.

Returns

A wide-character string containing the value of the Text or TextField widget.

Availability

Motif 1.2 and later.

Description

`XmTextGetStringWcs`() and `XmTextFieldGetStringWcs`() return a pointer to a wide-character string containing the value of the specified *widget*. `XmTextGetStringWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetStringWcs`() only works for a TextField widget. For each routine, if the string has a length of 0 (zero), the returned value is the empty string. Storage for the returned wide-character string is allocated by the routine and should be freed by calling `XtFree`(). Management of the allocated memory is the responsibility of the application.

Usage

`XmTextGetStringWcs`() and `XmTextFieldGetStringWcs`() are convenience routines that return the value of the XmNvalueWcs resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you use the XmNvalue resource to set the value of a widget, you can still use `XmTextGetStringWcs`() or `XmTextFieldGetStringWcs`() to retrieve the value of the widget, since the value can be converted to a wide-character string.

See Also

XmTextGetString(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextSetString(1), XmTextSetStringWcs(1), XmText(2), XmTextField(2).

Name

XmTextGetSubstring, XmTextFieldGetSubstring – get a copy of part of the text string.

Synopsis

```
#include <Xm/Text.h>

int XmTextGetSubstring ( Widget          widget,
                         XmTextPosition  start,
                         int             num_chars,
                         int             buffer_size,
                         char            *buffer)

#include <Xm/TextF.h>

int XmTextFieldGetSubstring (Widget          widget,
                             XmTextPosition  start,
                             int             num_chars,
                             int             buffer_size,
                             char            *buffer)
```

Inputs

widget	Specifies the Text or TextField widget.
start	Specifies the starting character position from which data is copied.
num_chars	Specifies the number of characters that are copied.
buffer_size	Specifies the size of buffer.
buffer	Specifies the character buffer where the copy is stored.

Returns

XmCOPY_SUCCEEDED on success, XmCOPY_TRUNCATED if fewer than *num_chars* are copied, or XmCOPY_FAILED on failure.

Availability

Motif 1.2 and later.

Description

`XmTextGetSubstring`() and `XmTextFieldGetSubstring`() get a copy of part of the internal text buffer for the specified *widget*. `XmTextGetSubstring`()[1] works when widget is a Text widget or a TextField widget, while `XmTextFieldGetSubstring`()[2] only works for a TextField widget. The routines copy *num_chars* characters starting at *start* position, which specifies the

1.Erroneously given as XmTextGetString() in 1st and 2nd editions.

2.Erroneously given as XmTextFieldGetString() in 1st and 2nd editions.

position as the number of characters from the beginning of the text, where the first character position is 0 (zero). The characters are copied into the provided *buffer* and are NULL-terminated.

`XmTextGetSubstring`() and `XmTextFieldGetSubstring`() return XmCOPY_SUCCEEDED on success. If the specified *num_chars* does not fit in the provided *buffer*, the routines return XmCOPY_TRUNCATED. In this case, *buffer* contains as many characters as would fit plus a NULL terminator. If either of the routines fails to make the copy, it returns XmCOPY_FAILED and the contents of *buffer* are undefined.

Usage

`XmTextGetSubstring`() and `XmTextFieldGetSubstring`() provide a convenient way to retrieve a portion of the text string in a Text or TextField widget. The routines return the specified part of the XmNvalue resource for the widget.

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you set the XmNvalueWcs resource, you can still use `XmTextGetSubstring`() or `XmTextFieldGetSubstring`() to retrieve part of the value of the widget, since the value is stored internally as a multi-byte string.

The necessary *buffer_size* for `XmTextGetSubstring`() and `XmTextFieldGetSubstring`() depends on the maximum number of bytes per character for the current locale. This information is stored in MB_CUR_MAX, a macro defined in <stdlib.h>. The *buffer* needs to be large enough to store the substring and a NULL terminator. You can use the following equation to calculate the necessary *buffer_size*:

buffer_size = (num_chars * *MB_CUR_MAX)* + *1*

See Also

`XmTextGetString`(1), `XmTextGetStringWcs`(1), `XmTextGetSubstringWcs`(1), `XmTextSetString`(1), `XmTextSetStringWcs`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextGetSubstringWcs, XmTextFieldGetSubstringWcs – get a copy of part of the wide-character text string.

Synopsis

```
#include <Xm/Text.h>

int XmTextGetSubstringWcs ( Widget          widget,
                            XmTextPosition  start,
                            int             num_chars,
                            int             buffer_size,
                            wchar_t         *buffer)

#include <Xm/TextF.h>

int XmTextFieldGetSubstringWcs ( Widget          widget,
                                 XmTextPosition  start,
                                 int             num_chars,
                                 int             buffer_size,
                                 wchar_t         *buffer)
```

Inputs

widget	Specifies the Text or TextField widget.
start	Specifies the starting character position from which data is copied.
num_chars	Specifies the number of wide-characters that are copied.
buffer_size	Specifies the size of buffer.
buffer	Specifies the wide-character buffer where the copy is stored.

Returns

XmCOPY_SUCCEEDED on success, XmCOPY_TRUNCATED if fewer than num_chars are copied, or XmCOPY_FAILED on failure.

Availability

Motif 1.2 and later.

Description

`XmTextGetSubstringWcs`() and `XmTextFieldGetSubstringWcs`() get a copy of part of the internal wide-character text buffer for the specified *widget*. `XmTextGetSubstringWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldGetSubstringWcs`() only works for a TextField widget. The routines copy num_chars wide-characters starting at *start* position, which specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). The wide-characters are copied into the provided *buffer* and are NULL-terminated.

XmTextGetSubstringWcs() and XmTextFieldGetSubstringWcs() return XmCOPY_SUCCEEDED on success. If the specified *num_chars* does not fit in the provided *buffer*, the routines return XmCOPY_TRUNCATED. In this case, *buffer* contains as many wide-characters as would fit plus a NULL terminator. If either of the routines fails to make the copy, it returns XmCOPY_FAILED and the contents of *buffer* are undefined.

Usage

XmTextGetSubstringWcs() and XmTextFieldGetSubstringWcs() provide a convenient way to retrieve a portion of the wide-character text string in a Text or TextField widget. The routines return the specified part of the XmNvalueWcs resource for the widget.

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. Even if you use the XmNvalue resource to set the value of a widget, you can still use XmTextGetSubstringWcs() or XmTextFieldGetSubstringWcs() to retrieve part of the value of the widget, since the value can be converted to a wide-character string.

The necessary *buffer_size* for XmTextGetSubstringWcs() and XmTextFieldGetSubstringWcs() is *num_chars* + 1.

See Also

XmTextGetString(1), XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextSetString(1), XmTextSetStringWcs(1), XmText(2), XmTextField(2).

Name

XmTextGetTopCharacter – get the position of the first character of text that is displayed.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextGetTopCharacter (Widget *widget*)

Inputs

widget Specifies the Text widget.

Returns

The position of the first visible character.

Description

`XmTextGetTopCharacter`() returns the value of the XmNtopCharacter resource for the specified Text *widget*. The returned value specifies the position of the first visible character of text as the number of characters from the beginning of the text, where the first character position in 0 (zero).

Usage

`XmTextGetTopCharacter`() is a convenience routine that returns the value of the XmNtopCharacter resource for a Text widget. Calling the routines is equivalent to calling `XtGetValues`() for the resource, although the routines accesses the value through the widget instance structures rather than through `XtGetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetLastPosition`(1), `XmTextScroll`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetTopCharacter`(1), `XmTextShowPosition`(1), `XmText`(2).

Name

XmTextInsert, XmTextFieldInsert – insert a string into the text string.

Synopsis

#include <Xm/Text.h>

void XmTextInsert (Widget *widget*, XmTextPosition *position*, char **value*)

#include <Xm/TextF.h>

void XmTextFieldInsert (Widget *widget*, XmTextPosition *position*, char **string*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position at which the string is inserted.
string	Specifies the string to be inserted.

Description

`XmTextInsert`() and `XmTextFieldInsert`() insert a text *string* in the specified Text or TextField *widget*. `XmTextInsert`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldInsert`() only works for a TextField widget. The specified *string* is inserted at *position*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To insert a string after the nth character, use a position value of n.

`XmTextInsert`() and `XmTextFieldInsert`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextInsert`() and `XmTextFieldInsert`() provide a convenient means of inserting text in a Text or TextField widget. The routines insert text by modifying the value of the XmNvalue resource of the widget.

Example

The following routine shows the use of `XmTextInsert`() to insert a message into a status Text widget:

```
Widget status;

void insert_text (char *message)
{
    XmTextPosition curpos = XmTextGetInsertionPosition (status);
```

```
    XmTextInsert (status, curpos, message);

    curpos = curpos + strlen (message);
    XmTextShowPosition (status, curpos);
    XmTextSetInsertionPosition (status, curpos);
}
```

See Also

XmTextInsertWcs(1), XmTextReplace(1), XmTextReplaceWcs(1), XmText(2), XmTextField(2).

Name

XmTextInsertWcs, XmTextFieldInsertWcs – insert a wide-character string into the text string.

Synopsis

#include <Xm/Text.h>

void XmTextInsertWcs (Widget *widget*, XmTextPosition *position*, wchar_t **wcstring*)[1]

#include <Xm/TextF.h>

void XmTextFieldInsertWcs (Widget *widget*, XmTextPosition *position*, wchar_t * *wcstring*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position at which the string is inserted.
wcstring	Specifies the wide-character string to be inserted.

Availability

Motif 1.2 and later.

Description

`XmTextInsertWcs`() and `XmTextFieldInsertWcs`() insert a wide-character text *wcstring* in the specified *widget*. `XmTextInsertWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldInsertWcs`() only works for a TextField widget. The specified *wcstring*[2] is inserted at *position*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To insert a string after the *n*th character, use a position value of n.

`XmTextInsertWcs`() and `XmTextFieldInsertWcs`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

1.Erroneously given as XmTextInsert() in 1st and 2nd editions.

2.Erroneously given as *string* in 1st and 2nd editions.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. `XmTextInsertWcs`() and `XmTextFieldInsertWcs`*()* provide a convenient means of inserting a wide-character string in a Text or TextField widget. The routines insert text by converting the wide-character string to a multi-byte string and then modifying the value of the XmNvalue resource of the widget.

See Also

`XmTextInsert`(1), `XmTextReplace`(1), `XmTextReplaceWcs`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextPaste, XmTextFieldPaste – insert the clipboard selection.

Synopsis

#include <Xm/Text.h>

Boolean XmTextPaste (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldPaste (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

True on success or False otherwise.

Description

`XmTextPaste`() and `XmTextFieldPaste`() insert the clipboard selection at the current position of the insertion cursor in the specified *widget*. `XmText-Paste`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldPaste`() only works for a TextField widget. If the insertion cursor is within the current selection and the value of XmNpendingDelete is True, the current selection is replaced by the clipboard selection. Both routines return True if successful. If the *widget* is not editable or if the function cannot obtain ownership of the clipboard selection, the routines return False.

In Motif 2.0 and later, `XmTextPaste`() interfaces with the Uniform Transfer Model by indirectly invoking the XmNdestinationCallback procedures of the widget.

`XmTextPaste`() and `XmTextFieldPaste`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextPaste`() and `XmTextFieldPaste`() get the current selection from the clipboard and insert it at the location of the insertion cursor in the Text or TextField widget.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of `XmTextPaste`()**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                     num = (int) client_data;
    XmAnyCallbackStruct     *cbs = (XmAnyCallbackStruct *) call_data;
    Boolean                 result = True;

    switch (num) {
        case 0:   result = XmTextCut (text_w, cbs->event->xbutton.time);break;
        case 1:   result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:   result = XmTextCopyLink (text_w, cbs->event->xbutton.time);
        break;
        case 3:   result = XmTextPaste (text_w);break;
        case 4:   result = XmTextPasteLink (text_w);break;
        case 5:   XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

`XmTextCopy`(1), `XmTextCopyLink`(1), `XmTextCut`(1), `XmTextPasteLink`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextPasteLink, XmTextFieldPasteLink – insert the clipboard selection.

Synopsis

#include <Xm/Text.h>

Boolean XmTextPasteLink (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldPasteLink (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

True on success or False otherwise.

Availability

Motif 2.0 and later.

Description

`XmTextPasteLink`() and `XmTextFieldPasteLink`() insert the clipboard selection at the current position of the insertion cursor in the specified *widget*. `XmTextPasteLink`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldPasteLink`() only works for a TextField widget. If the insertion cursor is within the current selection and the value of XmNpendingDelete is True, the current selection is replaced by the clipboard selection. Both routines return True if successful. If the *widget* is not editable or if the function cannot obtain ownership of the clipboard selection, the routines return False.

`XmTextPasteLink`() and `XmTextFieldPasteLink`() interface with the Uniform Transfer Model by indirectly invoking the XmNdestinationCallback procedures of the widget. The Text widget itself does not create links to the primary selection: destination callbacks provided by the programmer are responsible for performing any data transfer required.

`XmTextPasteLink`() and `XmTextFieldPasteLink`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

XmTextPasteLink() and XmTextFieldPasteLink() get the current selection from the clipboard and insert a link to it at the location of the insertion cursor in the Text or TextField widget.

Example

The following callback routine for the items on an **Edit** menu (**Cut**, **Copy**, **Link**, **Paste**, **PasteLink**, and **Clear**) shows the use of XmTextPasteLink()[1]**:**

```
Widget text_w, status;

void cut_paste (Widget widget, XtPointer client_data, XtPointer call_data)
{
    int                 num = (int) client_data;
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct *) call_data;
    Boolean             result = True;

    switch (num) {
        case 0:  result = XmTextCut (text_w, cbs->event->xbutton.time);break;
        case 1:  result = XmTextCopy (text_w, cbs->event->xbutton.time);
        break;
        case 2:  result = XmTextCopyLink (text_w, cbs->event->xbutton.time);
        break;
        case 3:  result = XmTextPaste (text_w);break;
        case 4:  result = XmTextPasteLink (text_w);break;
        case 5:  XmTextClearSelection (text_w, cbs->event->xbutton.time);
        break;
    }

    if (result == False)
        XmTextSetString (status, "There is no selection.");
    else
        XmTextSetString (status, NULL);
}
```

See Also

XmTextCopy(1), XmTextCopyLink(1), XmTextCut(1), XmTextPaste(1), XmText(2), XmTextField(2).

1.Erroneously given as XmTextPaste() in 1st and 2nd editions.

Name

XmTextPosToXY, XmTextFieldPosToXY – get the x, y position of a character position.

Synopsis

#include <Xm/Text.h>

Boolean XmTextPosToXY (Widget *widget*, XmTextPosition *position*, Position **x*, Position **y*)

#include <Xm/TextF.h>

Boolean XmTextFieldPosToXY (Widget *widget*, XmTextPosition *position*, Position **x*, Position **y*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the character position.

Outputs

x	Returns the x-coordinate of the character position.
y	Returns the y-coordinate of the character position.

Returns

True if the character position is displayed in the widget or False otherwise.

Description

`XmTextPosToXY`() and `XmTextFieldPosToXY`() return the *x* and *y* coordinates of the character at the specified *position* within the specified *widget*. `XmTextPosToXY`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldPosToXY`() only works for a TextField widget. Character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. The returned coordinate values are specified relative to the upper-left corner of widget. Both routines return True if the character at *position* is currently displayed in the widget. Otherwise, the routines return False and no values are returned in the x and y arguments.

Usage

`XmTextPosToXY`() and `XmTextFieldPosToXY`() provide a way to determine the actual position of a character in a Text or TextField widget. This information is useful if you need to perform additional event processing or draw special graphics in the widget.

See Also

`XmTextXYToPos`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextRemove, XmTextFieldRemove – delete the primary selection.

Synopsis

#include <Xm/Text.h>

Boolean XmTextRemove (Widget *widget*)

#include <Xm/TextF.h>

Boolean XmTextFieldRemove (Widget *widget*)

Inputs

widget Specifies the Text or TextField widget.

Returns

True on success or False otherwise.

Description

`XmTextRemove`() and `XmTextFieldRemove`() delete the primary selected text from the specified *widget*. `XmTextRemove`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldRemove`() only works for a TextField widget. Both routines return True if successful. If the *widget* is not editable, if the primary selection is NULL, or if it is not owned by the specified widget, the routines return False.

`XmTextRemove`() and `XmTextFieldRemove`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextRemove`() and `XmTextFieldRemove`() are like `XmTextCut`() and `XmTextFieldCut`(), in that they remove selected text from a Text or TextField widget. However, the routines do not copy the selected text to the clipboard before removing it.

See Also

`XmTextClearSelection`(1), `XmTextCut`(1), `XmTextGetSelection`(1), `XmTextGetSelectionPosition`(1), `XmTextGetSelectionWcs`(1), `XmTextSetSelection`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextReplace, XmTextFieldReplace – replace part of the text string.

Synopsis

```
#include <Xm/Text.h>

void XmTextReplace (Widget          widget,
                    XmTextPosition  from_pos,
                    XmTextPosition  to_pos,
                    char            *value)

#include <Xm/TextF.h>

void XmTextFieldReplace ( Widget          widget,
                          XmTextPosition  from_pos,
                          XmTextPosition  to_pos,
                          char            *value)
```

Inputs

widget	Specifies the Text or TextField widget.
from	Specifies the starting position of the text that is to be replaced.
to	Specifies the ending position of the text that is to be replaced.
value	Specifies the replacement string.

Description

`XmTextReplace`() and `XmTextFieldReplace`() replace a portion of the text *string* in the specified *widget*. `XmTextReplace`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldReplace`() only works for a TextField widget. The specified *value* replaces the text starting at *from_pos* and continuing up to, but not including, *to_pos*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To replace the characters after the *n*th character up to the *m*th character, use a *from_pos* value of *n* and a *to_pos* value of *m*.

`XmTextReplace`() and `XmTextFieldReplace`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

`XmTextReplace`() and `XmTextFieldReplace`() provide a convenient means of replacing text in a Text or TextField widget. The routines replace text by modifying the value of the XmNvalue resource of the widget.

Example

The following routine shows the use of XmTextReplace() to replace all of the occurrences of a string in a Text widget. The search and replacement strings are specified by the user in single-line Text widgets:

```
Widget text_w, search_w, replace_w;

void search_and_replace (void)
{
    char            *search_pat, *new_pat;
    XmTextPosition  curpos, searchpos;
    int             search_len, pattern_len;
    Boolean         found = False;

    search_len = XmTextGetLastPosition (search_w);

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    pattern_len = XmTextGetLastPosition (replace_w);

    if (!(new_pat = XmTextGetString (replace_w)) || !*new_pat) {
        XtFree (search_pat);
        XtFree (new_pat);
        return;
    }

    curpos = 0;
    found = XmTextFindString (text_w, curpos, search_pat,
    XmTEXT_FORWARD, &searchpos);

    while (found) {
        XmTextReplace (text_w, searchpos, searchpos + search_len, new_pat);
        curpos = searchpos + 1;
        found = XmTextFindString (text_w, curpos, search_pat,
        XmTEXT_FORWARD, &searchpos);
    }

    XtFree (search_pat);
    XtFree (new_pat);
}
```

See Also

XmTextInsert(1), XmTextInsertWcs(1), XmTextReplaceWcs(1), XmText(2), XmTextField(2).

Name

XmTextReplaceWcs, XmTextFieldReplaceWcs – replace part of the wide-character text string.

Synopsis

```
#include <Xm/Text.h>

void XmTextReplaceWcs (Widget          widget,
                       XmTextPosition  from_pos,
                       XmTextPosition  to_pos,
                       wchar_t         *wcstring)

#include <Xm/TextF.h>

void XmTextFieldReplaceWcs ( Widget          widget,
                             XmTextPosition  from_pos,
                             XmTextPosition  to_pos,
                             wchar_t         *wcstring)
```

Inputs

widget	Specifies the Text or TextField widget.
from_pos	Specifies the starting position of the text that is to be replaced.
to_pos	Specifies the ending position of the text that is to be replaced.
wcstring	Specifies the replacement wide-character string.

Availability

Motif 1.2 and later.

Description

`XmTextReplaceWcs`() and `XmTextFieldReplaceWcs`() replace a portion of the text string in the specified widget with the specified wide-character string *wcstring*. `XmTextReplaceWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldReplaceWcs`() only works for a TextField widget. The specified *wcstring* replaces the text starting at *from_pos* and continuing up to, but not including, *to_pos*, where character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text. To replace the characters after the *n*th character up to the *m*th character, use a *from_pos* value of *n* and a *to_pos* value of *m*.

`XmTextReplaceWcs`() and `XmTextFieldReplaceWcs`() also invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. XmTextReplaceWcs() and XmTextFieldReplaceWcs() provide a convenient means of replacing a string in a Text or TextField widget with a wide-character string. The routines convert the wide-character string to a multi-byte string and then replace the text by modifying the value of the XmNvalue resource of the widget.

See Also

XmTextInsert(1), XmTextInsertWcs(1), XmTextReplace(1), XmText(2), XmTextField(2).

Name

XmTextScroll – scroll the text.

Synopsis

#include <Xm/Text.h>

void XmTextScroll (Widget *widget*, int *lines*)

Inputs

widget	Specifies the Text widget.
lines	Specifies the number of lines.

Description

`XmTextScroll`() scrolls the text in the specified Text *widget* by the specified number of *lines*. The text is scrolled upward if *lines* is positive and downward if *lines* is negative. In the case of vertical text, a positive value scrolls the text forwards, and a negative value scrolls backwards.

Usage

`XmTextScroll`() provides a way to perform relative scrolling in a Text widget. The Text widget does not have to be the child of a ScrolledWindow for the scrolling to occur. The routine simply changes the currently viewable region of text.

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetLastPosition`(1), `XmTextGetTopCharacter`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextSetTopCharacter`(1), `XmText`(2).

Name

XmTextSetAddMode, XmTextFieldSetAddMode – set the add mode state.

Synopsis

```
#include <Xm/Text.h>

void XmTextSetAddMode (Widget widget, Boolean state)

#include <Xm/TextF.h>

void XmTextFieldSetAddMode (Widget widget, Boolean state)
```

Inputs

widget	Specifies the Text or TextField widget.
state	Specifies the state of add mode.

Description

`XmTextSetAddMode`() and `XmTextFieldSetAddMode`() set the state of add mode for the specified *widget*. `XmTextSetAddMode`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetAddMode`() only works for a TextField widget. If *state* is True add mode is turned on; if *state* is False, add mode is turned off. When a Text or TextField widget is in add mode, the user can move the insertion cursor without altering the primary selection.

Usage

`XmTextSetAddMode`() and `XmTextFieldSetAddMode`() provide a way to change the state of add mode in a Text or TextField widget. The distinction between normal mode and add mode is only important for making keyboard-based selections. In normal mode, the location cursor and the selection move together, while in add mode, the location cursor and the selection can be separate.

See Also

`XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetCursorPosition, XmTextFieldSetCursorPosition – set the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

void XmTextSetCursorPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/TextF.h>

void XmTextFieldSetCursorPosition (Widget *widget*, XmTextPosition *position*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position of the insertion cursor.

Description

`XmTextSetCursorPosition`() and `XmTextFieldSetCursorPosition`() set the value of the XmNcursorPosition resource to position for the specified *widget*. `XmTextSetCursorPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetCursorPosition`() only works for a TextField widget. This resource specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

`XmTextSetCursorPosition`() and `XmTextFieldSetCursorPosition`() also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget if the position of the insertion cursor changes.

Usage

`XmTextSetCursorPosition`() and `XmTextFieldSetCursorPosition`() are convenience routines that set the value of the XmNcursorPosition resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetEditable, XmTextFieldSetEditable – set the edit permission state.

Synopsis

#include <Xm/Text.h>

void XmTextSetEditable (Widget *widget*, Boolean *editable*)

#include <Xm/TextF.h>

void XmTextFieldSetEditable (Widget *widget*, Boolean *editable*)

Inputs

widget	Specifies the Text or TextField widget.
editable	Specifies whether or not the text can be edited.

Description

`XmTextSetEditable`() and `XmTextFieldSetEditable`() set the value of the XmNeditable resource to editable for the specified *widget*. `XmTextSetEditable`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetEditable`() only works for a TextField widget.

Usage

By default, the XmNeditable resource is True, which means that a user can edit the text string. Setting the resource to False makes a text area read-only. `XmTextSetEditable`() and `XmTextFieldSetEditable`() are convenience routines that set the value of the XmNeditable resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

See Also

`XmTextGetEditable`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetHighlight, XmTextFieldSetHighlight – highlight text.

Synopsis

```
#include <Xm/Text.h>

void XmTextSetHighlight (  Widget          widget,
                           XmTextPosition  left,
                           XmTextPosition  right,
                           XmHighlightMode mode)

#include <Xm/TextF.h>

void XmTextFieldSetHighlight (  Widget          widget,
                                XmTextPosition  left,
                                XmTextPosition  right,
                                XmHighlightMode mode)
```

Inputs

widget	Specifies the Text or TextField widget.
left	Specifies the left boundary position of the text to be highlighted.
right	Specifies the right boundary position of the text to be highlighted.
mode	Specifies the highlighting mode. Pass one of the following values: XmHIGHLIGHT_NORMAL, XmHIGHLIGHT_SELECTED, or XmHIGHLIGHT_SECONDARY_SELECTED[1].

Description

`XmTextSetHighlight`() and `XmTextFieldSetHighlight`() highlight text in the specified *widget* without selecting the text. `XmTextSetHighlight`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetHighlight`() only works for a TextField widget. The *left* and *right* arguments specify the boundary positions of the text that is to be highlighted. Each boundary value specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). The *mode* parameter indicates the type of highlighting that is done. XmHIGHLIGHT_NORMAL removes any highlighting, XmHIGHLIGHT_SELECTED uses reverse video highlighting, and XmHIGHLIGHT_SECONDARY_SELECTED uses underline highlighting.

1.Motif 2.0 defines the additional value XmSEE_DETAIL for the enumerated type, but does not use it for the Text components. The Compound String Text, CSText, supports the notion, but this widget is abortive, and has been removed from the 2.1 distribution. XmSEE_DETAIL is therefore redundant.

Usage

XmTextSetHighlight() and XmTextFieldSetHighlight() provide a way to highlight text in a Text or TextField widget. These routines are useful if you need to emphasize certain text in a widget. These routine only highlight text; they do not select the specified text.

Example

The following routine shows the use of XmTextSetHighlight() to highlight all of the occurrences of a string in a Text widget. The search string is specified by the user in a single-line Text widget:

```
Widget text_w, search_w;

void search_text (void)
{
    char            *search_pat;
    XmTextPosition  curpos, searchpos;
    int             len;
    Boolean         found = False;

    len = XmTextGetLastPosition (search_w);

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    curpos = 0;
    found = XmTextFindString (text_w, curpos, search_pat,
    XmTEXT_FORWARD, &searchpos);

    while (found) {
        XmTextSetHighlight (text_w, searchpos, searchpos + len,
                            XmHIGHLIGHT_SECONDARY_SELECTE
                            D);
        curpos = searchpos + 1;
        found = XmTextFindString (text_w, curpos, search_pat,
        XmTEXT_FORWARD, &searchpos);
    }
    XtFree (search_pat);
}
```

See Also

XmTextSetSelection(1), XmText(2), XmTextField(2).

Name

XmTextSetInsertionPosition, XmTextFieldSetInsertionPosition – set the position of the insertion cursor.

Synopsis

#include <Xm/Text.h>

void XmTextSetInsertionPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/TextF.h>

void XmTextFieldSetInsertionPosition (Widget *widget*, XmTextPosition *position*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the position of the insertion cursor.

Description

The functions, XmTextSetInsertionPosition() and `XmTextFieldSetInsertionPosition`(), set the value of the XmNcursorPosition resource to *position* for the specified *widget*. `XmTextSetInsertionPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetInsertionPosition`() only works for a TextField widget. This resource specifies the location of the insertion cursor as the number of characters from the beginning of the text, where the first character position is 0 (zero).

`XmTextSetInsertionPosition`() and `XmTextFieldSetInsertionPosition`() also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget if the position of the insertion cursor changes.

Usage

The functions, `XmTextSetInsertionPosition`() and `XmTextFieldSetInsertionPosition`(), are convenience routines that set the value of the XmNcursorPosition resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

Example

The following code shows the use of `XmTextSetInsertionPosition`() in a routine that searches for a string in a Text widget and moves the insertion cursor to the string if it is found:

```
Widget text_w, search_w;
```

```
void search_text (void)
{
    char            *search_pat;
    XmTextPosition  pos, searchpos;
    Boolean         found = False;

    if (!(search_pat = XmTextGetString (search_w)) || !*search_pat) {
        XtFree (search_pat);
        return;
    }

    pos = XmTextGetCursorPosition (text_w);
    found = XmTextFindString (text_w, pos, search_pat,
    XmTEXT_FORWARD, &searchpos);

    if (!found) {
        found = XmTextFindString (text_w, 0, search_pat,
        XmTEXT_FORWARD, &searchpos);
    }

    if (found)
        XmTextSetInsertionPosition (text_w, searchpos);
    XtFree (search_pat);
}
```

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextSetCursorPosition`(1), `XmTextShowPosition`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetMaxLength, XmTextFieldSetMaxLength – set the maximum possible length of a text string.

Synopsis

#include <Xm/Text.h>

void XmTextSetMaxLength (Widget *widget*, int *max_length*)

#include <Xm/TextF.h>

void XmTextFieldSetMaxLength (Widget *widget*, int *max_length*)

Inputs

widget	Specifies the Text or TextField widget.
max_length	Specifies the maximum allowable length of the text string.

Description

`XmTextSetMaxLength`() and `XmTextFieldSetMaxLength`() set the value of the XmNmaxLength resource to *max_length* for the specified *widget*. `XmTextSetMaxLength`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetMaxLength`() only works for a TextField widget. This resource specifies the maximum allowable length of a text string that a user can enter from the keyboard.

Usage

`XmTextSetMaxLength`() and `XmTextFieldSetMaxLength`() are convenience routines that set the XmNmaxLength resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`(). The resource limits the length of a text string that a user may type, but it does not limit the length of strings entered with the XmNvalue or XmNvalueWcs resources or the `XmTextSetString`(), `XmTextFieldSetString`(), `XmTextSetStringWcs`(), and `XmTextFieldSetStringWcs`() routines.

See Also

`XmTextGetMaxLength`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetSelection, XmTextFieldSetSelection – set the value of the primary selection.

Synopsis

#include <Xm/Text.h>

void XmTextSetSelection (Widget *widget*, XmTextPosition *first*, XmTextPosition *last*, Time *time*)

#include <Xm/TextF.h>

void XmTextFieldSetSelection (Widget *widget*, XmTextPosition *first*, XmTextPosition *last*, Time *time*)

Inputs

widget	Specifies the Text or TextField widget.
first	Specifies the first character position to be selected.
last	Specifies the last character position to be selected.
time	Specifies the time of the event that caused the request.

Description

`XmTextSetSelection`() and `XmTextFieldSetSelection`() set the primary selection in the specified *widget*. `XmTextSetSelection`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetSelection`() only works for a TextField widget. The *first* and *last* arguments specify the beginning and ending positions of the text that is to be selected. Each of these values specifies the position as the number of characters from the beginning of the text, where the first character position is 0 (zero). For each routine, *time* specifies the server time of the event that caused the request to set the selection.

`XmTextSetSelection`() and `XmTextFieldSetSelection`() change the insertion cursor for the *widget* to the last position of the selection. The routines also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget.

Usage

`XmTextSetSelection`() and `XmTextFieldSetSelection`() provide a convenient way to set the current selection in a Text or TextField widget.

See Also

`XmTextClearSelection`(1), `XmTextCopy`(1), `XmTextCut`(1), `XmTextGetSelection`(1), `XmTextGetSelectionPosition`(1), `XmTextGetSelectionWcs`(1), `XmTextRemove`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetSource – set the text source.

Synopsis

```
#include <Xm/Text.h>

void XmTextSetSource (Widget        widget,
                      XmTextSource  source,
                      XmTextPosition top_character,
                      XmTextPosition cursor_position)
```

Inputs

widget	Specifies the Text widget.
source	Specifies the text source.
top_character	Specifies the character position to display at the top of the widget.
cursor_position	Specifies the position of the insertion cursor.

Description

`XmTextSetSource`() sets the source of the specified Text *widget*. The *top_character* and *cursor_position* values specify positions as the number of characters from the beginning of the text, where the first character position is 0 (zero). If *source* is NULL, the Text widget creates a default string source and displays a warning message.

Usage

Multiple text widgets can share the same text source, which means that editing in one of the widgets is reflected in all of the others. `XmTextGetSource`() retrieves the source for a widget; this source can then be used to set the source of another Text widget using `XmTextSetSource`(). `XmTextSetSource`() is a convenience routine that sets the value of the XmNsource resource for the Text widget. Calling the routine is equivalent to calling `XtSetValues`() for the resource, although the routine accesses the value through the widget instance structures rather than through `XtSetValues`().

When a new text source is set, the old text source is destroyed unless another Text widget is using the old source. If you want to replace a text source without destroying it, create an unmanaged Text widget and set its source to the text source you want to save.

See Also

`XmTextGetSource`(1), `XmText`(2).

Name

XmTextSetString, XmTextFieldSetString – set the text string.

Synopsis

#include <Xm/Text.h>

void XmTextSetString (Widget *widget*, char **value*)

#include <Xm/TextF.h>

void XmTextFieldSetString (Widget *widget*, char **value*)

Inputs

widget	Specifies the Text or TextField widget.
value	Specifies the string value.

Description

`XmTextSetString`() and `XmTextFieldSetString`() set the current text string in the specified *widget* to the specified *value*. `XmTextSetString`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetString`() only works for a TextField widget. Both functions also set the position of the insertion cursor to the beginning of the new text string.

`XmTextSetString`() and `XmTextFieldSetString`() invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified widget. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures. The routines also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget.

Usage

`XmTextSetString`() and `XmTextFieldSetString`() are convenience routines that set the value of the XmNvalue resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

Example

The following code shows the use of `XmTextSetString`() in a routine that displays the contents of file in a Text widget. The filename is specified by the user in a TextField widget:

```
Widget text_w, file_w;

void read_file (void)
```

```
{
    char        *filename, *text;
    struct stat statb;
    int         fd, len;

    if (!(filename = XmTextFieldGetString (file_w)) || !*filename) {
        XtFree (filename);
        return;
    }

    if (!(fd = open (filename, O_RDONLY))) {
        XtWarning ("internal error -- can't open file");
    }

    if (fstat (fd, &statb) == -1 || !(text = XtMalloc ((len = statb.st_size) + 1))) {
        XtWarning("internal error -- can't show text");
        (void) close (fd);
    }

    (void) read (fd, text, len);
    text[len] = '\0';
    XmTextSetString (text_w, text);
    XtFree (text);
    XtFree (filename);
    (void) close (fd);
}
```

See Also

`XmTextGetString`(1), `XmTextGetStringWcs`(1), `XmTextGetSubstring`(1), `XmTextGetSubstringWcs`(1), `XmTextSetStringWcs`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetStringWcs, XmTextFieldSetStringWcs – set the wide-character text string.

Synopsis

```
#include <Xm/Text.h>

void XmTextSetStringWcs (Widget widget, wchar_t *wcstring)

#include <Xm/TextF.h>

void XmTextFieldSetStringWcs (Widget widget, wchar_t *wcstring)
```

Inputs

widget	Specifies the Text or TextField widget.
wcstring	Specifies the wide-character string value.

Availability

Motif 1.2 and later.

Description

`XmTextSetStringWcs`() and `XmTextFieldSetStringWcs`() set the current wide-character text string in the specified *widget* to the specified *wcstring*[1]. `XmTextSetStringWcs`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldSetStringWcs`() only works for a TextField widget. Both functions also set the position of the insertion cursor to the beginning of the new text string.

`XmTextSetStringWcs`() and `XmTextFieldSetStringWcs`() invoke the callback routines for the XmNvalueChangedCallback, the XmNmodifyVerifyCallback, and the XmNmodifyVerifyCallbackWcs callbacks for the specified *widget*. If both verification callbacks are present, the XmNmodifyVerifyCallback procedures are invoked first and the results are passed to the XmNmodifyVerifyCallbackWcs procedures. The routines also invoke the callback routines for the XmNmotionVerifyCallback for the specified widget.

1.Erroneously given as *string* in 1st and 2nd editions.

Usage

In Motif 1.2, the Text and TextField widgets support wide-character strings. The resource XmNvalueWcs can be used to set the value of a Text or TextField widget to a wide-character string. `XmTextSetStringWcs`() and `XmTextFieldSetStringWcs`() are convenience routines that set the value of the XmNvalueWcs resource for a Text or TextField widget. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

See Also

`XmTextGetString`(1), `XmTextGetStringWcs`(1), `XmTextGetSubstring`(1), `XmTextGetSubstringWcs`(1), `XmTextSetString`(1), `XmText`(2), `XmTextField`(2).

Name

XmTextSetTopCharacter – set the position of the first character of text that is displayed.

Synopsis

#include <Xm/Text.h>

void XmTextSetTopCharacter (Widget *widget*, XmTextPosition *top_character*)

Inputs

widget Specifies the Text widget.
top_character Specifies the position that is to be displayed at the top of the widget.

Description

`XmTextSetTopCharacter`() sets the value of the XmNtopCharacter resource to *top_character* for the specified Text *widget*. If the XmNeditMode resource is set to XmMULTI_LINE_EDIT, the routine scrolls the text so that the line containing the character position specified by *top_character* appears at the top of the widget, but does not shift the text left or right. Otherwise, the character position specified by *top_character* is displayed as the first visible character in the widget. *top_character* specifies a character position as the number of characters from the beginning of the text, where the first character position in 0 (zero).

Usage

`XmTextSetTopCharacter`() is a convenience routine that sets the value of the XmNtopCharacter resource for a Text widget. Calling the routines is equivalent to calling `XtSetValues`() for the resource, although the routines accesses the value through the widget instance structures rather than through `XtSetValues`().

See Also

`XmTextGetCursorPosition`(1), `XmTextGetInsertionPosition`(1), `XmTextGetLastPosition`(1), `XmTextGetTopCharacter`(1), `XmTextScroll`(1), `XmTextSetCursorPosition`(1), `XmTextSetInsertionPosition`(1), `XmTextShowPosition`(1), `XmText`(2).

Name

XmTextShowPosition, XmTextFieldShowPosition – display the text at a specified position.

Synopsis

#include <Xm/Text.h>

void XmTextShowPosition (Widget *widget*, XmTextPosition *position*)

#include <Xm/TextF.h>

void XmTextFieldShowPosition (Widget *widget*, XmTextPosition *position*)

Inputs

widget	Specifies the Text or TextField widget.
position	Specifies the character position that is to be displayed.

Description

`XmTextShowPosition`() and `XmTextFieldShowPosition`() cause the text character at *position* to be displayed in the specified *widget*. `XmTextShowPosition`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldShowPosition`() only works for a TextField widget. The *position* argument specifies the position as the number of characters from the beginning of the text, where the first character position in 0 (zero).

Usage

`XmTextShowPosition`() and `XmTextFieldShowPosition`() provide a way to force a Text or TextField widget to display a certain portion of its text. This routine is useful if you modify the value of widget and want the modification to be immediately visible without the user having to scroll the text. If the value of the XmNautoShowCursorPosition resource is True, you should set the insertion cursor to position as well. You can set the insertion cursor by setting the XmNcursorPosition[1] resource or by using `XmTextSetInsertionPosition`() or `XmTextFieldSetInsertionPosition`().

1.Erroneously given as XmcursorPosition in 1st and 2nd editions.

Example

The following code shows the use of XmTextShowPosition() in a routine that inserts a message into a status Text widget:

```
Widget status;

void insert_text (char *message)
{
    XmTextPosition curpos = XmTextGetInsertionPosition (status);

    XmTextInsert (status, curpos, message);
    curpos = curpos + strlen (message);
    XmTextShowPosition (status, curpos);
    XmTextSetInsertionPosition (status, curpos);
}
```

See Also

XmTextGetCursorPosition(1), XmTextGetInsertionPosition(1), XmTextSetCursorPosition(1), XmTextSetInsertionPosition(1), XmText(2), XmTextField(2).

Name

XmTextXYToPos, XmTextFieldXYToPos – get the character position for an x, y position.

Synopsis

#include <Xm/Text.h>

XmTextPosition XmTextXYToPos (Widget *widget*, Position *x*, Position *y*)

#include <Xm/TextF.h>

XmTextPosition XmTextFieldXYToPos (Widget *widget*, Position *x*, Position *y*)

Inputs

widget	Specifies the Text or TextField widget.
x	Specifies the x-coordinate relative to the upper-left corner of the widget.
y	Specifies the y-coordinate relative to the upper-left corner of the widget.

Returns

The character position that is closest to the x, y position.

Description

`XmTextXYToPos`() and `XmTextFieldXYToPos`() return the position of the character closest to the specified *x* and *y* coordinates within the specified *widget*. `XmTextXYToPos`() works when *widget* is a Text widget or a TextField widget, while `XmTextFieldXYToPos`() only works for a TextField widget. The *x* and *y* coordinates are relative to the upper-left corner of the widget. Character positions are numbered sequentially, starting with 0 (zero) at the beginning of the text.

Usage

`XmTextXYToPos`() and `XmTextFieldXYToPos`() provide a way to determine the character at a particular coordinate in a Text or TextField widget. This information is useful if you need to perform additional event processing or draw special graphics in the widget.

See Also

`XmTextPosToXY`(1), `XmText`(2), `XmTextField`(2).

Name

XmToggleButtonGetState, XmToggleButtonGadgetGetState – get the state of a ToggleButton.

Synopsis

#include <Xm/ToggleB.h>

Boolean XmToggleButtonGetState (Widget *widget*)

#include <Xm/ToggleBG.h>

Boolean XmToggleButtonGadgetGetState (Widget *widget*)

Inputs

widget Specifies the ToggleButton or ToggleButtonGadget.

Returns

The state of the button.

Description

`XmToggleButtonGetState`() and `XmToggleButtonGadgetGetState`() return the state of the specified *widget.* `XmToggleButtonGetState`() works when *widget* is a ToggleButton or a ToggleButtonGadget, while `XmToggleButtonGadgetGetState`() only works for a ToggleButtonGadget. In Motif 1.2 and earlier, each of the routines returns True if the button is selected or False if the button is unselected.

In Motif 2.0 and later, a Toggle can be in any of three states: XmSET, XmINDETERMINATE, and XmUNSET, where XmUNSET is equivalent to False and XmSET is equivalent to True. The third indeterminate state is enabled if the Motif 2.x XmNtoggleMode resource of the widget is set to the value XmTOGGLE_INDETERMINATE. If the toggle mode is XmTOGGLE_BOOLEAN, the widget has only two dynamic states, which is consistent with Motif 1.2 behavior.

Usage

`XmToggleButtonGetState`() and `XmToggleButtonGadgetGetState`() are convenience routines that return the value of the XmNset resource for a ToggleButton or ToggleButtonGadget. Calling one of the routines is equivalent to calling `XtGetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtGetValues`().

Because XmToggleButtonGetState() returns the toggle *set* element of its widget instance structure directly, and because XmINDETERMINATE is neither True nor False, programs relying on the strictly Boolean nature of XmToggleButtonGetState() are at risk of error if the toggle is configured for three states. Setting tri-state toggles using a convenience function should be performed using XmToggleButtonSetValue().

See Also

XmToggleButtonSetState(1), XmToggleButtonSetValue(1), XmToggleButton(2), XmToggleButtonGadget(2).

Name

XmToggleButtonSetState, XmToggleButtonGadgetSetState – set the state of a ToggleButton.

Synopsis

#include <Xm/ToggleB.h>

void XmToggleButtonSetState (Widget *widget*, Boolean *state*, Boolean *notify*)

#include <Xm/ToggleBG.h>

void XmToggleButtonGadgetSetState (Widget *widget*, Boolean *state*, Boolean *notify*)

Inputs

widget	Specifies the ToggleButton or ToggleButtonGadget.
state	Specifies the state of the button.
notify	Specifies whether or not the XmNvalueChangedCallback is called.

Description

`XmToggleButtonSetState`() and `XmToggleButtonGadgetSetState`() set the state of the specified *widget*. `XmToggleButtonSetState`() works when *widget* is a ToggleButton or a ToggleButtonGadget, while `XmToggleButtonGadgetSetState`() only works for a ToggleButtonGadget. In Motif 1.2 and earlier, if *state* is True, the button is selected, and when *state* is False, the button is deselected.

In Motif 2.0 and later, a Toggle can be in any of three states: XmSET, XmINDETERMINATE, and XmUNSET, where XmUNSET is equivalent to False and XmSET is equivalent to True. The third indeterminate state is enabled if the Motif 2.x XmNtoggleMode resource of the widget is set to the value XmTOGGLE_INDETERMINATE. If the toggle mode is XmTOGGLE_BOOLEAN, the widget has only two dynamic states, which is consistent with Motif 1.2 behavior.

If *notify* is True, the routines invoke the callbacks specified by the XmNvalueChangedCallback resource. If the specified *widget* is the child of a RowColumn with XmNradioBehavior set to True, the currently selected child of the RowColumn is deselected.

Usage

XmToggleButtonSetState() and XmToggleButtonGadgetSet-State() are convenience routines that set the value of the XmNset resource for a ToggleButton or ToggleButtonGadget. Calling one of the routines is equivalent to calling XtSetValues() for the resource, although the routines access the value through the widget instance structures rather than through XtSetValues().

In Motif 2.0 and later, passing the value XmINDETERMINATE is mapped to XmSET. It is therefore not possible to set the XmINDETERMINATE state using XmToggleButtonSetState(). To set a Toggle into an indeterminate state through the convenience functions, call XmToggleButtonSetValue() or XmToggleButtonGadgetSetValue().

See Also

XmToggleButtonGetState(1), XmToggleButtonSetValue(1), XmToggleButton(2), XmToggleButtonGadget(2).

Name

XmToggleButtonSetValue, XmToggleButtonGadgetSetValue – set the value of a ToggleButton.

Synopsis

```
#include <Xm/ToggleB.h>

Boolean XmToggleButtonSetValue (Widget widget, XmToggleButtonState state,
Boolean notify)

#include <Xm/ToggleBG.h>

Boolean XmToggleButtonGadgetSetValue ( Widget                widget,
                                       XmToggleButtonState   state,
                                       Boolean               notify)
```

Inputs

widget	Specifies the ToggleButton or ToggleButtonGadget.
state	Specifies the state of the button.
notify	Specifies whether or not the XmNvalueChangedCallback is called.

Availability

Motif 2.0 and later.

Description

`XmToggleButtonSetValue`() and `XmToggleButtonGadgetSetValue`() are similar to `XmToggleButtonSetState`() and `XmToggleButtonGadgetSetState`(), except that it is possible to set the ToggleButton into an XmINDETERMINATE state, provided that the Toggle is in the correct mode. If the widget has the XmNtoggleMode resource of XmTOGGLE_INDETERMINATE, the routine sets the XmNset resource of the widget to the required state, calls any XmNvalueChangedCallback procedures if *notify* is True, and then returns True. Otherwise, the function returns False.

Usage

`XmToggleButtonSetValue`() and `XmToggleButtonGadgetSetValue`() are convenience routines that set the value of the XmNset resource for a ToggleButton or ToggleButtonGadget which can display an indeterminate state. Calling one of the routines is equivalent to calling `XtSetValues`() for the resource, although the routines access the value through the widget instance structures rather than through `XtSetValues`().

Structures

The XmToggleButtonState type has the following possible values:

XmSET XmUNSET XmINDETERMINATE

See Also

`XmToggleButtonGetState`(1), `XmToggleButtonSetState`(1), `XmToggleButton`(2), `XmToggleButtonGadget`(2).

Name

XmTrackingEvent – allow for modal selection of a component.

Synopsis

#include <Xm/Xm.h>

Widget XmTrackingEvent (Widget *widget*, Cursor *cursor*, Boolean *confine_to*, XEvent **event_return*)

Inputs

widget	Specifies the widget in which the modal interaction occurs.
cursor	Specifies the cursor that is to be used as the pointer.
confine_to	Specifies whether or not the pointer is confined to widget.

Outputs

event_return	Returns the ButtonRelease or KeyRelease event.

Returns

The widget or gadget that contains the pointer or NULL if no widget or gadget contains the pointer.

Availability

Motif 1.2 and later.

Description

`XmTrackingEvent`() grabs the pointer and waits for the user to release BSelect or press and release a key, discarding all of the intervening events. The routine returns the ID of the widget or gadget containing the pointer when BSelect or the key is released and *event_return* contains the release event. If no widget or gadget contains the pointer when the release occurs, the function returns NULL. The modal interaction occurs within the specified *widget*, which is typically a top-level shell. During the interaction, *cursor* is used as the pointer shape. If *confine_to* is True, the pointer is confined to *widget* during the interaction; otherwise the pointer is not confined.

Usage

`XmTrackingEvent`() provides a way to allow a user to select a component. This modal interaction is meant to support a context-sensitive help system, where the user clicks on a widget to obtain more information about it. `XmTrackingEvent`() returns the selected widget, so that a help callback can be invoked to provide the appropriate information.

Example

The following code shows the use of XmTrackingEvent() in a routine that initiates context-sensitive help:

```
Widget toplevel, help_button;
...
XtAddCallback (help_button, XmNactivateCallback, query_for_help, toplevel);
...
void query_for_help (Widget widget, XtPointer client_data, XtPointer call_data)
{
    Cursor                  cursor;
    Widget                  top, help_widget;
    XmAnyCallbackStruct     cb;
    XtCallbackStatus        hascb;
    XEvent                  *event;

    top = (Widget) client_data;
    cursor = XCreateFontCursor (XtDisplay (top), XC_question_arrow);
    help_widget = XmTrackingEvent (top, cursor, True, &event);

    while (help_widget != NULL) {
        hascb = XtHasCallbacks (help_widget, XmNhelpCallback);

        if (hascb == XtCallbackHasSome) {
            cb.reason = XmCR_HELP;
            cb.event = event;
            XtCallCallbacks (help_widget, XmNhelpCallback, (XtPointer)
            &cb);
            help_widget = NULL;
        }
        else
            help_widget = XtParent (help_widget);
    }
}
```

See Also

XmTrackingLocate(1).

Name

XmTrackingLocate – allow for modal selection of a component.

Synopsis

Widget XmTrackingLocate (Widget *widget*, Cursor *cursor*, Boolean *confine_to*)

Inputs

widget	Specifies the widget in which the modal interaction occurs.
cursor	Specifies the cursor that is to be used as the pointer.
confine_to	Specifies whether or not the pointer is confined to widget.

Returns

The widget or gadget that contains the pointer or NULL if no widget or gadget contains the pointer.

Availability

In Motif 1.2, `XmTrackingLocate`() is obsolete. It has been superseded by `XmTrackingEvent`().

Description

`XmTrackingLocate`() grabs the pointer and waits for the user to release BSelect or press and release a key, discarding all of the intervening events. The routine returns the ID of the widget or gadget containing the pointer when BSelect or the key is released. If no widget or gadget contains the pointer when the release occurs, the function returns NULL. The modal interaction occurs within the specified *widget*, which is typically a top-level shell. During the interaction, *cursor* is used as the pointer shape. If *confine_to* is True, the pointer is confined to widget during the interaction; otherwise the pointer is not confined. `XmTrackingLocate`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

`XmTrackingLocate`() provides a way to allow a user to select a component. This modal interaction is meant to support a context-sensitive help system, where the user clicks on a widget to obtain more information about it. `XmTrackingLocate`() returns the selected widget, so that a help callback can be invoked to provide the appropriate information.

See Also

`XmTrackingEvent`(1).

Name

XmTransfer – introduction to the uniform transfer model.

Synopsis

Public Header:
<Xm/Transfer.h>

Functions/Macros:
`XmTransferDone`*()*, `XmTransferSendRequest`(), `XmTransferSetParameters`*()*, `XmTransferStartRequest`(), `XmTransferValue`*()*.

Availability

Motif 2.0 and later.

Description

Motif widgets support several methods of data transfer. Data can be transferred from a widget to the Primary or Secondary selection, the Clipboard, or, through the drag and drop mechanisms, to another widget. Up until Motif 2.0, each of these data transfer operations require a different treatment by the programmer. In Motif 2.0 and later, the Uniform Transfer Model (UTM) makes it possible to perform data transfer using any of the transfer methods using a single programming interface. UTM is designed to allow applications to use common code for all the supported data transfer requirements, and is intentionally written to ease the way in which new transfer targets can be written. Data transfer code written prior to Motif 2.0 will continue to work in newer versions of the toolkit, although all the widgets have been rewritten to internally use the UTM where appropriate.

The UTM is implemented through two new callback resources: *XmNconvertCallback*, and *XmNdestinationCallback*, which are available in the Primitive widget class (and in any derived classes), as well as in the Container, Scale, and DrawingArea widget classes. The programmer provides XmNconvertCallback and XmNdestinationCallback procedures which communicate with one another in order to negotiate the target format in which the data is required. In addition, the programmer provides a transfer procedure which performs the insertion of data in the right format into the destination widget.

An XmNconvertCallback procedure is associated with the source of the data. It is responsible for converting the data, typically the selected items of the source widget, into the format requested by the destination. It may also provide a list of the supported transfer targets requested by a XmNdestinationCallback procedure. The convert procedure transfers data to the destination widget by placing values within the XmConvertCallbackStruct structure passed as a parameter to the callback.

The XmNdestinationCallback procedure is responsible for negotiating the format in which data is required at the destination widget. It may request the set of supported formats in which the source can export the data, although the simplest procedure requests data in a specific target format. In specifying the request to the source of the data, the callback specifies a further transfer procedure which performs the actual insertion of data at the destination. The destination callback communicates with the source by issuing requests using the `XmTransferValue`() routine. If the destination callback requests the full list of supported source targets, the transfer procedure itself decides the best format for the destination, and internally issues a further `XmTransferValue`() call, requesting the data in a specific target format. The programmer will have to provide the logic which determines the best format within the transfer procedure.

Usage

It is not necessary for the programmer to provide XmNconvertCallback and XmNdestinationCallback procedures for all the targets which Motif supports. As part of the UTM, Motif widgets which support the XmNconvertCallback and XmNdestinationCallback resources also have internal routines which enable automatic data transfer of a set of built-in target types. The internal routines are implemented using the XmQTtransfer trait, which has convertProc and destinationProc methods. Where no XmNconvertCallback is supplied by the programmer, the convertProc is invoked to perform the data conversion. Similarly, in the absence of a XmNdestinationCallback, UTM calls the destinationProc. A programmer only needs to implement XmNconvertCallback or XmNdestinationCallback procedures where a new target type is being provided over and above those handled by the widget class default procedures.

Structures

A pointer to the following structure is passed to callbacks on the XmNconvertCallback list:

```
typedef struct {
    int         reason;         /* the reason that the callback is
    invoked     */
    XEvent      *event;         /* points to event that triggered call-
    back        */
    Atom        selection;      /* selection for which conversion is
    requested   */
    Atom        target;         /* the conversion target */
    XtPointer   source_data;    /* selection source information */
    XtPointer   location_data;  /* information about the data to be
    transferred */
    int         flags;          /* input status of the conversion*/
```

```
    XtPointer       parm;           /* parameter data for the target*/
    int             parm_format;    /* format of parameter data*/
    unsigned long   parm_length;    /* number of elements in parameter
    data            */
    Atom            parm_type;      /* the type of the parameter data*/
    int             status;         /* output status of the conversion*/
    XtPointer       value;          /* returned conversion data*/
    Atom            type;           /* type of conversion data returned*/
    int             format;         /* format of the conversion data*/
    unsigned long   length;         /* number of elements in the conver-
    sion data       */
} XmConvertCallbackStruct;
```

selection represents the selection for which conversion is requested. CLIPBOARD, PRIMARY, SECONDARY, or _MOTIF_DROP are the possible values (that is, one of the types of data transfer which the UTM rationalizes).

target represents the required format for the data to transfer, expressed as an Atom.

source_data provides data related to the source of the selection. If *selection* is _MOTIF_DROP, then *source_data* points to a DragContext object, otherwise it is NULL.

location_data specifies where the data to be converted is to be found. If *location_data* is NULL, conversion data is interpreted as the widget's current selection. Otherwise, the interpretation of *location_data* is widget class specific.

flags specifies the current status of the conversion. Possible values of the enumerated type:

```
    XmCONVERTING_NONE       /* unused                              */
    XmCONVERTING_PARTIAL    /* some, but not all, of target data is
    converted               */
    XmCONVERTING_SAME       /* conversion target is source of the
    transfer data           */
    XmCONVERTING_TRANSACT   /* unused                              */
```

parm contains extra data associated with *target*. If *target* is the Atom represented by _MOTIF_CLIPBOARD_TARGETS or _MOTIF_DEFERRED_CLIPBOARD_TARGETS, then *parm* is one of XmCOPY, XmMOVE, or XmLINK. *parm* is an array of data items, the number of such items is specified by *parm_length*, and the type of each item by *parm_format*.

parm_format specifies whether the data within *parm* is represented by a list of char, short, or long quantities. If *parm_format* is 0 (zero), *parm* is NULL. A *parm_format* value of 8 indicates *parm* is logically a list of char, 16 represents a list of short quantities, and 32 is for a list of long values. *parm_format* indicates logical type, and not physical implementation: a *parm_format* of 32 indicates a list of long quantities even if a particular machine has 64-bit longs.

parm_length specifies the number of data items at the *parm* address.

parm_type specifies the type of the *parm* data, expressed as an Atom. The default is XA_INTEGER.

status specifies the status of the conversion. The initial (default) value is XmCONVERT_DEFAULT, which, if unchanged by the callback, invokes any conversion procedures associated with the widget class when the callback finishes. These are the convertProc methods of any XmQTtransfer trait which the widget holds. Any converted data produced by a widget class routine overwrites any data from the callback. If the callback sets *status* to XmCONVERT_MERGE, widget class conversion procedures are invoked, merging any data so produced with conversion data from the callback. A returned *status* of XmCONVERT_REFUSE indicates that the callback terminates the conversion process without completion, and no widget class procedures are to be invoked. XmCONVERT_DONE similarly results in no widget class procedure invocation, except that the callback indicates that conversion has been successfully completed.

value is where the callback places the result of the conversion process. The default is NULL. It is assumed that any *value* specified is dynamically allocated by the programmer, although the programmer must not subsequently free the value: this is performed by the UTM. It is the responsibility of the programmer to ensure that the *type*, *format*, and *length* elements are appropriately set to match any data placed in *value*.

type specifies the logical type of any data returned within *value*, expressed as an Atom.

format specifies whether the data within *value* is represented by a list of char, short, or long quantities. If *format* is 0 (zero), *value* is NULL. A *format* value of 8 indicates *value* is logically a list of char, 16 represents a list of short quantities, and 32 is for a list of long values. *format* indicates logical type, and not physical implementation: a *format* of 32 indicates a list of long values even if they are actually 64 bits.

length specifies the number of data items at the *value* address.

Callbacks on the XmNdestinationCallback list are passed a pointer to the following structure:

```
typedef struct {
    int         reason;           /* reason that the callback is invoked       */
    XEvent      *event;           /* points to event that triggered callback   */
    Atom        selection;        /* the requested selection type, as an Atom  */
    XtEnum      operation;        /* the type of transfer requested            */
    int         flags;            /* whether destination and source are the same
    */
    XtPointer   transfer_id;      /* unique identifier for the request         */
    XtPointer   destination_data; /* information about the destination         */
    XtPointer   location_data;    /* information about the data                */
    Time        time;             /* time when transfer operation started      */
} XmDestinationCallbackStruct;
```

selection specifies, as an Atom, the type of selection for which data transfer is required. CLIPBOARD, PRIMARY, SECONDARY, or _MOTIF_DROP are the possible logical values.

operation indicates the type of data transfer operation requested. The possible values are:

```
XmCOPY      /* copy transfer */
XmMOVE      /* move transfer */
XmLINK      /* link transfer */
XmOTHER     /* information contained within destination_data element */
```

If *operation* is XmOTHER, *destination_data* contains a pointer to an XmDropProcCallbackStruct, which contains an operation element. See XmDropSite for more information concerning the XmDropProcCallbackStruct.

flags indicates whether the source of the transfer data is also the destination. Possible values are:

```
XmCONVERTING_NONE /* destination is not the source of the data */
XmCONVERTING_SAME /* destination is the source of the data     */
```

transfer_id specifies a unique identifier for the current transfer request.

destination_data specifies information about the destination of the transfer operation. If the *selection* is _MOTIF_DROP, then the callback has been invoked by an XmDropProc of the drop site, and *destination_data* contains a pointer to an XmDropProcCallbackStruct. If the *selection* is SECONDARY, *destination_data* is an Atom representing a target type into which the selection owner suggests the transfer should be converted. Otherwise, *destination_data* is NULL.

location_data determines where the data is to be transferred. The interpretation varies between the widget classes. In the Container widget, the value of *location_data* is a pointer to an XPoint structure, containing the x and y coordinates of the transfer location. If *location_data* is NULL, data is to be transferred to the current cursor location for the widget.

time is the server time when the transfer operation was initiated.

Transfer procedures are passed a pointer to the following structure: the interpretation of the elements is the same as those in the callbacks described above.

```
typedef struct {
    int             reason;         /* reason that the callback is invoked
    */
    XEvent          *event;         /* points to event that triggered callback
    */
    Atom            selection;      /* the requested selection type, as an Atom
    */
    Atom            target;         /* the conversion target
    */
    Atom            type;           /* type of conversion data returned
    */
    XtPointer       transfer_id;    /* unique identifier for the request
    */
    int             flags;          /* whether destination and source are same
    */
    int             remaining;      /* number transfers remaining for transfer_id
    */
    XtPointer       value;          /* returned conversion data
    */
    unsigned long   length;         /* number of elements in conversion data
    */
    int             format;         /* format of the conversion data
    */
} XmSelectionCallbackStruct;
```

Example

The following is a specimen XmNdestinationCallback which simply requests from the source the list of export targets, and which specifies a transfer procedure for handling the data import.

```
void destination_handler (Widget w, XtPointer client_data, XtPointer call_data)
{
```

```
    XmDestinationCallbackStruct *dptr = (XmDestinationCallbackStruct *)
    call_data;
    Atom TARGETS = XmInternAtom (XtDisplay (w), "TARGETS", False);

    /* transfer procedure will issue a subsequent request */
    /* for data in a specific target format. it receives */
    /* the list of supported targets from the source.    */
    XmTransferValue (dptr->transfer_id, TARGETS, (XtCallbackProc)
    transfer_procedure,
                        NULL, XtLastTimestampProcessed (XtDisplay (w))[1]);
}
```

The following specimen XmNconvertCallback procedure exports the selected text within a Text widget: although the convertProc method associated with the Text's XmQTtransfer trait performs this task in a modified form, the code does outline the basic structure required.

```
void convert_callback (Widget w, XtPointer client_data, XtPointer call_data)
{
    XmConvertCallbackStruct    *cptr = (XmConvertCallbackStruct *)
    call_data;
    Atom                       TARGETS, CB_TARGETS,
    SELECTED_TEXT;
    Atom                       targets[1];
    Display                    *display = XtDisplay (w);

    TARGETS = XmInternAtom (display, "TARGETS", False);
    CB_TARGETS    = XmInternAtom (display,
    "_MOTIF_CLIPBOARD_TARGETS", False);
    SELECTED_TEXT = XmInternAtom (display, "SELECTED_TEXT",
    False);

    /* if the destination has requested the list of supported targets */
    /* this is returned in the callback data                          */
    if ((cptr->target == TARGETS) || (cptr->target == CB_TARGETS)) {
        targets[0]   = SELECTED_TEXT;
        cptr->type   = XA_ATOM;
        cptr->value = (XtPointer) targets;
        cptr->length = 1;
        cptr->format = 32;
        /* merge the data with the list of targets supported by */
        /* the convertProc method of the XmQTtransfer trait */
```

1.Erroneously given as XtLastTimestampProcessed() in 2nd edition.

```
        cptr->status = XmCONVERT_MERGE;
    }
    else {
        if (cptr->target == SELECTED_TEXT) {
            char *selection = XmTextGetSelection (w);
            /* destination has requested the new target */
            cptr->value = selection;
            /* exported target is the requested target */
            cptr->type   = cptr->target;
            cptr->format = 8;
            cptr->length = (selection ? strlen (selection) : 0);
            /* conversion complete */
            cptr->status = XmCONVERT_DONE;
        }
        else {
            /* target is one this procedure is not handling        */
            /* result is either XmCONVERT_MERGE or
            XmCONVERT_DEFAULT */
            /* depending on whether we throw away results from any   */
            /* other convert callback we have registered.   */
            /* the default is XmCONVERT_DEFAULT */
            return XmCONVERT_MERGE;[1]
        }
    }
    return XmCONVERT_MERGE;[2]
}
```

The following is a specimen transfer procedure, which is registered by a XmNdestinationCallback using XmTransferValue():

```
void transfer_procedure (Widget w, XtPointer client_data, XtPointer call_data)
{
    XmSelectionCallbackStruct    *sptr = (XmSelectionCallbackStruct *)
    call_data;
    Atom                         TARGETS, CB_TARGETS,
    SELECTED_TEXT;
    Display                      *display = XtDisplay (w);
```

1.The 2nd edition gave XmCONVERT_DEFAULT as the return value here. Since certain focus operations built into the toolkit use the Uniform Transfer Model as mechanism, you need to inherit these, so XmCONVERT_MERGE is the better value. Apologies..

2.As above.

```
Atom                    *targets, choice;
int                     i;

choice = (Atom) 0;
TARGETS = XmInternAtom (display, "TARGETS", False);
CB_TARGETS = XmInternAtom (display,
"_MOTIF_CLIPBOARD_TARGETS", False);    SELECTED_TEXT =
XmInternAtom (display, "SELECTED_TEXT", False);

if (((sptr->target == TARGETS) || (sptr->target == CB_TARGETS)) &&
    (sptr->type == XA_ATOM)) {
    /* destination callback requested list of targets from the source */
    /* the source convertCallback returns the list. We now choose... */
    targets = (Atom *) sptr->value;

    for (i = 0; i < sptr->length; i++) {
        if (targets[i] == SELECTED_TEXT) {
        /* the source exports selected text. lets pick this one... */
        choice = targets[i];
        }
    }

    /* There's no selection we like... */
    if (choice == (Atom) 0) {
        XmTransferDone (sptr->transfer_id,
        XmTRANSFER_DONE_FAIL);
        return;
    }

    /* now go back to source and ask for the data in format of choice */
    /* might as well use ourself again as transfer procedure...      */
    XmTransferValue (sptr->transfer_id, choice, /* Preferred
    SELECTED_TEXT target */
                        transfer_procedure, NULL, XtLastTimestampProc-
                        essed (display)[1]);
}
else if (sptr->target == SELECTED_TEXT) {
    /* insert the selected text at our own insertion point */
    XmTextPosition pos = XmTextGetInsertionPosition (w);
    XmTextInsert (w, pos, (char *) sptr->value);
    /* all done */
```

1.Erroneously given as XtLastTimestampProcessed() in 2nd edition.

```
            XmTransferDone (sptr->transfer_id,
            XmTRANSFER_DONE_SUCCEED);
        }
    }
```

See Also

XmTransferDone(1), XmTransferSendRequest(1), XmTransferSetParameters(1), XmTransferStartRequest(1), XmTransferValue(1), XmContainer(2), XmDrawingArea(2), XmPrimitive(2), XmScale(2).

Name

XmTransferDone – complete a data transfer operation.

Synopsis

#include <Xm/Transfer.h>

void XmTransferDone (XtPointer *transfer_id*, XmTransferStatus *status*)

Inputs

transfer_id	Specifies a unique identifier for the transfer operation.
status	Specifies the completion status of the transfer.

Availability

Motif 2.0 and later.

Description

Under the Uniform Transfer Model, `XmTransferDone`() completes a data transfer operation. The procedure is called from destination callbacks or transfer procedures in order to signal the end of data transfer back to the source of the data.

transfer_id uniquely identifies a transfer operation, and the value is supplied either from the *transfer_id* element of a XmDestinationCallbackStruct passed to the destination callback, or from the *transfer_id* element of a XmSelectionCallbackStruct passed to a transfer procedure. *status* is set to indicate the status of the current transfer operation, which is notified back to the selection owner.

status is one of XmTRANSFER_DONE_FAIL, XmTRANSFER_DONE_SUCCEED, XmTRANSFER_DONE_CONTINUE, or XmTRANSFER_DONE_DEFAULT. The *status* XmTRANSFER_DONE_DEFAULT ignores all remaining queued transfer operations which may have been initiated within the destination callbacks and invokes the widget class default transfer procedures. That is, any unprocessed multiple batched requests created between `XmTransferStartRequest`() and `XmTransferSendRequest`()[1] calls are skipped. If *status* is XmTRANSFER_DONE_FAIL, the XmNtransferStatus of the current DropTransfer object is set to XmTRANSFER_FAILURE. XmTRANSFER_DONE_SUCCEED and XmTRANSFER_DONE_CONTINUE are similar, except that with XmTRANSFER_DONE_CONTINUE the owner of the selection is not notified if the target is _MOTIF_SNAPSHOT.

1.Erroneously given as XmTransferEndRequest() in 2nd edition.

Usage

The Uniform Transfer Model (UTM) enhances the Motif 1.2 data transfer mechanisms by providing a standard interface through which Drag and Drop, Primary and Secondary selection, and Clipboard data transfer is achieved both to and from a widget. The implementation of the UTM is through XmNconvertCallback and XmNdestinationCallback resource procedures. A convert callback is associated with the source of the data, and it is responsible for exporting data in the format required by the destination widget.

The destination callback is responsible for requesting data from the source in the format which it requires, and it calls the function `XmTransferValue`() to do this. The destination callback typically does not import the data directly, but specifies a transfer procedure to perform the insertion of data at the destination widget. The transfer procedure is specified by passing a routine as a parameter to the `XmTransferValue`() call. When the transfer is finished, either because it is completed or because it is aborted due to an error, the transfer procedure calls `XmTransferDone`() to return the status to the source.

Structures

The XmTransferStatus type has the following possible values:

XmTRANSFER_DONE_CONTINUE
XmTRANSFER_DONE_DEFAULT
XmTRANSFER_DONE_FAIL
XmTRANSFER_DONE_SUCCEED

Example

The following specimen transfer procedure calls `XmTransferDone`() to indicate the status of the data drop:

```
void transfer_procedure (Widget w, XtPointer client_data, XtPointer call_data)
{
    XmSelectionCallbackStruct    *sptr = (XmSelectionCallbackStruct *)
    call_data;
    Atom                         TARGETS, CB_TARGETS,
    IMPORT_FORMAT;
    Display                      *display = XtDisplay (w);
    Atom                         *targets, choice;
    int                          i;

    TARGETS = XmInternAtom (display, "TARGETS", False);
    CB_TARGETS = XmInternAtom (display,
    "_MOTIF_CLIPBOARD_TARGETS", False);
```

```
IMPORT_FORMAT = XmInternAtom (display, "IMPORT_FORMAT",
False);

if (((sptr->target == TARGETS) || (sptr->target == CB_TARGETS)) &&
    (sptr->type == XA_ATOM)) {
    /* destination callback requested list of targets from the source */
    /* the source convertCallback returns the list. We now choose... */
    targets = (Atom *) sptr->value;
    choice = (Atom) 0;

    for (i = 0; i < sptr->length; i++) {
        if (targets[i] == IMPORT_FORMAT) {
            /* the source exports our required target... */
            choice = targets[i];
        }
    }

    if (choice == (Atom) 0) {
        /* source does not export what we require        */
        /* assume destinationProc in the XmQTtransferTrait */
        /* does not either...                            */
        XmTransferDone (sptr->transfer_id,
        XmTRANSFER_DONE_FAIL);
        return;
    }

    /* now go back to source and ask for the data in format of choice */
    /* might as well use ourself again as transfer procedure...      */
    XmTransferValue (sptr->transfer_id, choice, /* IMPORT_FORMAT */
                     transfer_procedure, NULL, XtLastTimestampProc-
                     essed (display)[1]);
}
else if (sptr->target == IMPORT_FORMAT) {
    /* perform whatever is required to import sptr->value */
    ...
    /* all done */
    XmTransferDone (sptr->transfer_id,
    XmTRANSFER_DONE_SUCCEED);
}
else {
    /* wrong export target */
```

1.Erroneously given as XtLastTimestampProcessed() in 2nd edition.

```
            XmTransferDone (sptr->transfer_id, XmTRANSFER_DONE_FAIL);
        }
    }
```

See Also

XmTransferSendRequest(1), XmTransferSetParameters(1), XmTransferStartRequest(1), XmTransferValue(1), XmTransfer(1), XmDropTransfer(1).

Name

XmTransferSendRequest – send a multiple transfer request.

Synopsis

#include <Xm/Transfer.h>

void XmTransferSendRequest (XtPointer *transfer_id*, Time *time*)

Inputs

transfer_id	Specifies a unique identifier for the transfer operation.
time	Specifies the time of the transfer.

Availability

Motif 2.0 and later.

Description

In the Uniform Transfer Model, `XmTransferSendRequest`() marks the end of a series of transfer requests started by `XmTransferStartRequest`(). *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallbackStruct passed to a destination callback or transfer procedure respectively. *time* specifies the time of the XEvent which initiated the data transfer. `XtLastTimestampProcessed`() is the simplest method of specifying the time value.

Usage

The Uniform Transfer Model (UTM) enhances the Motif 1.2 data transfer mechanisms by providing a standard interface through which Drag and Drop, Primary and Secondary selection, and Clipboard data transfer is achieved both to and from a widget. The implementation of the UTM is through XmNconvertCallback and XmNdestinationCallback resource procedures. The destination callback is responsible for requesting data from the source in the format which it requires, and it calls the function `XmTransferValue`() to do this. A set of data transfer requests can be queued by wrapping the series of `XmTransferValue`() calls within `XmTransferStartRequest`() and `XmTransferSendRequest`() calls.

See Also

`XmTransferDone`(1), `XmTransferSetParameters`(1), `XmTransferStartRequest`(1), `XmTransferValue`(1), `XmTransfer`(1).

Name

XmTransferSetParameters – set parameters for next transfer

Synopsis

```
#include <Xm/Transfer.h>

void XmTransferSetParameters (  XtPointer       transfer_id,
                                XtPointer       parm,
                                int             parm_format,
                                unsigned long   parm_length,
                                Atom            parm_type)
```

Inputs

transfer_id	Specifies a unique identifier for the transfer operation.
parm	Specifies parameters to be passed to conversion routines.
parm_format	Specifies the format of data in the parm argument.
parm_length	Specifies the number of elements within the parm data.
parm_type	Specifies the type of parm.

Availability

Motif 2.0 and later.

Description

In the Uniform Transfer Model, `XmTransferSetParameters`() defines parameter data for a subsequent `XmTransferValue`() call. *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallback-Struct passed to a destination callback or transfer procedure respectively.

parm specifies parameter data to be passed to the conversion function, and the XmNconvertCallback procedures, of the source widget which owns the selection. *parm_format* specifies whether the data within parm consists of 8, 16, or 32 bit quantities. *parm_length* specifies the number of elements, of size determined by *parm_format*, which are at the address *parm*. *parm_type* specifies the logical type of *parm*, and is application specific. Neither Motif, the X toolkit, nor the X library interpret *parm_type* in any manner.

Usage

The Uniform Transfer Model enhances the Motif 1.2 data transfer mechanisms by providing a standard interface by which the source and destination of the data transfer can communicate. The programmer provides XmNdestinationCallback procedures which issue the request to transfer data from the source of the transfer by calling XmTransferValue(). Any parameterized data for the XmTransferValue() procedure is established through a prior XmTransferSetParameters() call. XmTransferSetParameters() is a convenience function which maps simply onto a X Toolkit Intrinsics XtSetSelectionParameters() call.

See Also

XmTransferDone(1), XmTransferSendRequest(1), XmTransferStartRequest(1), XmTransferValue(1), XmTransfer(1).

Name

XmTransferStartRequest – initiate a multiple data transfer request

Synopsis

#include <Xm/Transfer.h>

void XmTransferStartRequest (XtPointer *transfer_id*)

Inputs

transfer_id Specifies a unique identifier for the current data transfer operation.

Availability

Motif 2.0 and later.

Description

`XmTransferStartRequest`() initiates the start of a series of transfer requests. *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallbackStruct passed to a destination callback or transfer procedure respectively.

Usage

In Motif 2.0 and later, the Uniform Transfer Model enhances the Motif 1.2 data transfer mechanisms by providing a standard interface by which the source and destination of the data transfer can communicate. A set of data transfer requests can be queued by wrapping the series of requests within `XmTransferStartRequest`() and `XmTransferSendRequest`() calls. The procedure `XmTransferValue`() provides the data transfer requests in the queue.

See Also

`XmTransferDone`(1), `XmTransferSendRequest`(1), `XmTransferSetParameters`(1), `XmTransferValue`(1), `XmTransfer`(1).

Name

XmTransferValue – transfer data to a destination

Synopsis

```
#include <Xm/Transfer.h>

void XmTransferValue ( XtPointer      transfer_id,
                       Atom           target,
                       XtCallbackProc callback,
                       XtPointer      client_data,
                       Time           time)
```

Inputs

transfer_id	Specifies a unique identifier for the current data transfer operation.
target	Specifies the target to which the selection is to be converted.
callback	Specifies a transfer procedure to be called when the selection has been converted by the source.
client_data	Specifies application data to be passed to callback.
time	Specifies the time of the transfer.

Availability

Motif 2.0 and later.

Description

The Uniform Transfer Model (UTM) enhances the Motif 1.2 data transfer mechanisms by providing a standard interface through which Drag and Drop, Primary and Secondary selection, and Clipboard data transfer is achieved both to and from a widget. The implementation of the UTM is through XmNconvertCallback and XmNdestinationCallback resource procedures. A convert callback is associated with the source of the data, and it is responsible for exporting data in the format required by the destination. The destination callback is responsible for requesting data from the source in the format which it requires, and it calls the function `XmTransferValue`() to do this. The destination callback itself does not typically insert the transferred data into the destination widget: it specifies a transfer procedure, which performs the import, as a parameter to the `XmTransferValue`() call.

`XmTransferValue`() arranges to transfer data from the source of transfer data to the destination. *transfer_id* uniquely identifies a transfer operation, and the value is supplied from the *transfer_id* element of a XmDestinationCallbackStruct or XmSelectionCallbackStruct passed to a destination or transfer callback respectively.

target specifies the selection which is to be converted and transferred. If *target* is _MOTIF_DROP, the function invokes XmDropTransferStart() with internal transfer procedures to perform the data transfer. Otherwise, the data is extracted from the selection using XtGetSelectionValue().

callback is a transfer procedure, which is an application procedure that is called when the data is converted and available from the source. *client_data* is any data which the programmer wants to be passed to the *callback*. The *callback* is invoked with three parameters: the destination widget, the application *client_data*, and a pointer to a XmSelectionCallbackStruct.

time specifies the time of the XEvent which initiated the data transfer. XtLastTimestampProcessed() is the simplest method of specifying the time value.

Usage

XmTransferValue() is called from destination callbacks or transfer procedures to effect the actual transfer of data from the source, whether that be the clipboard or a widget. The programmer-defined callback replaces the XmNtransferProc added to a DropTransfer object, which the Uniform Transfer Model internally encapsulates and hides.

See Also

XmTransferDone(1), XmTransferSendRequest(1), XmTransferSetParameters(1), XmTransferStartRequest(1), XmTransfer(1), XmDropTransfer(2).

Name

XmTranslateKey – convert a keycode to a keysym using the default translator.

Synopsis

```
#include <Xm/Xm.h>

void XmTranslateKey (  Display     *display,
                       KeyCode     keycode,
                       Modifiers   modifiers,
                       Modifiers   *modifiers_return,
                       KeySym      *keysym_return)
```

Inputs

display Specifies a connection to an X server; returned from XOpenDisplay() or XtDisplay().

keycode Specifies the keycode that is translated.

modifiers Specifies the modifier keys that are applied to the keycode.

Outputs

modifiers_return Returns the modifiers used by the key translator to generate the keysym.

keysym_return Returns the resulting keysym.

Availability

Motif 1.2 and later.

Description

`XmTranslateKey`() is the default XtKeyProc translation procedure used by Motif applications. The routine takes a keycode and modifiers and returns the corresponding osf keysym.

Usage

The Motif toolkit uses a mechanism called *virtual bindings* to map one set of keysyms to another set. This mapping permits widgets and applications to use one set of keysyms in translation tables; applications and users can then customize the keysyms used in the translations based on the particular keyboard that is being used. Keysyms that can be used in this way are called *osf keysyms*. Motif maintains a mapping between the osf keysyms and the actual keysyms that represent keys on a particular keyboard. See the introduction to Section 2, *Motif and Xt Widget and Classes*, for more information about the mapping of osf keysyms to actual keysyms.

XmTranslateKey() is used by the X Toolkit during event processing to translate the keycode of an event to the appropriate osf keysym if there is a mapping for the keysym. The event is then dispatched to the appropriate action routine if there is a translation for the osf keysym.

If you need to provide a new translator with expanded functionality, you can call XmTranslateKey() to get the default translation. Use XtSetKeyTranslator() to register a new key translator. To reinstall the default behavior, you can call XtSetKeyTranslator() with XmTranslateKey() as the proc argument.

See Also

xmbind(4).

Name

XmUninstallImage – remove an image from the image cache.

Synopsis

Boolean XmUninstallImage (XImage **image*)

Inputs

image Specifies the image structure to be removed.

Returns

True on success or False if image is NULL or it cannot be found.

Description

`XmUninstallImage`() removes the specified *image* from the image cache. The routine returns True if it is successful. It returns False if image is NULL or if *image* is not found in the image cache.

Usage

`XmUninstallImage`() removes an image from the image cache. Once an image is uninstalled, it cannot be referenced again and a new image can be installed with the same name. If you have created any pixmaps that use the image, they are not affected by the image being uninstalled, since they are based on image data, not the image itself. After an image has been uninstalled, you can safely free the image.

See Also

`XmDestroyPixmap`(1), `XmGetPixmap`(1), `XmInstallImage`(1).

Name

XmUpdateDisplay – update the display.

Synopsis

void XmUpdateDisplay (Widget *widget*)

Inputs

widget Specifies any widget.

Description

`XmUpdateDisplay`() causes all pending exposure events to be processed immediately, instead of having them remain in the queue until all of the callbacks have been invoked.

Usage

`XmUpdateDisplay`() provides applications with a way to force an visual update of the display. Because callbacks are invoked before normal exposure processing occurs, when a menu or a dialog box is unposted, the display is not updated until all of the callbacks have been called. This routine is useful whenever a time-consuming action might delay the redrawing of the windows on the display.

See Also

`XmDisplay`(2).

Name

XmVaCreateSimpleCheckBox – create a CheckBox compound object.

Synopsis

```
Widget XmVaCreateSimpleCheckBox ( Widget          parent,
                                  String          name,
                                  XtCallbackProc  callback,
                                  ...,
                                  NULL)
```

Inputs

parent Specifies the widget ID of the parent of the new widget.

name Specifies the string name of the new widget for resource lookup.

callback Specifies the callback procedure that is called when the value of a button changes.

..., NULL A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleCheckBox`() is a RowColumn convenience routine that creates a CheckBox with ToggleButtonGadgets as its children. This routine is similar to `XmCreateSimpleCheckBox`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the CheckBox. The *callback* argument specifies the callback routine that is added to the XmNvalueChangedCallback of each ToggleButtonGadget child of the CheckBox. When the *callback* is invoked, the button number of the button whose value has changed is passed to the *callback* in the *client_data* parameter.

The name of each ToggleButtonGadget child is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the CheckBox. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaCHECKBUTTON, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaCHECKBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a ToggleButtonGadget child of the CheckBox and its associated resources. (As of Motif 1.2, all but the label argument are ignored.)

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleCheckBox`() as in the following example:

```
Widget      toplevel, check_box;
XmString    normal, bold, italic;

normal = XmStringCreateLocalized ("normal");
bold = XmStringCreateLocalized ("bold");
italic = XmStringCreateLocalized ("italic");
check_box = XmVaCreateSimpleCheckBox (toplevel, "check_box", toggled,
                                      XmVaCHECKBUTTON, normal,
                                      NULL, NULL, NULL,
                                      XmVaCHECKBUTTON, bold,
                                      NULL, NULL, NULL,
                                      XmVaCHECKBUTTON, italic,
                                      NULL, NULL, NULL,
                                      NULL);
XmStringFree (normal);
XmStringFree (bold);
XmStringFree (italic);
```

See Also

`XmCheckBox`(2), `XmRowColumn`(2), `XmToggleButtonGadget`(2).

Name

XmVaCreateSimpleMenuBar – create a MenuBar compound object.

Synopsis

Widget XmVaCreateSimpleMenuBar (Widget *parent*, char **name*,..., NULL)

Inputs

parent	Specifies the widget ID of the parent of the new widget.
name	Specifies the string name of the new widget for resource lookup.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleMenuBar`() is a RowColumn convenience routine that creates a MenuBar with CascadeButtonGadgets as its children. This routine is similar to `XmCreateSimpleMenuBar`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the MenuBar.

The name of each CascadeButtonGadget is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the MenuBar. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaCASCADEBUTTON, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget child of the MenuBar and its associated resources.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If

the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleMenuBar`() as in the following example:

```
Widget      top, mainw, menubar, fmenu, emenu;
XmString    file, edit, new, quit, cut, clear, copy, paste;

file = XmStringCreateLocalized ("File");
edit = XmStringCreateLocalized ("Edit");
menubar = XmVaCreateSimpleMenuBar (mainw, "menubar",
                                   XmVaCASCADEBUTTON,
                                   file,'F',
                                   XmVaCASCADEBUTTON,
                                   edit,'E',
                                   NULL);
XmStringFree (file);
XmStringFree (edit);

new = XmStringCreateLocalized ("New");
quit = XmStringCreateLocalized ("Quit");
fmenu = XmVaCreateSimplePulldownMenu (menubar, "file_menu", 0, file_cb,
                                      XmVaPUSHBUTTON,
                                      new,'N', NULL, NULL,
                                      XmVaSEPARATOR,
                                      XmVaPUSHBUTTON,
                                      quit,'Q', NULL, NULL,
                                      NULL);
XmStringFree (new);
XmStringFree (quit);

cut = XmStringCreateLocalized ("Cut");
copy = XmStringCreateLocalized ("Copy");
clear = XmStringCreateLocalized ("Clear");
paste = XmStringCreateLocalized ("Paste");
emenu = XmVaCreateSimplePulldownMenu (menubar, "edit_menu", 0,
cut_paste,
                                      XmVaPUSHBUTTON, cut,'C',
                                      NULL, NULL,
                                      XmVaPUSHBUTTON,
                                      copy,'o', NULL, NULL,
                                      XmVaPUSHBUTTON,
                                      paste,'P', NULL, NULL,
```

```
                                        XmVaSEPARATOR,
                                        XmVaPUSHBUTTON,
                                        clear,'l', NULL, NULL,
                                        NULL);
    XmStringFree (cut);
    XmStringFree (clear);
    XmStringFree (copy);
    XmStringFree (paste);
```

See Also

`XmCascadeButtonGadget`(2), `XmMenuBar`(2), `XmRowColumn`(2).

Name

XmVaCreateSimpleOptionMenu – create an OptionMenu compound object.

Synopsis

```
Widget XmVaCreateSimpleOptionMenu (  Widget        parent,
                                     String        name,
                                     XmString      option_label,
                                     KeySym
option_mnemonic,
                                     int           button_set,
                                     XtCallbackProc callback,
                                     ...,
                                     NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the new widget.
name	Specifies the string name of the new widget for resource lookup.
option_label	Specifies the label used for the OptionMenu.
option_mnemonic	Specifies the mnemonic character associated with the OptionMenu.
button_set	Specifies the initial setting of the OptionMenu.
callback	Specifies the callback procedure that is called when a button is activated.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleOptionMenu`() is a RowColumn convenience routine that creates an OptionMenu along with its submenu of CascadeButtonGadget and/or PushButtonGadget children. This routine is similar to `XmCreateSimpleOptionMenu`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the OptionMenu.

The *option_label*, *option_mnemonic*, and *button_set* arguments are used to set the XmNlabelString, XmNmnemonic, and XmNmenuHistory resources of the RowColumn respectively. The *button_set* parameter specifies the *n*th button child of the OptionMenu, where the first button is button 0 (zero); the XmNmenuHistory resource is set to the actual widget. The *callback* argument specifies the call-

back routine that is added to the XmNactivateCallback of each CascadeButtonGadget and PushButtonGadget child in the submenu of the OptionMenu. When the *callback* is invoked, the button number of the button whose value has changed is passed to the *callback* in the *client_data* parameter.

The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the submenu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 (zero) to 1 less than the number of separators in the submenu. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaPUSHBUTTON, XmVaCASCADEBUTTON, XmVaSEPARATOR, XmVaDOUBLE_SEPARATOR, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaPUSHBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a PushButtonGadget in the pulldown submenu of the OptionMenu and its associated resources. If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget in the pulldown submenu of the OptionMenu and its associated resources. If the first argument in a group is XmVaSEPARATOR or XmVaDOUBLE_SEPARATOR, it is not followed by any arguments. These groups specify SeparatorGadgets in the pulldown submenu of the OptionMenu.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleOptionMenu`() as in the following example:

```
Widget      rc, option_menu;
XmString    draw_shape, line, square, circle;
```

```
draw_shape = XmStringCreateLocalized ("Draw Mode:");
line = XmStringCreateLocalized ("Line");
square = XmStringCreateLocalized ("Square");
circle = XmStringCreateLocalized ("Circle");
option_menu = XmVaCreateSimpleOptionMenu (rc, "option_menu",
draw_shape, 'D', 0, option_cb,
                                          XmVaPUSHBUTTON, line,
                                          'L', NULL, NULL,
                                          XmVaPUSHBUTTON, square,
                                          'S', NULL, NULL,
                                          XmVaPUSHBUTTON, circle,
                                          'C', NULL, NULL,
                                          NULL);
XmStringFree (line);
XmStringFree (square);
XmStringFree (circle);
XmStringFree (draw_shape);
```

See Also

`XmOptionButtonGadget`(1), `XmOptionLabelGadget`(1), `XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmOptionMenu`(2), `XmPushButtonGadget`(2), `XmRowColumn`(2), `XmSeparatorGadget`(2).

Name

XmVaCreateSimplePopupMenu – create a PopupMenu compound object as the child of a MenuShell.

Synopsis

```
Widget XmVaCreateSimplePopupMenu ( Widget          parent,
                                   String          name,
                                   XtCallbackProc  callback,
                                   ...,
                                   NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the MenuShell.
name	Specifies the string name of the new widget for resource lookup.
callback	Specifies the callback procedure that is called when a button is activated or its value changes.
...,NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimplePopupMenu`() is a RowColumn convenience routine that creates a PopupMenu along with its button children. The routine creates the PopupMenu as a child of a MenuShell. This routine is similar to `XmCreateSimplePopupMenu`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the PopupMenu. The *callback* argument specifies the callback routine that is added to the XmNactivateCallback of each CascadeButtonGadget and PushButtonGadget child and the XmNvalueChangedCallback of each ToggleButtonGadget child in the PopupMenu. When the callback is invoked, the button number of the button whose value has changed is passed to the callback in the *client_data* parameter.

The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the menu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 (zero) to 1 less than the number of separators in the menu. The name of each title is label_*n*, where *n* is the number of the title, ranging from 0 (zero) to 1 less than the number of titles in the menu. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaPUSHBUTTON, XmVaCASCADEBUTTON, XmVaRADIOBUTTON, XmVaCHECKBUTTON, XmVaTITLE, XmVaSEPARATOR, XmVaDOUBLE_SEPARATOR, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaPUSHBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a PushButtonGadget child of the PopupMenu and its associated resources. If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget child of the PopupMenu and its associated resources. If the first argument in a group is XmVaRADIOBUTTON or XmVaCHECKBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. These groups specify ToggleButtonGadget children of the PopupMenu and their associated resources.

If the first argument is XmVaTITLE, it is followed by a title argument. This group specifies a LabelGadget title in the PopupMenu and its associated resource. If the first argument in a group is XmVaSEPARATOR or XmVaDOUBLE_SEPARATOR, it is not followed by any arguments. These groups specify SeparatorGadgets in the PopupMenu.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimplePopupMenu`() as in the following example:

```
Widget        drawing_a, popup_menu;
XmString      line, square, circle, quit, quit_acc;

line = XmStringCreateLocalized ("Line");
square = XmStringCreateLocalized ("Square");
circle = XmStringCreateLocalized ("Circle");
quit = XmStringCreateLocalized ("Quit");
```

```
quit_acc = XmStringCreateLocalized ("Ctrl-C");
popup_menu = XmVaCreateSimplePopupMenu (drawing_a, "popup", popup_cb,
                                        XmVaPUSHBUTTON, line, NULL, NULL,
                                        NULL,
                                        XmVaPUSHBUTTON, square, NULL,
                                        NULL, NULL,
                                        XmVaPUSHBUTTON, circle, NULL,
                                        NULL, NULL,
                                        XmVaSEPARATOR,
                                        XmVaPUSHBUTTON, quit, NULL,
                                        "Ctrl<Key>c", quit_acc,
                                        NULL);
XmStringFree (line);
XmStringFree (square);
XmStringFree (circle);
XmStringFree (quit);
XmStringFree (quit_acc);
```

See Also

`XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmMenuShell`(2), `XmPopupMenu`(2), `XmPushButtonGadget`(2), `XmRowColumn`(2), `XmSeparatorGadget`(2), `XmToggleButtonGadget`(2).

Name

XmVaCreateSimplePulldownMenu – create a PulldownMenu compound object as the child of a MenuShell.

Synopsis

```
Widget XmVaCreateSimplePulldownMenu (  Widget            parent,
                                       String            name,
                                       int
post_from_button,
                                       XtCallbackProc    callback,
                                       ...,
                                       NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the MenuShell.
name	Specifies the string name of the new widget for resource lookup.
post_from_button	Specifies the CascadeButton or CascadeButtonGadget in the parent widget to which the menu is attached.
callback	Specifies the callback procedure that is called when a button is activated or its value changes.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimplePulldownMenu`() is a RowColumn convenience routine that creates a PulldownMenu along with its button children. The routine creates the PulldownMenu as a child of a MenuShell. This routine is similar to `XmCreateSimplePulldownMenu`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the PulldownMenu.

The *post_from_button* parameter specifies the CascadeButton or CascadeButtonGadget to which the PulldownMenu is attached as a submenu. The argument specifies the nth CascadeButton or CascadeButtonGadget, where the first button is button 0 (zero). The *callback* argument specifies the callback routine that is added to the XmNactivateCallback of each CascadeButtonGadget and PushButtonGadget child and the XmNvalueChangedCallback of each ToggleButtonGadget child in the PulldownMenu. When the *callback* is invoked, the button

number of the button whose value has changed is passed to the callback in the *client_data* parameter.

The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the menu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 (zero) to 1 less than the number of separators in the menu. The name of each title is label_*n*, where *n* is the number of the title, ranging from 0 (zero) to 1 less than the number of titles in the menu. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaPUSHBUTTON, XmVaCASCADEBUTTON, XmVaRADIOBUTTON, XmVaCHECKBUTTON, XmVaTITLE, XmVaSEPARATOR, XmVaDOUBLE_SEPARATOR, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaPUSHBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a PushButtonGadget child of the PulldownMenu and its associated resources. If the first argument in a group is XmVaCASCADEBUTTON, it is followed by two arguments: label and mnemonic. This group specifies a CascadeButtonGadget child of the PulldownMenu and its associated resources. If the first argument in a group is XmVaRADIOBUTTON or XmVaCHECKBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. These groups specify ToggleButtonGadget children of the PulldownMenu and their associated resources.

If the first argument is XmVaTITLE, it is followed by a title argument. This group specifies a LabelGadget title in the PulldownMenu and its associated resource. If the first argument in a group is XmVaSEPARATOR or XmVaDOUBLE_SEPARATOR, it is not followed by any arguments. These groups specify SeparatorGadgets in the PulldownMenu.

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use XmVaCreateSimplePulldownMenu() as in the following example:

```
Widget      top, mainw, menubar, fmenu, emenu;
XmString    file, edit, new, quit, cut, clear, copy, paste;

file = XmStringCreateLocalized ("File");
edit = XmStringCreateLocalized ("Edit");
menubar = XmVaCreateSimpleMenuBar (mainw, "menubar",
                                   XmVaCASCADEBUTTON, file, 'F',
                                   XmVaCASCADEBUTTON, edit, 'E',
                                   NULL);
XmStringFree (file);
XmStringFree (edit);

new = XmStringCreateLocalized ("New");
quit = XmStringCreateLocalized ("Quit");
fmenu = XmVaCreateSimplePulldownMenu (menubar, "file_menu", 0, file_cb,
                                      XmVaPUSHBUTTON, new,
                                      'N', NULL, NULL,
                                      XmVaSEPARATOR,
                                      XmVaPUSHBUTTON, quit,
                                      'Q', NULL, NULL,
                                      NULL);
XmStringFree (new);
XmStringFree (quit);

cut = XmStringCreateLocalized ("Cut");
copy = XmStringCreateLocalized ("Copy");
clear = XmStringCreateLocalized ("Clear");
paste = XmStringCreateLocalized ("Paste");
emenu = XmVaCreateSimplePulldownMenu (menubar, "edit_menu", 1,
cut_paste,
                                      XmVaPUSHBUTTON, cut, 'C',
                                      NULL, NULL,
                                      XmVaPUSHBUTTON, copy, 'o',
                                      NULL, NULL,
                                      XmVaPUSHBUTTON, paste,
                                      'P', NULL, NULL,
                                      XmVaSEPARATOR,
                                      XmVaPUSHBUTTON, clear, 'l',
                                      NULL, NULL,
                                      NULL);
```

```
XmStringFree (cut);
XmStringFree (clear);
XmStringFree (copy);
XmStringFree (paste);
```

See Also

XmCascadeButtonGadget(2), XmLabelGadget(2), XmMenuShell(2), XmPulldownMenu(2), XmPushButtonGadget(2), XmRowColumn(2), XmSeparatorGadget(2), XmToggleButtonGadget(2).

Name

XmVaCreateSimpleRadioBox – create a RadioBox compound object.

Synopsis

```
Widget XmVaCreateSimpleRadioBox (  Widget          parent,
                                   String          name,
                                   int             button_set,
                                   XtCallbackProc  callback,
                                   ...,
                                   NULL)
```

Inputs

parent	Specifies the widget ID of the parent of the new widget.
name	Specifies the string name of the new widget for resource lookup.
button_set	Specifies the initial setting of the RadioBox.
callback	Specifies the callback procedure that is called when the value of a button changes.
..., NULL	A NULL-terminated variable-length list of resource name/value pairs.

Returns

The widget ID of the RowColumn widget.

Description

`XmVaCreateSimpleRadioBox`() is a RowColumn convenience routine that creates a RadioBox with ToggleButtonGadgets as its children. This routine is similar to `XmCreateSimpleRadioBox`(), but it uses a NULL-terminated variable-length argument list in place of the arglist and argcount parameters. The variable-length argument list specifies resource name/value pairs as well as the children of the CheckBox. The *button_set* argument is used to set the XmNmenuHistory resource of the RowColumn. The parameter specifies the *n*th button child of the RadioBox, where the first button is button 0 (zero); the XmNmenuHistory resource is set to the actual widget. The *callback* argument specifies the callback routine that is added to the XmNvalueChangedCallback of each ToggleButtonGadget child of the RadioBox. When the *callback* is invoked, the button number of the button whose value has changed is passed to the *callback* in the *client_data* parameter.

The name of each ToggleButtonGadget child is button_*n*, where *n* is the number of the button, ranging from 0 (zero) to 1 less than the number of buttons in the RadioBox. The buttons are created and named in the order in which they are specified in the variable-length argument list.

Usage

A variable-length argument list is composed of several groups of arguments. Within each group, the first argument is a constant or a string that specifies which arguments follow in the group. The first argument can be one of the following values: XmVaRADIOBUTTON, a resource name, XtVaTypedList, or XtVaNestedList. The variable-length argument list must be NULL-terminated.

If the first argument in a group is XmVaRADIOBUTTON, it is followed by four arguments: label, mnemonic, accelerator, and accelerator text. This group specifies a ToggleButtonGadget child of the RadioBox and its associated resources. (As of Motif 1.2, all but the label argument are ignored.)

If the first argument in a group is a resource name string, it is followed by a resource value of type XtArgVal. This group specifies a standard resource name/value pair for the RowColumn widget. If the first argument in a group is XtVaTypedArg, it is followed by four arguments: name, type, value, and size. This group specifies a resource name and value using the standard XtVaTypedArg format. If the first argument in a group is XtVaNestedList, it is followed by one argument of type XtVarArgsList, which is returned by `XtVaCreateArgsList`().

Example

You can use `XmVaCreateSimpleRadioBox`() as in the following example:

```
Widget      toplevel, radio_box;
XmString    one, two, three;

one = XmStringCreateLocalized ("WFNX");
two = XmStringCreateLocalized ("WMJX");
three = XmStringCreateLocalized ("WXKS");
radio_box = XmVaCreateSimpleRadioBox (toplevel, "radio_box", 0, toggled,
                                      XmVaRADIOBUTTON, one, NULL,
                                      NULL, NULL,
                                      XmVaRADIOBUTTON, two, NULL,
                                      NULL, NULL,
                                      XmVaRADIOBUTTON, three,
                                      NULL, NULL, NULL,
                                      NULL);
XmStringFree (one);
XmStringFree (two);
XmStringFree (three);
```

See Also

`XmRadioBox`(2), `XmRowColumn`(2), `XmToggleButtonGadget`(2).

Name

XmWidgetGetBaselines – get the positions of the baselines in a widget.

Synopsis

Boolean XmWidgetGetBaselines (Widget *widget*, Dimension **_baselines_, int *_line_count_)

Inputs

widget Specifies the widget for which to get baseline values.

Outputs

baselines Returns an array containing the value of each baseline of text in the widget.

line_count Returns the number of lines of text in the widget.

Returns

True if the widget contains at least one baseline or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmWidgetGetBaselines`() returns an array that contains the baseline values for the specified *widget*. For each line of text in the widget, the baseline value is the vertical offset in pixels from the origin of the bounding box of the widget to the text baseline. The routine returns the baseline values in *baselines* and the number of lines of text in the widget in *line_count*. `XmWidgetGetBaselines`() returns True if the *widget* contains at least one line of text and therefore has a baseline. If the *widget* does not contain any text, the routine returns False and the values of *baselines* and *line_count* are undefined. The routine allocates storage for the returned values. The application is responsible for freeing this storage using `XtFree`().

Usage

XmWidgetGetBaselines() provide information that is useful when you are laying out an application and trying to align different components.

See Also

`XmWidgetGetDisplayRect`(1).

Name

XmWidgetGetDisplayRect – get the display rectangle for a widget.

Synopsis

Boolean XmWidgetGetDisplayRect (Widget *widget*, XRectangle **displayrect*)

Inputs

widget Specifies the widget for which to get the display rectangle.

Outputs

displayrect Returns an XRectangle that specifies the display rectangle of the widget.

Returns

True if the widget has a display rectangle or False otherwise.

Availability

Motif 1.2 and later.

Description

`XmWidgetGetDisplayRect`() gets the display rectangle for the specified *widget*. The routine returns the width, the height, and the x and y-coordinates of the upper left corner of the display rectangle in the *displayrect* XRectangle. All of the values are specified as pixels. The display rectangle for a widget is the smallest rectangle that encloses the string or the pixmap in the widget. `XmWidgetGetDisplayRect`() returns True if the widget has a display rectangle; other it returns False and the value of *displayrect* is undefined.

Usage

`XmWidgetGetDisplayRect`() provide information that is useful when you are laying out an application and trying to align different components.

See Also

`XmWidgetGetBaselines`(1).

Section 2 - Motif and Xt Widget Classes

This page describes the format and contents of each reference page in Section 2, which covers each of the Motif and Xt Intrinsics widget types.

Name

Widget – a brief description of the widget.

Synopsis

Public Headers:
The files to include when you use this widget.

Class Name:
The name of the widget class; used as the resource class for each instance of the widget.

Class Hierarchy:
The superclasses of this widget, listed in superclass-to-subclass order. The arrow symbol (→) indicates a subclass.

Class Pointer:
The global variable that points to the widget class structure. This is the value used when creating a widget.

Instantiation:
C code that instantiates the widget, for widgets that can be instantiated. For the widgets and gadgets in the Motif toolkit, we have shown how to instantiate the widget using `XtCreateWidget`(). Each widget and gadget has a convenience creation routine of the general form:

```
Widget XmCreateobject ( Widget    parent
                        String    name
                        ArgList   arglist,
                        Cardinal  argcount)
```

where *object* is the shorthand for the class.

Functions/Macros:
Functions and/or macros specific to this widget class.

Availability

This section describes the availability of the widget class across various versions of Motif. The section is omitted if the widget class has always been present in the toolkit.

Description

This section gives an overview of the widget class and the functionality it provides.

Traits

This section appears for any traits that are set by the widget class. The Trait mechanisms are available in Motif 2.0 and later.

New Resources

This section presents a table of the resources that are newly defined by each widget class (not inherited from a superclass). In addition to the resource's name, class, data type, and default value, a fifth column lists a code consisting of one or more of the letters C, S, and G. This code indicates whether the resource can be set when the widget is created (C), whether it can be set with XtSetValues() (S), and whether it can be read with `XtGetValues()` (G). A brief description of each new resource follows the table. For resources whose values are defined constants, these constants are listed. Unless otherwise noted, they are defined in `<Xm/Xm.h>`.

Other New Resources

If present, these sections describe resources associated with specific uses of the widget; for example, RowColumn widget resources for use with simple creation routines, or Text widget resources for use in text input.

Callback Resources

This section presents a table of the callback resources that are newly defined by this class. The table lists the name of each resource along with its reason constant.

Callback Structure

This section lists the structure(s) associated with the object's callback functions.

New Constraint Resources

This section defines any constraint resources that are newly defined by each widget class (not inherited from a superclass). In addition to the resource's name, class, data type, and default value, a fifth column lists a code consisting of one or more of the letters C, S, and G. This code indicates whether the constraint resource can be set when a child widget is created (C), whether it can be set with XtSetValues() (S), and whether it can be read with `XtGetValues()` (G). A brief description of each new constraint resource follows the table. For resources whose values are defined constants, these constants are listed. Unless otherwise noted, they are defined in `<Xm/Xm.h>`.

Procedures

This section lists any procedure or function prototypes associated with the widget.

Default Resource Values

This section presents a table of the default resource values that are set when a compound object is created.

Inherited Resources

This section presents an alphabetically arranged table of inherited resources, along with the superclass that defines them.

Widget Hierarchy

This section presents the widget instance hierarchy that results from creating a compound object.
The full widget hierarchy is shown in Figure 1.

Translations

This section presents the translations associated with each widget or gadget. Because the button events and key events used in Motif do not necessarily correspond to the events in the X Window System, the Motif toolkit has created a mechanism called *virtual bindings*. Virtual bindings link the translations used in Motif to their X event counterparts. The "Translations" sections list their events in terms of these virtual bindings. In order to understand the syntax used in the "Translations" sections of these reference pages, you must understand the correspondence between virtual bindings and actual keysyms or buttons. The following tables describe the virtual bindings of events.

Virtual Modifier	Actual Modifier
MAlt	<Mod1>
MCtrl	<Ctrl>
MShift	<Shift>
MLink	<Ctrl><Shift>
MMove	<Shift>
MCopy	<Ctrl>

Virtual Button	Actual Button Events
BCustom	<Btn3>
BTransfer	<Btn2>
BExtend	<Shift><Btn1>
BMenu	<Btn3>
BSelect	<Btn1>

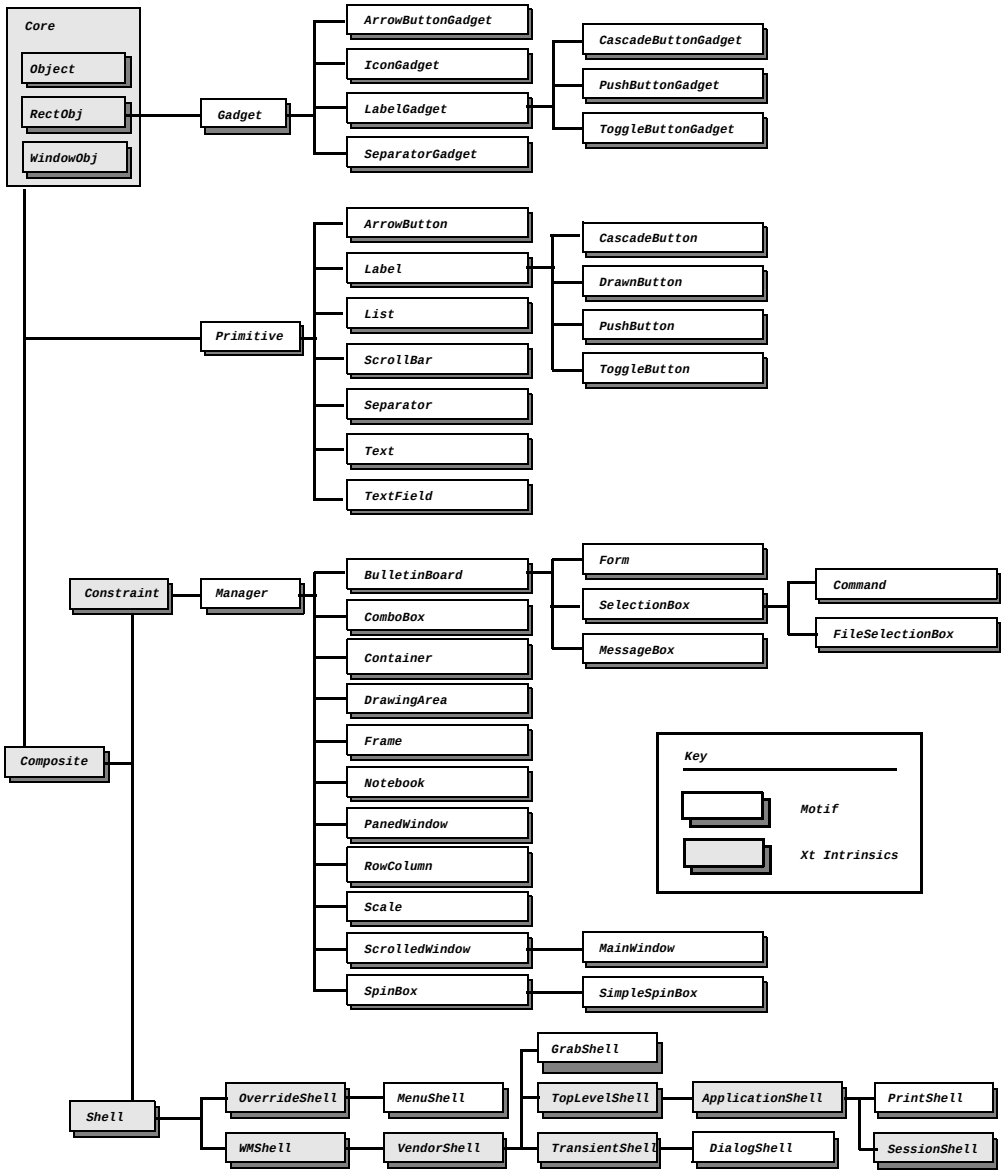

Figure 1: Class Hierarchy of the Motif widget set

Virtual Button	Actual Button Events
BToggle	<Ctrl><Btn1>

Virtual Key	Actual Key Events
KActivate	<Key>Return <Ctrl><Key>Return <Key>osfActivate
KAddMode	<Key>osfAddMode
KBackSpace	<Key>osfBackSpace
KBackTab	<Shift><Key>Tab
KBeginData	<Ctrl><Key>osfBeginLine
KBeginLine	<Key>osfBeginLine
KCancel	<Key>osfCancel
KClear	<Key>osfClear
KCopy	<Key>osfCopy <Ctrl><Key>osfInsert
KCut	<Key>osfCut <Shift><Key>osfDelete
KDelete	<Key>osfDelete
KDeselectAll	<Ctrl><Key>backslash
KDown	<Key>osfDown
KEndData	<Ctrl><Key>osfEndLine
KEndLine	<Key>osfEndLine
KEnter	<Key>Return
KEscape	<Key>Escape
KExtend	<Ctrl><Shift><Key>space <Shift><Key>osfSelect
KHelp[	<Key>osfHelp
KInsert	<Key>osfInsert
KLeft	<Key>osfLeft
KMenu	<Key>osfMenu
KMenuBar	<Key>osfMenuBar
KNextField	<Key>Tab <Ctrl><Key>Tab

Virtual Key	Actual Key Events
KNextMenu	<Ctrl><Key>osfDown <Ctrl><Key>osfRight
KPageDown	<Key>osfPageDown
KPageLeft	<Ctrl<Key>osfPageUp <Key>osfPageUp
KPageRight	<Ctrl><Key>osfPageDown
KPageUp	<Key>osfPageUp
KPaste	<Key>osfPaste <Shift><Key>osfInsert
KPrevField	<Shift><Key>Tab <Ctrl><Key><Shift><Tab>
KPrevMenu	<Ctrl><Key>osfUp <Ctrl><Key>osfLeft
KPrimaryCopy	<Ctrl><Key>osfPrimaryPaste <Mod1><Key>osfCopy <Mod1><Ctrl><Key>osfInsert
KPrimaryCut	<Mod1><Key>osfPrimaryPaste <Mod1><Key>osfCut <Mod1><Shift><Key>osfDelete
KPrimaryPaste	<Key>osfPrimaryPaste
KQuickCopy	<Ctrl><Key>osfQuickPaste
KQuickCut	<Mod1><Key>osfQuickPaste
KQuickExtend	<Shift><Key>osfQuickPaste
KQuickPaste	<key>osfQuickPaste
KReselect	<Ctrl><Shift><Key>osfSelect
KRestore	<Ctrl><Shift><Key>osfInsert
KRight	<Key>osfRight
KSelect	<Key>space <Ctrl><Key>space <Key>osfSelect
KSelectAll	<Ctrl><Key>slash
KSpace	<Key>space
KTab	<Key>Tab
KUndo	<Key>osfUndo <Mod1><Key>osfBackSpace

Virtual Key	Actual Key Events
KUp	<Key>osfUp
KAny	<Key>

Keysyms that begin with the letters osf are not defined by the X server. These keysyms are generated at run time by a client, interpreted by `XmTranslateKey()`, and used by the translation manager when the server sends an actual key event. An application maintains a mapping between osf keysyms and actual keysyms that is based on information that is retrieved at application startup. This information comes from one of the following sources, listed in order of precedence:

• The `XmNdefaultVirtualBindings` resource in a resource database. A sample specification is shown below:

```
*defaultVirtualBindings:\
osfBackSpace:       <Key>BackSpace \n\
osfInsert:          <Key>InsertChar \n\
osfDelete:          <Key>DeleteChar
```

• A property on the root window. *mwm* sets this property on startup. It can also be set by the *xmbind* client in Motif 1.2 or later, or the prior startup of another Motif application.

• A file named *.motifbind*, in the user's home directory. In this file, the previous specification would be typed as follows:

```
osfBackSpace:       <Key>BackSpace
osfInsert:          <Key>InsertChar
osfDelete:          <Key>DeleteChar
```

• A vendor-specific set of bindings located using the file *xmbind.alias*. If this file exists in the user's home directory, it is searched for a pathname associated with the vendor string or the vendor string and vendor release. If the search is unsuccessful, Motif continues looking for *xmbind.alias* in the directory specified by XMBINDDIR or in */usr/lib/Xm/bindings* if the variable is not set. If this file exists, it is searched for a pathname as before. If either search locates a pathname and the file exists, the bindings in that file are used. An *xmbind.alias* file contains lines of the following form:

```
"vendor_string[vendor_release]"bindings_file
```

• Via fixed fallback defaults. *osf* keysym strings have the fixed fallback default bindings listed below:

```
osfActivate         <unbound>
```

osfAddMode	<Shift> F8
osfBackSpace	Backspace
osfBeginLine	Home
osfClear	Clear
osfCopy	<unbound>
osfCut	<unbound>
osfDelete	Delete
osfDown	Down
osfEndLine	End
osfCancel	<Escape>
osfHelp	F1
osfInsert	Insert
osfLeft	Left
osfMenu	F4
osfMenuBar	F10
osfPageDown	Next
osfPageLeft	<unbound>
osfPageRight	<unbound>
osfPageUp	Prior
osfPaste	<unbound>
osfPrimaryPaste	<unbound>
osfQuickPaste	<unbound>
osfRight	Right
osfSelect	Select
osfUndo	Undo
osfUp	Up

Action Routines

This section describes the action routines that are listed in the "Translations" section.

Behavior

This section describes the keyboard and mouse events that affect gadgets, which do not have translations or actions.

Additional Behavior

This section describes any additional widget behavior that is not provided by translations and actions.

See Also

This section refers you to related functions and widget classes. The numbers in parentheses following each reference refer to the sections of this book in which they are found.

Name

ApplicationShell widget class – the main shell for an application.

Synopsis

Public Headers:

<Xm/Xm.h>
<X11/Shell.h>

Class Name:

ApplicationShell

Class Hierarchy:

Core →Composite →Shell →WMShell →VendorShell →TopLevelShell → ApplicationShell

Class Pointer:

`applicationShellWidgetClass`

Instantiation:

```
widget = XtAppInitialize (...)
```

or

```
widget = XtAppCreateShell (app_name, app_class,
                           applicationShellWidget-
                           Class,...)
```

Functions/Macros:

`XtAppCreateShell()`, `XtVaAppCreateShell()`, `XtIsApplica-tionShell()`

Availability

From X11R6, the ApplicationShell is considered deprecated: you should give preference to the SessionShell widget class.

Description

An ApplicationShell is the normal top-level window for an application. It does not have a parent and it is at the root of the widget tree. An application should have only one ApplicationShell, unless the application is implemented as multiple logical applications. Normally, an application will use TopLevelShell widgets for other top-level windows.

An ApplicationShell is returned by the call to `XtVaAppInitialize()`. It can also be created explicitly with a call to `XtVaAppCreateShell()`.

New Resources

ApplicationShell defines the following resources:

Name	Class	Type	Default	Access
XmNargc	XmCArgc	int	0	CSG
XmNargv	XmCArgv	String *	NULL	CSG

`XmNargc`
Number of arguments in `XmNargv`.

`XmNargv`
List of command-line arguments used to start the application. This is the standard C *argv*, passed in the call to `XtAppInitialize()`. It is used to set the WM_COMMAND property for this window, which is the argument list required by a session manager to restart the application if necessary. The resource value can be changed at appropriate points if some specific internal state has been reached from which the application can be directly restarted.

Inherited Resources

ApplicationShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of `XmNborderWidth` is reset to 0 by VendorShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNmappedWhenManaged	Core
XmNallowShellResize	Shell	XmNmaxAspectX	WMShell
XmNancestorSensitive	Core	XmNmaxAspectY	WMShell
XmNaudibleWarning	VendorShell	XmNmaxHeight	WMShell
XmNbackground	Core	XmNmaxWIdth	WMShell
XmNbackgroundPixmap	Core	XmNminAspectX	WMShell
XmNbaseHeight	WMShell	XmNminAspectY	WMShell
XmNbaseWidth	WMShell	XmNminHeight	WMShell
XmNborderColor	Core	XmNminWidth	WMShell
XmNborderPixmap	Core	XmNmwmDecorations	VendorShell
XmNborderWidth	Core	XmNmwmFunctions	VendorShell
XmNbuttonFontList	VendorShell	XmNmwmInputMode	VendorShell
XmNbuttonRenderTable	VendorShell	XmNmwmMenu	VendorShell
XmNchildren	Composite	XmNnumChildren	Composite
XmNcolormap	Core	XmNoverrideRedirect	Shell

Resource	Inherited From	Resource	Inherited From
XmNcreatePopupChildProc	Shell	XmNpopdownCalback	Shell
XmNdefaultFontList	VendorShell	XmNpopupCallback	Shell
XmNdeleteResponse	VendorShell	XmNpreeditType	VendorShell
XmNdepth	Core	XmNsaveUnder	Shell
XmNdestroyCallback	Core	XmNscreen	Core
XmNgeometry	Shell	XmNsensitive	Core
XmNheight	Core	XmNshellUnitType	VendorShell
XmNheightInc	WMShell	XmNtextFontList	VendorShell
XmNiconic	TopLevelShell	XmNtextRenderTable	VendorShell
XmNiconMask	WMShell	XmNtitle	WMShell
XmNiconName	TopLevelShell	XmNtitleEncoding	WMShell
XmNiconNameEncoding	TopLevelShell	XmNtransient	WMShell
XmNiconPixmap	WMShell	XmNtranslations	Core
XmNiconWindow	WMShell	XmNuseAsyncGeometry	VendorShell
XmNinitialResourcesPersistent	Core	XmNunitType	VendorShell
XmNinitialState	WMShell	XmNvisual	Shell
XmNinput	WMShell	XmNwaitForWm	WMShell
XmNinputMethod	VendorShell	XmNwidth	Core
XmNinputPolicy	VendorShell	XmNwidthInc	WMShell
XmNinsertPosition	Composite	XmNwindowGroup	WMShell
XmNkeyboardFocusPolicy	VendorShell	XmNwinGravity	WMShell
XmNlabelFontList	VendorShell	XmNwmTimeout	WMShell
XmNlabelRenderTable	VendorShell	XmNx	Core
XmNlayoutDirection	VendorShell	XmNy	Core

The VendorShell superclass installs a handler which intercepts the window manager WM_DELETE_WINDOW message. The handler is inherited by sub-classes of VendorShell, and has the behavior that if XmNdeleteResponse is XmDESTROY, and the widget is an instance of an ApplicationShell, then the application context associated with the widget is destroyed, followed by a call to exit().

See Also

`Composite`(2), `Core`(2), `SessionShell`(2), `Shell`(2), `TopLevelShell`(2), `VendorShell`(2), `WMShell`(2).

Name

Composite widget class – the fundamental widget that can have children.

Synopsis

Public Headers:

<Xm/Xm.h>
<X11/Composite.h>

Class Name:

Composite

Class Hierarchy:

Core →Composite

Class Pointer:

compositeWidgetClass

Instantiation:

Composite is an Intrinsics meta-class and is not normally instantiated.

Functions/Macros:

`XtIsComposite()`

Description

Composite widgets contain other widgets. A Composite widget supports an arbitrary number of children, although derived classes may impose a limit for whatever reason. Composite handles the geometry management of its children. It also manages the destruction of descendants when it is destroyed. Children of a Composite widget are ordered, and Composite provides the means to sort or place the list of children in some logical order.

New Resources

Composite defines the following resources:

Name	Class	Type	Default	Access
XmNchildren	XmCReadOnly	WidgetList	NULL	G
XmNinsertPosition	XmCInsertPosition	XtOrderProc	NULL	CSG
XmNnumChildren	XmCReadOnly	Cardinal	0	G

XmNchildren

List of widget's children.

XmNinsertPosition

Points to an `XtOrderProc()` function that is called to determine the position at which each child is inserted into the XmNchildren array. Composite supplies a default function that appends children in the order of creation.

XmNnumChildren
Length of the list in XmNchildren.

Procedures

XtOrderProc
An `XtOrderProc` is a pointer to a function, specified as follows:

```
typedef Cardinal (*XtOrderProc) (Widget);
```

An `XtOrderProc` function is called by the insert_child method of a Composite or derived class, when a new child is created within the widget. The function has a single parameter, which is the widget ID of the new child. The function returns the number of children that go before the new child in the XmNchildren array. Composite supplies a default function that simply appends new children in the order of creation. Sub-classes may supply alternative default behavior. Programmers may supply their own XtOrderProc to sort children in some specified manner.

Inherited Resources

Composite inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them.

Name	Inherited From	Name	Inherited From
XmNaccelerators	Core	XmNheight	Core
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNscreen	Core
XmNborderColor	Core	XmNsensitive	Core
XmNborderPixmap	Core	XmNtranslations	Core
XmNborderWidth	Core	XmNwidth	Core
XmNcolormap	Core	XmNx	Core
XmNdepth	Core	XmNy	Core
XmNdestroyCallback	Core		

See Also

`Core`(2).

Name

Constraint widget class – a widget that provides constraint resources for its children.

Synopsis

Public Headers:

<Xm/Xm.h>
<X11/Constraint.h>

Class Name:

Constraint

Class Hierarchy:

Core →Composite →Constraint

Class Pointer:

constraintWidgetClass

Instantiation:

Constraint is an Intrinsics meta-class and is not normally instantiated.

Functions/Macros:

`XtIsConstraint()`

Description

Constraint widgets are so named because they may manage the geometry of their children based on constraints associated with each child. These constraints can be as simple as the maximum width and height the parent allows the child to occupy, or as complicated as how other children change if a child is moved or resized. Constraint widgets let a parent define resources that are supplied for their children. For example, if a Constraint parent defines the maximum width and height for its children, these resources are retrieved for each child as if they are resources that are defined by the child widget itself.

New Resources

Constraint does not define any new resources.

Inherited Resources

Constraint inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them.

Name	Inherited From	Name	Inherited From
XmNaccelerators	Core	XmNheight	Core
XmNancestorSensitive	Core	XmNinsertPosition	Composite
XmNbackground	Core	XmNinitialResourcesPersistent	Core

Name	Inherited From	Name	Inherited From
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNscreen	Core
XmNborderWidth	Core	XmNsensitive	Core
XmNchildren	Composite	XmNtranslations	Core
XmNcolormap	Core	XmNwidth	Core
XmNdepth	Core	XmNx	Core
XmNdestroyCallback	Core	XmNy	Core

See Also

`Composite`(2), `Core`*(2)*.

Name

Core widget class – the fundamental class for windowed widgets.

Synopsis

Public Header:
<Xm/Xm.h>
<X11/Core.h>

Class Name:
Core

Class Hierarchy:
Object →RectObj →*unnamed* →Core

Class Pointer:
widgetClass or coreWidgetClass

Instantiation:
Core is an Intrinsics meta-class and is not normally instantiated.

Functions/Macros:
`XtIsWidget`()

Description

Core is the fundamental class for windowed widgets. All widgets with windows are subclasses of Core. The Object and RectObj classes support gadgets (windowless widgets). Core is sometimes instantiated for use as a basic drawing area.

New Resources

Core defines the following resources (some of which are actually defined by the Object and RectObj classes)

Name	Class	Type	Default	Access
XmNaccelerators	XmCAccelerators	XtAccelerators	dynamic	CSG
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNborderColor	XmCBorderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNborderWidth	XmCBorderWidth	Dimension	1	CSG
XmNcolormap	XmCColormap	Colormap	dynamic	CSG

Name	Class	Type	Default	Access
XmNdepth	XmCDepth	int	dynamic	CSG
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

XmNaccelerators
: A translation table bound with its actions for a widget. A destination widget can be set up to use this accelerator table.

XmNancestorSensitive
: Tells whether a widget's immediate parent should receive input. Default value is True if the widget is a top-level shell, copied from the XmNancestorSensitive resource of its parent if the widget is a popup shell, or the bitwise AND of the XmNsensitive and XmNancestorSensitive resources of the parent for other widgets.

XmNbackground
: Widget's background color.

XmNbackgroundPixmap
: Pixmap with which to tile the background, beginning at the upper-left corner.

XmNborderColor
: Pixel value that defines the color of the border.

XmNborderPixmap
: Pixmap with which to tile the border, beginning at the upper-left corner of the border.

XmNborderWidth
: Width (in pixels) of the window's border.

XmNcolormap
: Colormap used in converting to pixel values. Previously created pixel values are unaffected. The default value is the screen's default colormap for top-level shells or is copied from the parent for other widgets.

XmNdepth
: Number of bits allowed for each pixel. The Xt Intrinsics set this resource when the widget is created. As with the XmNcolormap resource, the default value comes from the screen's default or is copied from the parent.

XmNdestroyCallback
: List of callbacks invoked when the widget is destroyed.

XmNheight
: Window height (in pixels), excluding the border.

XmNinitialResourcesPersistent
: Tells whether resources should be reference counted. If True (default), it is assumed that the widget won't be destroyed while the application is running, and thus the widget's resources are not reference counted. Set this resource to False if your application might destroy the widget and will need to deallocate the resources.

XmNmappedWhenManaged
: If True (default), the widget becomes visible (is mapped) as soon as it is both realized and managed. If False, the application performs the mapping and unmapping of the widget. If changed to False after the widget is realized and managed, the widget is unmapped.

XmNscreen
: Screen location of the widget. The default value comes either from the screen's default or is copied from the parent.

XmNsensitive
: Tells whether a widget is sensitive to input. The `XtSetSensitive`() routine can be used to change a widget's sensitivity and to guarantee that if a parent has its XmNsensitive resource set to False, then its children will have their ancestor-sensitive flag set correctly.

XmNtranslations
: Points to a translation table; must be compiled with `XtParseTranslationTable`().

XmNwidth
: Window width (in pixels), excluding the border.

XmNx
: The x-coordinate of the widget's upper-left outer corner, relative to the upper-left inner corner of its parent.

XmNy
: The y-coordinate of the widget's upper-left outer corner, relative to the upper-left inner corner of its parent.

See Also

`Object`(2), `RectObj`(2).

Name

Object widget class – fundamental object class.

Synopsis

Public Headers:
<Xm/Xm.h>
<X11/Object.h>

Class Name:
Object

Class Hierarchy:
Object

Class Pointer:
objectClass

Instantiation:
Object is an Intrinsics meta-class and is not normally instantiated.

Functions/Macros:
`XtIsObject`()

Description

Object is the root of the class hierarchy; it does not have a superclass. All widgets and gadgets are subclasses of Object. Object encapsulates the mechanisms for resource management and is never instantiated.

New Resources

Object defines the following resources:

Name	Class	Type	Default	Access
XmNdestroyCallback	XmCCallback	XtCallbackList	NULL	C

XmNdestroyCallback
List of callbacks invoked when the Object is destroyed.

See Also

`Core`(2).

Name

OverrideShell widget class – a popup shell that bypasses window management.

Synopsis

Public Header:
<X11/Shell.h>

Class Name:
OverrideShell

Class Hierarchy:
Core →Composite →Shell →OverrideShell

Class Pointer:
overrideShellWidgetClass

Instantiation:
widget = `XtCreatePopupShell` (name, overrideShellWidgetClass,...)

Functions/Macros:
`XtIsOverrideShell`()

Description

OverrideShell is a direct subclass of Shell that performs no interaction with window managers. It is used for widgets, such as popup menus, that should bypass the window manager.

New Resources

OverrideShell does not define any new resources.

Inherited Resources

OverrideShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. OverrideShell sets the default values of both XmNoverrideRedirect and XmNsaveUnder to True.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinitialResourcesPersistent	Core
XmNallowShellResize	Shell	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNmappedWhenManaged	Core
XmNaudibleWarning	VendorShell	XmNnumChildren	Composite
XmNbackground	Core	XmNoverrideRedirect	Shell
XmNbackgroundPixmap	Core	XmNpopdownCalback	Shell
XmNborderColor	Core	XmNpopupCallback	Shell
XmNborderPixmap	Core	XmNsaveUnder	Shell
XmNborderWidth	Core	XmNscreen	Core

Resource	Inherited From	Resource	Inherited From
XmNchildren	Composite	XmNsensitive	Core
XmNcolormap	Core	XmNtranslations	Core
XmNcreatePopupChildProc	Shell	XmNvisual	Shell
XmNdepth	Core	XmNwidth	Core
XmNdestroyCallback	Core	XmNx	Core
XmNgeometry	Shell	XmNy	Core
XmNheight	Core		

See Also

`Composite`(2), `Core`(2), `Shell`(2).

Name

RectObj widget class – fundamental object class with geometry.

Synopsis

Public Header:
<Xm/Xm.h>
<X11/RectObj.h>

Class Name:
RectObj

Class Hierarchy:
Object →RectObj

Class Pointer:
rectObjClass

Instantiation:
RectObj is an Intrinsics meta-class and is not normally instantiated.

Functions/Macros:
`XtIsRectObj`()

Description

RectObj is a supporting superclass for widgets and gadgets, defined by the X toolkit intrinsics. All of the Motif widgets are ultimately derived from RectObj. It does not have a window, but it does have a height, width, and location, and it encapsulates the mechanisms for geometry management.

New Resources

RectObj defines the following resources:

Name	Class	Type	Default	Access
XmNancestorSensitive	XmCSensitive	Boolean	dynamic	G
XmNborderWidth	XmCBorderWidth	Dimension	1	CSG
XmNheight	XmCHeight	Dimension	dynamic	CSG
XmNsensitive	XmCSensitive	Boolean	True	CSG
XmNwidth	XmCWidth	Dimension	dynamic	CSG
XmNx	XmCPosition	Position	0	CSG
XmNy	XmCPosition	Position	0	CSG

XmNancestorSensitive
Tells whether a gadget's immediate parent should receive input. Default value is the bitwise AND of the XmNsensitive and XmNancestorSensitive resources of the parent.

XmNborderWidth
Width (in pixels) of the window's border.

XmNheight
Window height (in pixels), excluding the border.

XmNsensitive
Tells whether a widget receives input (is sensitive). The XtSetSensitive() routine can be used to change a widget's sensitivity and to guarantee that if a parent has its XmNsensitive resource set to False, then its children will have their ancestor-sensitive flag set correctly.

XmNwidth
Window width (in pixels), excluding the border.

XmNx
The x-coordinate of the widget's upper-left outer corner, relative to the upper-left inner corner of its parent.

XmNy
The y-coordinate of the widget's upper-left outer corner (that is, outside of any border or shadow rectangle), relative to its parents upper-left inner corner (inside all border or shadow rectangles).

Inherited Resources

RectObj inherits the following resource:

Resource	Inherited From
XmNdestroyCallback	Core

See Also

`Object`(2).

Name

SessionShell widget class – the main shell for an application.

Synopsis

Public Headers:
<Xm/Xm.h>
<X11/Shell.h>

Class Name:
SessionShell

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell →TopLevelShell →
ApplicationShell →SessionShell

Class Pointer:
`sessionShellWidgetClass`

Instantiation:
```
widget = XtAppInitialize (...)
```
or
```
widget = XtAppCreateShell (app_name, app_class,
                           sessionShellWidget-
                           Class,...)
```

Functions/Macros:
`XtAppCreateShell()`, `XtVaAppCreateShell()`, `XtIsSession-Shell()`

Availability

The Session shell is only available from X11R6.

Description

A SessionShell is the normal top-level window for an application. It does not have a parent and it is at the root of the widget tree. An application should have only one SessionShell, unless the application is implemented as multiple logical applications. Normally, an application will use TopLevelShell widgets for other top-level windows.

The SessionShell differs from the ApplicationShell in that it interfaces to the X11R6 Session Management facilities, which enable applications to save or restart themselves in a known state in response to commands from the desktop

A SessionShell is returned by the call to `XtVaAppInitialize()`. It can also be created explicitly with a call to `XtVaAppCreateShell()`.

Interaction with the user during session management is implemented through a token-passing mechanism. A *Checkpoint* token is passed between the Session

Manager and the Session Shell callbacks; the application may interact with the user directly (usually to ask the user if unsaved changes to the application should be saved) only if the SessionShell of the application holds the token. The application may only interact with the user after issuing a request to do so. Issuing a request takes the form of registering a procedure on the XtNinteractCallback list. If and when the Session Manager decides that it is time to allow user interaction, the interact procedures are invoked, with the *checkpoint* token passed to the application through *call_data* for the procedures so registered. After the interaction with the user is complete, the application returns the *checkpoint* by calling `XtSessionReturnToken`(). Unusually, callbacks on the XtNinteractCallback list are invoked one at a time; after the first procedure is called, it is removed from the list.

New Resources

SessionShell defines the following resources:

Name	Class	Type	Default	Access
XtNcancelCallback	XtCCallback	XtCallbackList	NULL	C
XtNcloneCommand	XtCCloneCommand	String *	dynamic	CSG
XtNconnection	XtCConnection	SmcConn	NULL	CSG
XtNcurrentDirectory	XtCCurrentDirectory	String	NULL	CSG
XtNdieCallback	XtCCallback	XtCallbackList	NULL	C
XtNdiscardCommand	XtCDiscardCommand	String *	NULL	CSG
XtNenvironment	XtCEnvironment	String *	NULL	CSG
XtNerrorCallback	XtCCallback	XtCallbackList	NULL	C
XtNinteractCallback	XtCCallback	XtCallbackList	NULL	C
XtNjoinSession	XtCJoinSession	Boolean	True	CSG
XtNprogramPath	XtCProgramPath	String	dynamic	CSG
XtNresignCommand	XtCResignCommand	String *	NULL	CSG
XtNrestartCommand	XtCRestartCommand	String *	dynamic	CSG
XtNrestartStyle	XtCRestartStyle	unsigned char	SmRestartIfRunning	CSG
XtNsaveCallback	XtCCallback	XtCallbackList	NULL	C
XtNsaveCompleteCallback	XtCCallback	XtCallbackList	NULL	C
XtNsessionID	XtCSessionID	String	NULL	CSG
XtNshutdownCommand	XtCShutdownCommand	String *	NULL	CSG

`XtNcancelCallback`
List of callbacks to be invoked when the SessionShell receives a ShutdownCancelled message from the Session Manager.

XtNcloneCommand
: Specifies a command which the Session Manager uses to create a new instance of the application. If the value is NULL, the Session Manager will use the value of the XtNrestartCommand resource.

XtNconnection
: Specifies the connection between the SessionShell and the Session Manager. Normally the SessionShell instantiates this value when the shell is created, although the programmer can specify a value if the application has already established a private connection.

XtNcurrentDirectory
: Specifies a location in the file system where the Session Manager should arrange to restart the application when required to do so.

XtNdieCallback
: List of callbacks invoked when the application receives a Die message from the Session Manager. The application should take whatever steps are required to cleanly terminate.

XtNdiscardCommand
: Specifies a command which, if invoked, will cause the host to discard all information pertaining to the current application state. If NULL, the Session Manager assumes that the application state is fully recoverable from the XtNrestartCommand specification.

XtNenvironment
: Specifies the environment variables (and values) which the Session Manager should set up prior to restarting the application. The resource is assumed to consist of a list of “name=value” strings.

XtNerrorCallback
: List of callbacks to be invoked if the connection between the Session Manager and the SessionShell becomes irrevocably lost. The XtNconnection is reset to NULL by the SessionShell in these circumstances.

XtNinteractCallback
: List of callbacks invoked when a client wants to interact with the user before a session shutdown. This callback list is implemented in a special manner: each time the Session Manager is issued a request to interact with the user, the Session Manager calls *and then removes* the top callback from the list. Furthermore, the request to interact with the user during Session Management operations is performed *simply by registering a callback on this list*. If there is more than one callback on the list, subsequent
callbacks below the top are not called until the application calls `XtSessionReturnToken`(), returning the checkpoint token to the Session Manager.

XtNjoinSession
Specifies whether the SessionShell should automatically initialize a connection to the Session Manager. Setting the resource True at any time will initialize the connection; subsequently setting it False will resign from the session.

XtNprogramPath
The full path of the program which is running. If NULL, the session management uses the first element of the XtNrestartCommand array as the program path.

XtNresignCommand
Specifies a command which logically undoes the client: saved state should be removed.

XtNrestartCommand
Specifies a command which should cause an instance of the application to be invoked, such that it restarts in its current state. If NULL, the XmNargv resource is used as fallback.

XtNrestartStyle
Specifies a hint to the session manager, indicating how the application would like to be restarted. Possible values are:

SmRestartIfRunning
SmRestartAnyway
SmRestartImmediately
SmRestartNever

SmRestartIfRunning is the default, and specifies that the client should restart if it was running at the end of the current session.

SmRestartAnyway specifies that the client should be restarted even if it terminated before the end of the current session.

SmRestartImmediately specifies that the client is meant to run continuously. If it exits at any time, the session manager should restart it in the current session.

SmRestartNever specifies that the client should not be restarted under session management control.

XtNsaveCallback
Specifies a list of callbacks to be invoked when the client receives a SaveYourself message from the session manager. The procedures are responsible for saving the application state.

XtNsaveCompleteCallback
Specifies a list of callbacks to be invoked when the session manager sends a SaveComplete message to the client. Clients can continue their normal operations thereafter.

XtNsessionID
: This resource identifies the client to the session manager. This is either assigned by the session manager when connection is established, or deduced from any -xtsessionID command line argument to the application.

XtNshutdownCommand
: Specifies a command which the session manager will invoke at shutdown; the command should clean up after the client, but not remove any saved state.

Callback Structure

Callbacks on the XtNsaveCallback and XtNinteractCallback lists are each passed the following structure as call_data when invoked:

```
typedef struct _XtCheckpointTokenRec {
    int             save_type;
    int             interact_style;
    Boolean         shutdown;
    Boolean         fast;
    Boolean         cancel_shutdown;
    int             phase;
    int             interact_dialog_type;
    Boolean         request_cancel;
    Boolean         request_next_phase;
    Boolean         save_success;
    int             type;
    Widget          widget;
} XtCheckpointTokenRec, *XtCheckpointToken;
```

The *save_type* element indicates the type of information which the application should attempt to save. Possible values are: SmSaveLocal, SmSaveGlobal, SmSaveBoth.

The *interact_style* element indicates the kind of user interaction which is currently permitted. Possible values are: SmInteractStyleNone, SmInteractStyleErrors, SmInteractStyleAny.

The *shutdown* element indicates whether the save interaction is being performed prior to a session shutdown.

If *fast* is True, the client should endeavour to save the minimum recovery state possible.

If *cancel_shutdown* is True, the Session Manager has sent a ShutdownCancelled message to the client.

The *phase* element is for specialized manager clients use only (the window manager), and indicates the state of the interaction between the Session Manager and the client. The value will be either 1 or 2.

The remaining fields are where the client communicates back to the Session Manager.

The *interact_dialog_type* element specifies the kind of interaction required by the client. The initial value is SmDialogNormal, which is for a normal interactive dialog. The value of SmDialogError requests an error dialog interaction.

The *request_cancel* element is only used by XtNinteractCallbacks, and is a hint to the session manager that the client requests that the current shutdown operation should be cancelled.

The *request_next_phase* element is used by the specialized manager clients: the default value is False, but can be set True by these clients.

The *save_success* element is where the client indicates to the Session Manager the status of the application save-state operations. The value False indicates that the client could not save its state successfully.

The *type* and *widget* fields are internal to the implementation, and are not to be used by application programmers.

Inherited Resources

SessionShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of `XmNborderWidth` is reset to 0 by VendorShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNlabelRenderTable	VendorShell
XmNallowShellResize	Shell	XmNlayoutDirection	VendorShell
XmNancestorSensitive	Core	XmNmappedWhenManaged	Core
XmNargc	ApplicationShell	XmNmaxAspectX	WMShell
XmNargv	ApplicationShell	XmNmaxAspectY	WMShell
XmNaudibleWarning	VendorShell	XmNmaxHeight	WMShell
XmNbackground	Core	XmNmaxWIdth	WMShell
XmNbackgroundPixmap	Core	XmNminAspectX	WMShell
XmNbaseHeight	WMShell	XmNminAspectY	WMShell
XmNbaseWidth	WMShell	XmNminHeight	WMShell
XmNborderColor	Core	XmNminWidth	WMShell
XmNborderPixmap	Core	XmNmwmDecorations	VendorShell

Resource	Inherited From	Resource	Inherited From
XmNborderWidth	Core	XmNmwmFunctions	VendorShell
XmNbuttonFontList	VendorShell	XmNmwmInputMode	VendorShell
XmNbuttonRenderTable	VendorShell	XmNmwmMenu	VendorShell
XmNchildren	Composite	XmNnumChildren	Composite
XmNcolormap	Core	XmNoverrideRedirect	Shell
XmNcreatePopupChildProc	Shell	XmNpopdownCalback	Shell
XmNdefaultFontList	VendorShell	XmNpopupCallback	Shell
XmNdeleteResponse	VendorShell	XmNpreeditType	VendorShell
XmNdepth	Core	XmNsaveUnder	Shell
XmNdestroyCallback	Core	XmNscreen	Core
XmNgeometry	Shell	XmNsensitive	Core
XmNheight	Core	XmNshellUnitType	VendorShell
XmNheightInc	WMShell	XmNtextFontList	VendorShell
XmNiconic	TopLevelShell	XmNtextRenderTable	VendorShell
XmNiconMask	WMShell	XmNtitle	WMShell
XmNiconName	TopLevelShell	XmNtitleEncoding	WMShell
XmNiconNameEncoding	TopLevelShell	XmNtransient	WMShell
XmNiconPixmap	WMShell	XmNtranslations	Core
XmNiconWindow	WMShell	XmNvisual	Shell
XmNinitialResourcesPersistent	Core	XmNwaitForWm	WMShell
XmNinitialState	WMShell	XmNwidth	Core
XmNinput	WMShell	XmNwidthInc	WMShell
XmNinputMethod	VendorShell	XmNwindowGroup	WMShell
XmNinputPolicy	VendorShell	XmNwinGravity	WMShell
XmNinsertPosition	Composite	XmNwmTimeout	WMShell
XmNkeyboardFocusPolicy	VendorShell	XmNx	Core
XmNlabelFontList	VendorShell	XmNy	Core

The VendorShell superclass installs a handler which intercepts the window manager WM_DELETE_WINDOW message. The handler is inherited by sub-classes of VendorShell, and has the behavior that if XmNdeleteResponse is XmDESTROY, and the widget is an instance of an ApplicationShell, then the application context associated with the widget is destroyed, followed by a call to exit().

See Also

ApplicationShell(2), Composite(2), Core(2), Shell(2), TopLevelShell(2), VendorShell(2), WMShell(2).

Name

Shell widget class – fundamental widget class that controls interaction between top-level windows and the window manager.

Synopsis

Public Header:

<Xm/Xm.h>
<X11/Shell.h>

Class Name:

Shell

Class Hierarchy:

Core →Composite →Shell

Class Pointer:

shellWidgetClass

Instantiation:

Shell is an Intrinsics meta-class and is not normally instantiated.

Functions/Macros:

`XtIsShell`()

Description

Shell is a subclass of Composite that handles interaction between the window manager and its single child.

New Resources

Shell defines the following resources:

Name	Class	Type	Default	Access
XmNallowShellResize	XmCAllowShellResize	Boolean	False	CSG
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

XmNallowShellResize

If False (default), the Shell widget refuses geometry requests from its children (by returning XtGeometryNo).

XmNcreatePopupChildProc
A pointer to an XtCreatePopupChildProc procedure that creates a child widget--but only when the shell is popped up, not when the application is started. This is useful in menus, for example, since you don't need to create the menu until it is popped up. This procedure is called after any callbacks specified in the XmNpopupCallback resource.

XmNgeometry
This resource specifies the values for the resources XmNx, XmNy, XmNwidth, and
XmNheight in situations where an unrealized widget has added or removed some of its managed children.

XmNoverrideRedirect
If True, the widget is considered a temporary window that redirects the keyboard focus away from the main application windows. Usually this resource shouldn't be changed.

XmNpopdownCallback
List of callbacks that are called when the widget is popped down using `XtPopdown`().

XmNpopupCallback
List of callbacks that are called when the widget is popped up using `XtPopup`().

XmNsaveUnder
If True, screen contents that are obscured by a widget are saved, thereby avoiding the overhead of sending expose events after the widget is unmapped.

XmNvisual
The visual server resource that is used when creating the widget.

Procedures

XtCreatePopupChildProc
An XtCreatePopupChildProc is a pointer to a procedure, specified as follows:

```
typedef void (*XtCreatePopupChildProc) (Widget);
```

An XtCreatePopupChildProc procedure is called when a Shell or derived class is popped up, typically through a call to `XtPopup`(). The function has a single parameter, which is the widget ID of the shell.

Inherited Resources

Shell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them.

Name	Inherited From	Name	Inherited From
XmNaccelerators	Core	XmNheight	Core
XmNancestorSensitive	Core	XmNinsertPosition	Composite
XmNbackground	Core	XmNinitialResourcesPersistent	Core
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNscreen	Core
XmNborderWidth	Core	XmNsensitive	Core
XmNchildren	Composite	XmNtranslations	Core
XmNcolormap	Core	XmNwidth	Core
XmNdepth	Core	XmNx	Core
XmNdestroyCallback	Core	XmNy	Core

See Also

`Composite`(2), `Core`*(2)*.

Name

TopLevelShell widget class – additional top-level shells for an application.

Synopsis

Public Header:
<Xm/Xm.h>
<X11/Shell.h>

Class Name:
TopLevelShell

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell →TopLevelShell

Class Pointer:
topLevelShellWidgetClass

Instantiation:
widget = `XtCreatePopupShell` (name, topLevelShellWidgetClass,...)

Functions/Macros:
`XtIsTopLevelShell`()

Description

TopLevelShell is a subclass of VendorShell that is used for additional shells in applications having more than one top-level window.

New Resources

TopLevelShell defines the following resources:

Name	Class	Type	Default	Access
XmNiconic	XmCIconic	Boolean	False	CSG
XmNiconName	XmCIconName	String	NULL	CSG
XmNiconNameEncoding	XmCIconNameEncoding	Atom	dynamic	CSG

XmNiconic
If True, the widget is realized as an icon, otherwise as a normal window. XmNiconic overrides the value of the inherited XmNinitialState resource.

XmNiconName
The abbreviated name that labels an iconified application.

XmNiconNameEncoding
The property type for encoding the XmNiconName resource.

Inherited Resources

TopLevelShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. TopLevelShell resets XmNinput to True.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNmaxAspectX	WMShell
XmNallowShellResize	Shell	XmNmaxAspectY	WMShell
XmNancestorSensitive	Core	XmNmaxHeight	WMShell
XmNaudibleWarning	VendorShell	XmNmaxWIdth	WMShell
XmNbackground	Core	XmNminAspectX	WMShell
XmNbackgroundPixmap	Core	XmNminAspectY	WMShell
XmNbaseHeight	WMShell	XmNminHeight	WMShell
XmNbaseWidth	WMShell	XmNminWidth	WMShell
XmNborderColor	Core	XmNmwmDecorations	VendorShell
XmNborderPixmap	Core	XmNmwmFunctions	VendorShell
XmNborderWidth	Core	XmNmwmInputMode	VendorShell
XmNbuttonFontList	VendorShell	XmNmwmMenu	VendorShell
XmNbuttonRenderTable	VendorShell	XmNnumChildren	Composite
XmNchildren	Composite	XmNoverrideRedirect	Shell
XmNcolormap	Core	XmNpopdownCalback	Shell
XmNcreatePopupChildProc	Shell	XmNpopupCallback	Shell
XmNdefaultFontList	VendorShell	XmNpreeditType	VendorShell
XmNdeleteResponse	VendorShell	XmNsaveUnder	Shell
XmNdepth	Core	XmNscreen	Core
XmNdestroyCallback	Core	XmNsensitive	Core
XmNgeometry	Shell	XmNshellUnitType	VendorShell
XmNheight	Core	XmNtextFontList	VendorShell
XmNheightInc	WMShell	XmNtextRenderTable	VendorShell
XmNiconMask	WMShell	XmNtitle	WMShell
XmNiconPixmap	WMShell	XmNtitleEncoding	WMShell
XmNiconWindow	WMShell	XmNtransient	WMShell
XmNinitialResourcesPersistent	Core	XmNtranslations	Core
XmNinitialState	WMShell	XmNvisual	Shell
XmNinput	WMShell	XmNwaitForWm	WMShell
XmNinputMethod	VendorShell	XmNwidth	Core

Resource	Inherited From	Resource	Inherited From
XmNinputPolicy	VendorShell	XmNwidthInc	WMShell
XmNinsertPosition	Composite	XmNwindowGroup	WMShell
XmNkeyboardFocusPolicy	VendorShell	XmNwinGravity	WMShell
XmNlabelFontList	VendorShell	XmNwmTimeout	WMShell
XmNlabelRenderTable	VendorShell	XmNx	Core
XmNlayoutDirection	VendorShell	XmNy	Core
XmNmappedWhenManaged	Core		

See Also

`Composite`(2), `Core`(2). `Shell`(2), `VendorShell`(2), `WMShell`(2).

Name

TransientShell widget class – popup shell that interacts with the window manager.

Synopsis

Public Header:
<Xm/Xm.h>
<X11/Shell.h>

Class Name:
TransientShell

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell →TransientShell

Class Pointer:
transientShellWidgetClass

Instantiation:

Functions/Macros:
`XtIsTransientShell`()

Description

TransientShell is a subclass of VendorShell that is used for popup shell widgets, such as dialog boxes, that interact with the window manager. Most window managers will not allow the user to iconify a TransientShell window on its own and may iconify it automatically if the window that it is transient for is iconified.

New Resources

TransientShell defines the following resources:

Name	Class	Type	Default	Access
XmNtransientFor	XmCTransientFor	Widget	NULL	CSG

XmNtransientFor
The widget from which the TransientShell will pop up. If the value of this resource is NULL or identifies an unrealized widget, then TransientShell uses the value of the WMShell resource XmNwindowGroup.

Inherited Resources

TransientShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. TransientShell resets the resources XmNinput, XmNtransient, and XmNsaveUnder to True.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNmaxAspectX	WMShell
XmNallowShellResize	Shell	XmNmaxAspectY	WMShell
XmNancestorSensitive	Core	XmNmaxHeight	WMShell
XmNaudibleWarning	VendorShell	XmNmaxWIdth	WMShell
XmNbackground	Core	XmNminAspectX	WMShell
XmNbackgroundPixmap	Core	XmNminAspectY	WMShell
XmNbaseHeight	WMShell	XmNminHeight	WMShell
XmNbaseWidth	WMShell	XmNminWidth	WMShell
XmNborderColor	Core	XmNmwmDecorations	VendorShell
XmNborderPixmap	Core	XmNmwmFunctions	VendorShell
XmNborderWidth	Core	XmNmwmInputMode	VendorShell
XmNbuttonFontList	VendorShell	XmNmwmMenu	VendorShell
XmNbuttonRenderTable	VendorShell	XmNnumChildren	Composite
XmNchildren	Composite	XmNoverrideRedirect	Shell
XmNcolormap	Core	XmNpopdownCalback	Shell
XmNcreatePopupChildProc	Shell	XmNpopupCallback	Shell
XmNdefaultFontList	VendorShell	XmNpreeditType	VendorShell
XmNdeleteResponse	VendorShell	XmNsaveUnder	Shell
XmNdepth	Core	XmNscreen	Core
XmNdestroyCallback	Core	XmNsensitive	Core
XmNgeometry	Shell	XmNshellUnitType	VendorShell
XmNheight	Core	XmNtextFontList	VendorShell
XmNheightInc	WMShell	XmNtextRenderTable	VendorShell
XmNiconMask	WMShell	XmNtitle	WMShell
XmNiconPixmap	WMShell	XmNtitleEncoding	WMShell
XmNiconWindow	WMShell	XmNtransient	WMShell
XmNinitialResourcesPersistent	Core	XmNtranslations	Core
XmNinitialState	WMShell	XmNvisual	Shell
XmNinput	WMShell	XmNwaitForWm	WMShell
XmNinputMethod	VendorShell	XmNwidth	Core

Resource	Inherited From	Resource	Inherited From
XmNinputPolicy	VendorShell	XmNwidthInc	WMShell
XmNinsertPosition	Composite	XmNwindowGroup	WMShell
XmNkeyboardFocusPolicy	VendorShell	XmNwinGravity	WMShell
XmNlabelFontList	VendorShell	XmNwmTimeout	WMShell
XmNlabelRenderTable	VendorShell	XmNx	Core
XmNlayoutDirection	VendorShell	XmNy	Core
XmNmappedWhenManaged	Core		

See Also

`Composite`(2), `Core`(2), `Shell`(2), `VendorShell`(2), `WMShell`(2).

Name

VendorShell widget class – shell widget with Motif-specific hooks for window manager interaction.

Synopsis

Public Header:
<Xm/VendorS.h>
<X11/Shell.h>

Class Name:
VendorShell

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell

Class Pointer:
vendorShellWidgetClass

Instantiation:
VendorShell is a meta-class and is not normally instantiated.

Functions/Macros:
`XmIsVendorShell`()

Description

VendorShell is a vendor-specific supporting superclass for all shell classes that are visible to the window manager and that do not have override redirection. VendorShell defines resources that provide the Motif look-and-feel and manages the specific communication needed by the Motif Window Manager (*mwm*).

Traits

VendorShell holds the `XmQTspecifyRenderTable`, `XmQTspecifyLayoutDirection`, `XmQTaccessColors` and `XmQTspecifyUnitType` traits, which are inherited by any derived classes, and uses the `XmQTspecifyRenderTable` trait.

New Resources

VendorShell defines the following resources:

Name	Class	Type	Default	Access
XmNaudibleWarning	XmCAudibleWarning	unsigned char	XmBELL	CSG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNdeleteResponse	XmCDeleteResponse	unsigned char	XmDESTROY	CSG
XmNinputMethod	XmCInputMethod	String	NULL	CSG

Name	Class	Type	Default	Access
XmNinputPolicy	XmCInputPolicy	XmInputPolicy	XmPER_SHELL	CSG
XmNkeyboardFocusPolicy	XmCKeyboardFocusPolicy	unsigned char	XmEXPLICIT	CSG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	XmLEFT_TO_RIGHT	CG
XmNmwmDecorations	XmCMwMDecorations	int	-1	CSG
XmNmwmFunctions	XmCMwMFunctions	int	-1	CSG
XmNmwmInputMode	XmCMwMInputMode	int	-1	CSG
XmNmwmMenu	XmCMwmMenu	String	NULL	
XmNpreeditType	XmCPreeditType	String	dynamic	CSG
XmNshellUnitType	XmCShellUnitType	unsigned char	XmPIXELS	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNuseAsyncGeometry	XmCUseAsyncGeometry	Boolean	False	CSG
XmNunitType	XmCUnitType	unsigned char	XmPIXELS	CSG

XmNaudibleWarning
Specifies whether an action performs an associated audible cue. Possible values:

```
XmBELL      /* rings the bell  */
XmNONE      /* does nothing  */
```

XmNbuttonFontList
Specifies the font list used for the button descendants of the VendorShell widget. In Motif 2.0 and later, the XmFontList is considered obsolete, and is replaced by the XmRenderTable. The XmNbuttonRenderTable resource is the preferred method of specifying appearance.

XmNbuttonRenderTable
In Motif 2.0 and later, specifies the render table to be used by VendorShell's button descendants. If initially NULL, the value is taken from any specified XmNdefaultFontList value for backwards compatability. If this is also NULL, the nearest ancestor which has the XmQTspecifyRenderTable trait is sought, taking the XmBUTTON_RENDER_TABLE value from any widget so found.

XmNdefaultFontList
The default font list for the children of the VendorShell widget. The resource is obsolete, replaced by the XmNbuttonFontList, XmNlabelFontList, and XmNtextFontList resources, which are in their turn also obsolete.

XmNdeleteResponse
The action to perform when the shell receives a WM_DELETE_WINDOW message. Possible values:

```
XmDESTROY       /* destroy window      */
XmUNMAP         /* unmap window        */
XmDO_NOTHING    /* leave window as is  */
```

XmNinputMethod
Specifies the string that sets the locale modifier for the input method.

XmNinputPolicy
In Motif 2.0 and later, specifies the policy to adopt when creating an Input Context. Possible values:

```
XmPER_SHELL     /* one input context per shell hierarchy */
XmPER_WIDGET    /* one input context per widget          */
```

XmNkeyboardFocusPolicy
The method of assigning keyboard focus. Possible values:

```
XmEXPLICIT      /* click-to-type policy    */
XmPOINTER       /* pointer-driven policy   */
```

XmNlabelFontList
Specifies the font list used for the label descendants of the VendorShell widget. In Motif 2.0 and later, the XmFontList is considered obsolete, and is replaced by the XmRenderTable. The XmNlabelRenderTable resource is the preferred method of specifying appearance.

XmNlabelRenderTable
In Motif 2.0 and later, specifies the render table to be used by VendorShell's label descendants. If initially NULL, the value is taken from any specified XmNdefaultFontList value for backwards compatability. If this is also NULL, the nearest ancestor which has the XmQTspecifyRenderTable trait is sought, taking the XmLABEL_RENDER_TABLE value from any widget so found.

XmNlayoutDirection
In Motif 2.0 and later, specifies the default direction in which visual components are to be laid out. Descendants of VendorShell use this value in the absence of an explicit layout direction further down the widget hierarchy. Possible values:

```
XmLEFT_TO_RIGHT
XmRIGHT_TO_LEFT
XmBOTTOM_TO_TOP
XmTOP_TO_BOTTOM
XmBOTTOM_TO_TOP_LEFT_TO_RIGHT
XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
```

XmTOP_TO_BOTTOM_LEFT_TO_RIGHT
XmTOP_TO_BOTTOM_RIGHT_TO_LEFT
XmLEFT_TO_RIGHT_BOTTOM_TO_TOP
XmRIGHT_TO_LEFT_BOTTOM_TO_TOP
XmLEFT_TO_RIGHT_TOP_TO_BOTTOM
XmRIGHT_TO_LEFT_TOP_TO_BOTTOM

XmNmwmDecorations

This resource corresponds to the values assigned by the decorations field of the _MOTIF_WM_HINTS property. This resource determines which frame buttons and handles to include with a window. The value for the resource is a bitwise inclusive OR of one or more of the following, which are defined in *<Xm/MwmUtil.h>*:

```
MWM_DECOR_ALL          /* remove decorations from full set */
MWM_DECOR_BORDER       /* window border                    */
MWM_DECOR_RESIZEH      /* resize handles                   */
MWM_DECOR_TITLE        /* title bar                        */
MWM_DECOR_MENU         /* window's menu button             */
MWM_DECOR_MINIMIZE     /* minimize button                  */
MWM_DECOR_MAXIMIZE     /* maximize button                  */
```

XmNmwmFunctions

This resource corresponds to the values assigned by the functions field of the _MOTIF_WM_HINTS property. This resource determines which functions to include in the system menu. The value for the resource is a bitwise inclusive OR of one or more of the following, which are defined in the header file *<Xm/MwmUtil.h>*.

```
MWM_FUNC_ALL           /* remove functions from full set */
MWM_FUNC_RESIZE        /* f.resize                       */
MWM_FUNC_MOVE          /* f.move                         */
MWM_FUNC_MINIMIZE      /* f.minimize                     */
MWM_FUNC_MAXIMIZE      /* f.maximize                     */
MWM_FUNC_CLOSE         /* f.kill                         */
```

XmNmwmInputMode

This resource corresponds to the values assigned by the input_mode field of the _MOTIF_WM_HINTS property. This resource determines the constraints on the window's keyboard focus. That is, it determines whether the application takes the keyboard focus away from the primary window or not. The possible values are as follows, defined in *<Xm/MwmUtil.h>*:

MWM_INPUT_MODELESS[1]
MWM_INPUT_PRIMARY_APPLICATION_MODAL

MWM_INPUT_SYSTEM_MODAL
MWM_INPUT_FULL_APPLICATION_MODAL

If the value is MWM_INPUT_MODELESS, input can be directed to any window. If the value is MWM_INPUT_PRIMARY_APPLICATION_MODAL, input can not be directed at an ancestor of the window. The value MWM_INPUT_SYSTEM_MODAL indicates that input only goes to the current window. MWM_INPUT_FULL_APPLICATION_MODAL specifies that input may not be directed at any other window of the application.

XmNmwmMenu
The menu items to add at the bottom of the client's window menu. The string has this format:

label [mnemonic] [accelerator] mwm_f.function

XmNpreeditType
Specifies the input method style(s) that are available. The resource value is a comma separated list of the following values:

```
OffTheSpot      /* XIMPreeditArea      */
Root            /* XIMPreeditNothing   */
None            /* XIMPreeditNone      */
OverTheSpot     /* XIMPreeditPosition  */
OnTheSpot       /* XIMPreeditCallbacks */
```

XmNshellUnitType
The measurement units to use in resources that specify a size or position. In Motif 2.0 and later, the resource is obsolete, being replaced by the XmNunitType resource.

XmNtextFontList
Specifies the font list used for the text descendants of the VendorShell widget. In Motif 2.0 and later, the XmFontList is considered obsolete, and is replaced by the XmRenderTable. The XmNtextRenderTable resource is the preferred method of specifying appearance.

XmNtextRenderTable
In Motif 2.0 and later, specifies the render table to be used by VendorShell's text and list descendants. If initially NULL, the value is taken from any specified XmNdefaultFontList value for backwards compatability. If this is also NULL, the nearest ancestor which has the XmQTspecifyRenderTable trait is sought, taking the XmTEXT_RENDER_TABLE value from any widget so found.

1.Erroneously given as MWM_INPUT_MODELES in 2nd edition.

XmNuseAsyncGeometry

If True, the geometry manager doesn't wait to confirm a geometry request that was sent to the window manager. The geometry manager performs this by setting the WMShell resource XmNwaitForWm to False and by setting the WMShell resource XmNwmTimeout to 0.

If XmNuseAsyncGeometry is False (default), the geometry manager uses synchronous notification, and so it doesn't change the resources XmNwaitForWm and XmNwmTimeout.

XmNunitType

In Motif 2.0 and later, specifies the units in which size and position resources are calculated. The resource replaces XmNshellUnitType, which is considered obsolete. The values XmFONT_UNITS and Xm100TH_FONT_UNITS have a horizontal and vertical component, calculated from the values of the XmNhorizontalFontUnit and XmNverticalFontUnit resources of the XmScreen object. Possible values:

```
XmPIXELS            /* pixels                          */
XmMILLIMETERS       /* millimeters                     */
Xm100TH_MILLIMETERS /* 1/100 of a millimeter           */
XmCENTIMETERS       /* centimeters                     */
XmINCHES            /* inches                          */
Xm1000TH_INCHES     /* 1/1000 of an inch               */
XmPOINTS            /* point units (1/72 of an inch)   */
Xm100TH_POINTS      /* 1/100 of a point                */
XmFONT_UNITS        /* depends on XmScreen resources   */
Xm100TH_FONT_UNITS  /* 1/100 of the above              */
```

Inherited Resources

VendorShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. VendorShell resets XmNborderWidth from 1 to 0 and resets XmNinput to True.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNmaxAspectX	WMShell
XmNallowShellResize	Shell	XmNmaxAspectY	WMShell
XmNancestorSensitive	Core	XmNmaxHeight	WMShell
XmNbackground	Core	XmNmaxWIdth	WMShell
XmNbackgroundPixmap	Core	XmNminAspectX	WMShell
XmNbaseHeight	WMShell	XmNminAspectY	WMShell
XmNbaseWidth	WMShell	XmNminHeight	WMShell

Resource	Inherited From	Resource	Inherited From
XmNborderColor	Core	XmNminWidth	WMShell
XmNborderPixmap	Core	XmNnumChildren	Composite
XmNborderWidth	Core	XmNoverrideRedirect	Shell
XmNchildren	Composite	XmNpopdownCalback	Shell
XmNcolormap	Core	XmNpopupCallback	Shell
XmNcreatePopupChildProc	Shell	XmNsaveUnder	Shell
XmNdepth	Core	XmNscreen	Core
XmNdestroyCallback	Core	XmNsensitive	Core
XmNgeometry	Shell	XmNtitle	WMShell
XmNheight	Core	XmNtitleEncoding	WMShell
XmNheightInc	WMShell	XmNtransient	WMShell
XmNiconic	TopLevelShell	XmNtranslations	Core
XmNiconMask	WMShell	XmNvisual	Shell
XmNiconName	TopLevelShell	XmNwaitForWm	WMShell
XmNiconNameEncoding	TopLevelShell	XmNwidth	Core
XmNiconPixmap	WMShell	XmNwidthInc	WMShell
XmNiconWindow	WMShell	XmNwindowGroup	WMShell
XmNinitialResourcesPersistent	Core	XmNwinGravity	WMShell
XmNinitialState	WMShell	XmNwmTimeout	WMShell
XmNinput	WMShell	XmNx	Core
XmNinsertPosition	Composite	XmNy	Core
XmNmappedWhenManaged	Core		

See Also

`Composite`(2), `Core`(2), `Shell`(2), `WMShell`(2).

Name

WMShell widget class – fundamental shell widget that interacts with an ICCCM-compliant window manager.

Synopsis

Public Header:

<Xm/Xm.h>
<X11/Shell.h>

Class Name:

WMShell

Class Hierarchy:

Core →Composite →Shell →WMShell

Class Pointer:

wmShellWidgetClass

Instantiation:

WMShell is an Intrinsics meta-class and is not normally instantiated.

Functions/Macros:

`XtIsWMShell`()

Description

WMShell is a direct subclass of Shell that provides basic window manager interaction. WMShell is not directly instantiated; it encapsulates the application resources that applications use to communicate with window managers.

New Resources

WMShell defines the following resources:

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
XmNinitialState	XmCInitialState	int	NormalState	CSG
XmNiconX	XmCIconY	int	-1	CSG
XmNiconY	XmCIconY	int	-1	CSG
XmNinput	XmCInput	Boolean	False	CSG
XmNmaxAspectX	XmCMaxAspectX	int	XtUnspecifiedShellInt	CSG

Name	Class	Type	Default	Access
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	CSG

XmNbaseHeight

XmNbaseWidth
The base dimensions from which the preferred height and width can be stepped up or down (as specified by XmNheightInc or XmNwidthInc).

XmNheightInc
The amount by which to increment or decrement the window's height when the window manager chooses a preferred value. The base height is XmNbaseHeight, and the height can decrement to the value of XmNminHeight or increment to the value of XmN-maxHeight. See also XmNwidthInc.

XmNiconMask
A bitmap that the window manager can use in order to clip the application's icon into a non-rectangular shape.

XmNiconPixmap
The application's icon.

XmNiconWindow
The ID of a window that serves as the application's icon.

XmNiconX

XmNiconY
Window manager hints for the root window coordinates of the application's icon.

XmNinitialState
The initial appearance of the widget instance. Possible values are defined in *<X11/Xutil.h>*:

```
NormalState     /* application starts as a window */
IconicState     /* application starts as an icon */
```

XmNinput
A Boolean that, in conjunction with the WM_TAKE_FOCUS atom in the WM_PROTOCOLS property, determines the application's keyboard focus model. The result is determined by the value of XmNinput and the existence of the atom, as described below:

Value of XmNinput Resource	WM_TAKE_FOCUS Atom	Keyboard Focus Model
False	Does not exist	No input allowed
True	Does not exist	Passive
False	Exists	Globally active
True	Exists	Locally active

XmNmaxAspectX

XmNmaxAspectY
The numerator and denominator, respectively, of the maximum aspect ratio requested for this widget.

XmNmaxHeight

XmNmaxWidth
The maximum dimensions for the widget's preferred height or width.

XmNminAspectX

XmNminAspectY
The numerator and denominator, respectively, of the minimum aspect ratio requested for this widget.

XmNminHeight

XmNminWidth
The minimum dimensions for the widget's preferred height or width.

XmNtitle
The string that the window manager displays as the application's name. By default, the icon name is used, but if this isn't specified, the name of the application is used.

XmNtitleEncoding
The property type for encoding the XmNtitle resource.

XmNtransient
If True, this indicates a popup window or some other transient widget. This resource is usually not changed.

XmNwaitForWm
If True (default), the X Toolkit waits for a response from the window manager before acting as if no window manager exists. The waiting time is specified by the XmNwmTimeout resource.

XmNwidthInc
The amount by which to increment or decrement the window's width when the window manager chooses a preferred value. The base width is XmNbaseWidth, and the width can decrement to the value of XmNminWidth or increment to the value of XmN-maxWidth. See also XmNheightInc.

XmNwindowGroup
The window associated with this widget instance. This window acts as the primary window of a group of windows that have similar behavior.

XmNwinGravity
The window gravity used in positioning the widget. Unless an initial value is given, this resource will be set when the widget is realized. The default value is NorthWestGravity (if the Shell resource XmNgeometry is NULL); otherwise, XmNwinGravity assumes the value returned by the XmWMGeometry routine.

XmNwmTimeout
The number of milliseconds that the X Toolkit waits for a response from the window manager. This resource is meaningful when the XmNwaitForWm resource is set to True.

Inherited Resources

WMShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinitialResourcesPersistent	Core
XmNallowShellResize	Shell	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNmappedWhenManaged	Core
XmNbackground	Core	XmNnumChildren	Composite
XmNbackgroundPixmap	Core	XmNoverrideRedirect	Shell
XmNborderColor	Core	XmNpopdownCalback	Shell
XmNborderPixmap	Core	XmNpopupCallback	Shell

Resource	Inherited From	Resource	Inherited From
XmNborderWidth	Core	XmNsaveUnder	Shell
XmNchildren	Composite	XmNscreen	Core
XmNcolormap	Core	XmNsensitive	Core
XmNcreatePopupChildProc	Shell	XmNtranslations	Core
XmNdepth	Core	XmNvisual	Shell
XmNdestroyCallback	Core	XmNwidth	Core
XmNgeometry	Shell	XmNx	Core
XmNheight	Core	XmNy	Core

See Also

`Composite`(2), `Core`(2), `Shell`(2).

Name

XmArrowButton widget class – a directional arrow-shaped button widget.

Synopsis

Public Header:
<Xm/ArrowB.h>

Class Name:
XmArrowButton

Class Hierarchy:
Core →XmPrimitive →XmArrowButton

Class Pointer:
xmArrowButtonWidgetClass

Instantiation:
widget = `XmCreateArrowButton` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmArrowButtonWidgetClass,...)

Functions/Macros:
`XmCreateArrowButton`(), `XmIsArrowButton`()

Description

An ArrowButton is a directional arrow-shaped button that includes a shaded border. The shading changes to make the ArrowButton appear either pressed in when selected or raised when unselected.

Traits

ArrowButton holds the XmQTactivatable trait, which is inherited by any derived classes.

New Resources

ArrowButton defines the following resources:

Name	Class	Type	Default	Access
XmNarrowDirection	XmCArrowDirection	unsigned char	XmARROW_UP	CSG
XmNdetailShadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	CSG

XmNarrowDirection
Sets the arrow direction. Possible values:

XmARROW_UP	XmARROW_LEFT
XmARROW_DOWN	XmARROW_RIGHT

XmNdetailShadowThickness
In Motif 2.0 and later, specifies the thickness of the shadow inside the triangle of the ArrowButton. Values of 0 (zero), 1, and 2 are supported. In Motif 2.0, the default is 2. In Motif 2.1 and later, the default depends upon the value of the XmDisplay resource XmNenableThinThickness: if True, the default is 1, otherwise 2.

XmNmultiClick
A flag that determines whether successive button clicks are processed or ignored. Possible values:

```
XmMULTICLICK_DISCARD    /* ignore successive button clicks; */
                        /* default value in a menu system   */
XmMULTICLICK_KEEP       /* count successive button clicks; */
                        /* default value when not in a menu */
```

Callback Resources

ArrowButton defines the following callback resources:

Callback	Reason Constant
XmNactivateCallback	XmCR_ACTIVATE
XmNarmCallback	XmCR_ARM
XmNdisarmCallback	XmCR_DISARM

XmNactivateCallback
List of callbacks that are called when BSelect is pressed and released inside the widget.

XmNarmCallback
List of callbacks that are called when BSelect is pressed while the pointer is inside the widget.

XmNdisarmCallback
List of callbacks that are called when BSelect is released after it has been pressed inside the widget.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* the reason that the callback was called */
    XEvent      *event;         /* event structure that triggered callback */
    int         click_count;    /* number of clicks in multi-click sequence */
} XmArrowButtonCallbackStruct;
```

click_count is meaningful only for XmNactivateCallback. Furthermore, if the XmN-multiClick resource is set to XmMULTICLICK_KEEP, then XmN-activate-Callback is called for each click, and the value of click_count is the number of clicks that have occurred in the last sequence of multiple clicks. If the XmNmultiClick resource is set to XmMULTICLICK_DISCARD, then click_count always has a value of 1.

Inherited Resources

ArrowButton inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNhighlightThickness	XmPrimitive
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNlayoutDirection	XmPrimitive
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnavigationType	XmPrimitive
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmPrimitive	XmNsensitive	Core
XmNbottomShadowPixmap	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNcolormap	Core	XmNtopShadowColor	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNdepth	Core	XmNtranslations	Core
XmNdestroyCallback	Core	XmNtraversalOn	XmPrimitive
XmNforeground	XmPrimitive	XmNunitType	XmPrimitive
XmNheight	Core	XmNuserData	XmPrimitive
XmNhelpCallback	XmPrimitive	XmNwidth	Core
XmNhighlightColor	XmPrimitive	XmNx	Core
XmNhighlightOnEnter	XmPrimitive	XmNy	Core
XmNhighlightPixmap	XmPrimitive		

Translations

The translations of ArrowButton include those of Primitive:

Event	Action
BSelect Press	Arm()
BSelect Click	Activate() Disarm()
BSelect Release	Activate() Disarm()
Bselect Press 2+	MultiArm()
BSelect Release 2+	MultiActivate()
KSelect	ArmAndActivate()
MCtrl BSelect Press	ButtonTakeFocus()
KHelp	Help()

Action Routines

ArrowButton defines the following action routines:

Activate()
: Displays the ArrowButton as unselected, and invokes the list of callbacks specified by XmNactivateCallback.

Arm()
: Displays the ArrowButton as selected, and invokes the list of callbacks specified by XmNarmCallback.

ArmAndActivate()
: Displays the ArrowButton as selected, and invokes the list of callbacks specified by XmNarmCallback. After doing this, the action routine displays the ArrowButton as unselected, and invokes the list of callbacks specified by XmNactivateCallback and XmNdisarmCallback.

ButtonTakeFocus()
: In Motif 2.0 and later, moves the current keyboard focus to the ArrowButton, without activating the widget.

Disarm()
: Displays the ArrowButton as unselected, and invokes the list of callbacks specified by XmNdisarmCallback.

Help()
: Invokes the list of callbacks specified by XmNhelpCallback. If the ArrowButton doesn't have any help callbacks, the Help() routine invokes those associated with the nearest ancestor that has them.

MultiActivate()
: Increments the click_count member of XmArrowButtonCallbackStruct, displays the ArrowButton as unselected, and invokes the list of callbacks specified by XmNactivateCallback and XmNdisarmCallback. This action routine takes effect only when the XmNmultiClick resource is set to XmMULTICLICK_KEEP.

MultiArm()
: Displays the ArrowButton as selected, and invokes the list of callbacks specified by XmNarmCallback. This action routine takes effect only when the XmNmultiClick resource is set to XmMULTICLICK_KEEP.

Additional Behavior

ArrowButton has the following additional behavior:

<EnterWindow>
: Displays the ArrowButton as selected if the pointer leaves and re-enters the window while BSelect is pressed.

<LeaveWindow>
: Displays the ArrowButton as unselected if the pointer leaves the window while BSelect is pressed.

See Also

`XmCreateObject`(1), `Core`(2), `XmPrimitive`(2).

Name

XmArrowButtonGadget widget class – a directional arrow-shaped button gadget.

Synopsis

Public Header:
<Xm/ArrowBG.h>

Class Name:
XmArrowButtonGadget

Class Hierarchy:
Object →RectObj →XmGadget →XmArrowButtonGadget

Class Pointer:
xmArrowButtonGadgetClass

Instantiation:
widget = `XmCreateArrowButtonGadget` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmArrowButtonGadgetClass,...)

Functions/Macros:
`XmCreateArrowButtonGadget`(), `XmIsArrowButtonGadget`()

Description

ArrowButtonGadget is the gadget variant of ArrowButton.

ArrowButtonGadget's resources, callback resources, and callback structure are the same as those of ArrowButton.

Traits

ArrowButtonGadget holds the XmQTactivatable, XmQTcareParentVisual, and XmQTaccessColors traits, which are inherited by any derived classes.

Inherited Resources

ArrowButtonGadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of XmNborderWidth is reset to 0 by Gadget.

Resource	Inherited From	Resource	Inherited From
XmNancestorSensitive	RectObj	XmNhighlightThickness	XmGadget
XmNbackground	XmGadget	XmNlayoutDirection	XmGadget
XmNbackgroundPixmap	XmGadget	XmNnavigationType	XmGadget
XmNbottomShadowColor	XmGadget	XmNsensitive	RectObj
XmNbottomShadowPixmap	XmGadget	XmNshadowThickness	XmGadget
XmNborderWidth	RectObj	XmNtopShadowColor	XmGadget

Resource	Inherited From	Resource	Inherited From
XmNdestroyCallback	Object	XmNtopShadowPixmap	XmGadget
XmNforeground	XmGadget	XmNtraversalOn	XmGadget
XmNheight	RectObj	XmNunitType	XmGadget
XmNhelpCallback	XmGadget	XmNuserData	XmGadget
XmNhighlightColor	XmGadget	XmNwidth	RectObj
XmNhighlightOnEnter	XmGadget	XmNx	RectObj
XmNhighlightPixmap	XmGadget	XmNy	RectObj

Behavior

As a gadget subclass, ArrowButtonGadget has no translations associated with it. However, ArrowButtonGadget behavior corresponds to the action routines of the ArrowButton widget. See the ArrowButton action routines for more information.

Event	Action
BSelect Press	Arm()
BSelect Click	Activate() Disarm()
BSelect Release	Activate() Disarm()
Bselect Press 2+	MultiArm()
BSelect Release 2+	MultiActivate()
KSelect	ArmAndActivate()
MCtrl BSelect Press	ButtonTakeFocus()
KHelp	Help()

ArrowButtonGadget has additional behavior associated with <Enter> and <Leave>, which display the ArrowButtonGadget as selected if the pointer leaves and re-enters the gadget while BSelect is pressed or as unselected if the pointer leaves the gadget while BSelect is pressed.

See Also

`XmCreateObject`(1), `Core`(2), `Object`(2), `RectObj`(2), `XmArrowButton`(2), `XmGadget`(2), `XmPrimitive`(2).

Name

XmBulletinBoard widget class – a simple geometry-managing widget.

Synopsis

Public Header:
<Xm/BulletinB.h>

Class Name:
XmBulletinBoard

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmBulletinBoard

Class Pointer:
xmBulletinBoardWidgetClass

Instantiation:
widget = XmCreateBulletinBoard (parent, name,...)
or
widget = XtCreateWidget (name, xmBulletinBoardWidgetClass,...)

Functions/Macros:
`XmCreateBulletinBoard`(), `XmCreateBulletinBoardDialog`()

Description

BulletinBoard is a general-purpose manager that allows children to be placed at arbitrary x, y positions. The simple geometry management of BulletinBoard can be used to enforce margins and to prevent child widgets from overlapping. BulletinBoard is the base widget for most dialog widgets and defines many resources that have an effect only when it is an immediate child of a DialogShell.

Traits

BulletinBoard holds the XmQTspecifyRenderTable and XmQTdialogShellSavvy traits, which are inherited by any derived classes, and uses the XmQTtakesDefault and XmQTspecifyRenderTable traits.

New Resources

BulletinBoard defines the following resources:

Name	Class	Type	Default	Access
XmNallowOverlap	XmCAllowOverlap	Boolean	True	CSG
XmNautoUnmanage	XmCAutoUnmanage	Boolean	True	CSG
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNcancelButton	XmCWidget	Widget[a]	NULL	SG

Name	Class	Type	Default	Access
XmNdefaultButton	XmCWidget	Widget[b]	NULL	SG
XmNdefaultPosition	XmCDefaultPosition	unsigned char	True	CSG
XmNdialogStyle	XmCDialogStyle	unsigned char	dynamic	CSG
XmNdialogTitle	XmCDialogTitle	XmString	NULL	CSG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	10	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNnoResize	XmCNoResize	Boolean	False	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_ANY	CSG
XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_OUT	CSG
XmNtextFontList	XmCTextFontList	XmFontList	dynamic	CSG
XmNtextRenderTable	XmCTextRenderTable	XmRenderTable	dynamic	CSG
XmNtextTranslations	XmCTranslations	XtTranslations	NULL	C

a.Erroneously given as Window in 2nd Edition.
b.Erroneously given as Window in 2nd Edition.

XmNallowOverlap
If True (default), child widgets are allowed to overlap.

XmNautoUnmanage
If True (default), the BulletinBoard is automatically unmanaged after a button is activated unless the button is an **Apply** or **Help** button.

XmNbuttonFontList
Specifies the font list used for the button descendants of the BulletinBoard widget. In Motif 2.0 and later, the XmFontList is considered obsolete, and is replaced by the XmRenderTable. The XmNbuttonRenderTable resource is the preferred method of specifying appearance.

XmNbuttonRenderTable
In Motif 2.0 and later, specifies the render table used for any button descendants of the BulletinBoard widget. If NULL, this is inherited from the nearest ancestor that has the XmQTspecifyRenderTable trait, using the XmBUTTON_RENDER_TABLE value of any ancestor so found.
The button render table resource takes precedence over any specified XmNbuttonFontList.

XmNcancelButton
The widget ID of the **Cancel** button. The subclasses of BulletinBoard define a **Cancel** button and set this resource.

XmNdefaultButton
: The widget ID of the default button. Some of the subclasses of BulletinBoard define a default button and set this resource. To indicate that it is the default, this button appears different from the others.

XmNdefaultPosition
: If True (default) and if the BulletinBoard is the child of a DialogShell, then the BulletinBoard is centered relative to the DialogShell's parent.

XmNdialogStyle
: The BulletinBoard's dialog style, whose value can be set only if the BulletinBoard is unmanaged. Possible values:

```
XmDIALOG_WORK_AREA          /*default when parent is not a DialogShell */
XmDIALOG_MODELESS           /*default when parent is a DialogShell    */*
XmDIALOG_FULL_APPLICATION_MODAL
XmDIALOG_APPLICATION_MODAL
XmDIALOG_PRIMARY_APPLICATION_MODAL
XmDIALOG_SYSTEM_MODAL
```

  The value XmDIALOG_APPLICATION_MODAL, although maintained for backwards compatibility, is deprecated in Motif 1.2 and later. Use XmDIALOG_PRIMARY_APPLICATION_MODAL instead.

XmNdialogTitle
: The dialog title. Setting this resource also sets the resources XmNtitle and XmN--titleEncoding in a parent that is a subclass of WMShell.

XmNlabelFontList
: Specifies the font list used for the label descendants of the BulletinBoard widget. In Motif 2.0 and later, the XmFontList is considered obsolete, and is replaced by the XmRenderTable. The XmNlabelRenderTable resource is the preferred method of specifying appearance.

XmNlabelRenderTable
: In Motif 2.0 and later, specifies the render table used for any label descendants of the BulletinBoard widget. If NULL, this is inherited from the nearest ancestor that has the XmQTspecifyRenderTable trait, using the XmLABEL_RENDER_TABLE value of any ancestor so found.

XmNmarginHeight
: Minimum spacing between a BulletinBoard's top or bottom edge and any child widget.

XmNmarginWidth
: Minimum spacing between a BulletinBoard's right or left edge and any child widget.

XmNnoResize
If False (default), *mwm* includes resize controls in the window manager frame of the BulletinBoard's shell parent.

XmNresizePolicy
How BulletinBoard widgets are resized. Possible values:

```
XmRESIZE_NONE     /* remain at fixed size       */
XmRESIZE_GROW     /* expand only                */
XmRESIZE_ANY      /* shrink or expand, as needed */
```

XmNshadowType
The style in which shadows are drawn. Possible values:

```
XmSHADOW_IN           /* widget appears inset            */
XmSHADOW_OUT          /* widget appears outset           */
XmSHADOW_ETCHED_IN    /* double line; widget appears inset */
XmSHADOW_ETCHED_OUT   /* double line; widget appears raised */
```

XmNtextFontList
Specifies the font list used for the text descendants of the BulletinBoard widget. In Motif 2.0 and later, the XmFontList is considered obsolete, and is replaced by the XmRenderTable. The XmNtextRenderTable resource is the preferred method of specifying appearance.

XmNtextRenderTable
In Motif 2.0 and later, specifies the render table used for any text descendants of the BulletinBoard widget. If NULL, this is inherited from the nearest ancestor that has the XmQTspecifyRenderTable trait, using the XmTEXT_RENDER_TABLE value of any ancestor so found.

XmNtextTranslations
For any Text widget (or its subclass) that is a child of a BulletinBoard, this resource adds translations.

Callback Resources

BulletinBoard defines the following callback resources:

Callback	Reason Constant
XmNfocusCallback	XmCR_FOCUS
XmNmapCallback	XmCR_MAP
XmNunmapCallback	XmCR_UNMAP

XmNfocusCallback
List of callbacks that are called when the widget or one of its descendants receives the input focus.

XmNmapCallback
List of callbacks that are called when the widget is mapped, if it is a child of a DialogShell.

XmNunmapCallback
List of callbacks that are called when the widget is unmapped, if it is a child of a DialogShell.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* the reason that the callback was called */
    XEvent      *event;         /* points to event structure that triggered callback */
} XmAnyCallbackStruct;
```

Inherited Resources

BulletinBoard inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. BulletinBoard sets the value of XmNinitialFocus to the value of XmNdefaultButton. When it is a child of a DialogShell, BulletinBoard resets the default XmNshadowThickness from 0 to 1. The default value of XmNborderWidth is reset to 0 by Manager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core

Resource	Inherited From	Resource	Inherited From
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Translations

The translations for BulletinBoard include those of XmManager.

Additional Behavior

BulletinBoard has the following additional behavior:

MAny KCancel
: For a sensitive **Cancel** button, invokes the XmNactivateCallback callbacks.

KActivate
: For the button that has keyboard focus, invokes the XmNactivate-Callback callbacks.

<FocusIn>
: Invokes the XmNfocusCallback callbacks. The widget receives focus either when the user traverses to it (XmNkeyboardFocusPolicy is XmEXPLICIT) or when the pointer enters the window (XmNkeyboardFocusPolicy is XmPOINTER).

<Map>
: Invokes the XmNmapCallback callbacks.

<Unmap>
: Invokes the XmNunmapCallback callbacks.

See Also

`XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmBulletinBoardDialog`(2), `XmDialogShell`(2), `XmManager`(2).

Name

XmBulletinBoardDialog – an unmanaged BulletinBoard as a child of a DialogShell.

Synopsis

Public Header:
<Xm/BulletinB.h>

Instantiation:
widget = `XmCreateBulletinBoardDialog`(...)

Functions/Macros:
`XmCreateBulletinBoardDialog`()

Description

An XmBulletinBoardDialog is a compound object created by a call to XmCreateBulletinBoardDialog() that is useful for creating custom dialogs. A BulletinBoardDialog consists of a DialogShell with an unmanaged BulletinBoard widget as its child. The BulletinBoardDialog does not contain any labels, buttons, or other dialog components; these components are added by the application.

Default Resource Values

A BulletinBoardDialog sets the following default values for BulletinBoard resources:

Name	Default
XmNdialogStyle	XmDIALOG_MODELESS

Widget Hierarchy

When a BulletinBoardDialog is created with a specified *name*, the DialogShell is named *name_popup* and the BulletinBoard is called *name*.

See Also

`XmCreateObject`(1), `XmBulletinBoard`(2), `XmDialogShell`(2).

Name

XmCascadeButton widget class – a button widget that posts menus.

Synopsis

Public Header:
<Xm/CascadeB.h>

Class Name:
XmCascadeButton

Class Hierarchy:
Core →XmPrimitive →XmLabel →XmCascadeButton

Class Pointer:
xmCascadeButtonWidgetClass

Instantiation:
widget = `XmCreateCascadeButton` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmCascadeButtonWidgetClass,...)

Functions/Macros:
`XmCascadeButtonHighlight`(), `XmCreateCascadeButton`(), `XmIsCascadeButton`()

Description

CascadeButtons are used in menu systems to post menus. A CascadeButton either links a menu bar to a menu pane or connects a menu pane to another menu pane. The widget can have a menu attached to it as a submenu.

Traits

CascadeButton uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

New Resources

CascadeButton defines the following resources:

Name	Class	Type	Default	Access
XmNcascadePixmap	XmCPixmap	Pixmap	dynamic	CSG
XmNmappingDelay	XmCMappingDelay	int	180	CSG
XmNsubMenuId	XmCMenuWidget	Widget	NULL	CSG

XmNcascadePixmap
The pixmap within the CascadeButton that indicates a submenu. By default, this pixmap is an arrow pointing toward the submenu to be popped up.

XmNmappingDelay
The number of milliseconds it should take for the application to display a submenu after its CascadeButton has been selected.

XmNsubMenuId
The widget ID of the pulldown menu pane associated with the CascadeButton. The menu pane is displayed when the CascadeButton is selected. The pulldown menu pane and the CascadeButton must have a common parent.

Callback Resources

CascadeButton defines the following callback resources:

Callback	Reason Constant
XmNactivateCallback	XmCR_ACTIVATE
XmNcascadingCallback	XmCR_CASCADING

XmNactivateCallback
List of callbacks that are called when BSelect is pressed and released while the pointer is inside the widget and there is no submenu to post.

XmNcascadingCallback
List of callbacks that are called before the submenu associated with the CascadeButton is mapped.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int     reason;     /* the reason that the callback was called */
    XEvent  *event;     /* event structure that triggered callback */
} XmAnyCallbackStruct;
```

Inherited Resources

CascadeButton inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. CascadeButton sets the default values of XmNmarginBottom, XmNmarginRight, XmNmarginTop, XmNmarginWidth, and XmN-traversalOn dynamically. In Motif 2.0 and earlier, the default values of XmNhighlightThickness and XmNshadowThickness are reset to 2. In Motif 2.1, the default values depend upon the XmDisplay XmNenableThinThickness resource: if True the default is 1, otherwise 2. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNaccelerator	XmLabel	XmNlabelType	XmLabel
XmNaccelerators	Core	XmNlayoutDirection	XmPrimitive
XmNacceleratorText	XmLabel	XmNmappedWhenManaged	Core
XmNalignment	XmLabel	XmNmarginBottom	XmLabel
XmNancestorSensitive	Core	XmNmarginHeight	XmLabel
XmNbackground	Core	XmNmarginLeft	XmLabel
XmNbackgroundPixmap	Core	XmNmarginRight	XmLabel
XmNborderColor	Core	XmNmarginTop	XmLabel
XmNborderPixmap	Core	XmNmarginWidth	XmLabel
XmNborderWidth	Core	XmNmnemonicCharSet	XmLabel
XmNbottomShadowColor	XmPrimitive	XmNmnemonic	XmLabel
XmNbottomShadowPixmap	XmPrimitive	XmNnavigationType	XmPrimitive
XmNcolormap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNrecomputeSize	XmLabel
XmNdepth	Core	XmNrenderTable	XmLabel
XmNdestroyCallback	Core	XmNscreen	Core
XmNfontList	XmLabel	XmNsensitive	Core
XmNforeground	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNheight	Core	XmNstringDirection	XmLabel
XmNhelpCallback	XmPrimitive	XmNtopShadowColor	XmPrimitive
XmNhighlightColor	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNhighlightOnEnter	XmPrimitive	XmNtranslations	Core
XmNhighlightPixmap	XmPrimitive	XmNtraversalOn	XmPrimitive
XmNhighlightThickness	XmPrimitive	XmNunitType	XmPrimitive
XmNinitialResourcesPersistent	Core	XmNuserData	XmPrimitive
XmNlabelInsensitivePixmap	XmLabel	XmNwidth	Core
XmNlabelPixmap	XmLabel	XmNx	Core
XmNlabelString	XmLabel	XmNy	Core

Translations

The translations of CascadeButton include the menu traversal translations of Label.

Event	Action
BSelect Press	MenuBarSelect() (in a menu bar) StartDrag() (in a popup or pulldown menu)
BSelect Release	DoSelect()
MCtrl BSelect Press	MenuButtonTakeFocus()
MCtrl BSelect Release	MenuButtonTakeFocusUp()
KActivate	KeySelect()
KSelect	KeySelect()
KHelp	Help()
MAny KCancel	CleanupMenuBar()

Action Routines

CascadeButton defines the following action routines:

CleanupMenuBar()

Unposts any menus and restores the keyboard focus to the group of widgets (tab group) that had the focus before the CascadeButton was armed.

DoSelect()

Posts the CascadeButton's submenu and allows keyboard traversal. If there is no submenu attached to the CascadeButton, this action routine activates the CascadeButton and unposts all the menus in the cascade.

Help()

Similar to CleanupMenuBar() in that the Help() routine unposts any menus and restores keyboard focus. This routine also invokes the list of callbacks specified by XmNhelpCallback. If the CascadeButton doesn't have any help callbacks, the Help() routine invokes those associated with the nearest ancestor that has them.

KeySelect()

Posts the CascadeButton's submenu, provided that keyboard traversal is allowed. If there is no submenu attached to the CascadeButton, this action routine activates the CascadeButton and unposts all the menus in the cascade.

MenuBarSelect()
: Unposts any previously posted menus, posts the submenu associated with the CascadeButton, and enables mouse traversal.

MenuButtonTakeFocus()
: In Motif 2.0 and later, moves the current keyboard focus to the CascadeButton, without activating the widget.

StartDrag()
: Posts the submenu associated with the CascadeButton and enables mouse traversal.

Additional Behavior

CascadeButton has the following additional behavior:

<EnterWindow>	Arms the CascadeButton and posts its submenu.
<LeaveWindow>	Disarms the CascadeButton and unposts its submenu.

See Also

`XmCascadeButtonHighlight`(1), `XmCreateObject`(2), `Core`(2), `XmLabel`(2), `XmPrimitive`(2), `XmRowColumn`(2).

Name

XmCascadeButtonGadget widget class – a button gadget that posts menus.

Synopsis

Public Header:
<Xm/CascadeBG.h>

Class Name:
XmCascadeButtonGadget

Class Hierarchy:
Object →RectObj →XmGadget →XmLabelGadget →XmCascadeButtonGadget

Class Pointer:
xmCascadeButtonGadgetClass

Instantiation:
widget = `XmCreateCascadeButtonGadget` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmCascadeButtonGadgetClass,...)[1]

Functions/Macros:
`XmCascadeButtonGadgetHighlight`(), `XmCreateCascadeButtonGadget`(), `XmIsCascadeButtonGadget`(), `XmOptionButtonGadget`()

Description

CascadeButtonGadget is the gadget variant of CascadeButton.

CascadeButtonGadget's new resources, callback resources, and callback structure are the same as those for CascadeButton.

Traits

CascadeButtonGadget uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

Inherited Resources

CascadeButtonGadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. CascadeButtonGadget sets the default values of XmNmarginBottom, XmNmarginRight, XmNmarginTop, and XmNmarginWidth dynamically. It also sets the default value of XmNhighlightThickness to 0. The default value of XmNborderWidth is reset to 0 by Gadget.

1.Erroneously given as xmCascadeButtonWidgetClass in 2nd Edition.

Resource	Inherited From	Resource	Inherited From
XmNaccelerator	XmLabelGadget	XmNmarginBottom	XmLabelGadget
XmNacceleratorText	XmLabelGadget	XmNmarginHeight	XmLabelGadget
XmNalignment	XmLabelGadget	XmNmarginLeft	XmLabelGadget
XmNancestorSensitive	RectObj	XmNmarginRight	XmLabelGadget
XmNbackground	XmGadget	XmNmarginTop	XmLabelGadget
XmNbackgroundPixmap	XmGadget	XmNmarginWidth	XmLabelGadget
XmNbottomShadowColor	XmGadget	XmNmnemonic	XmLabelGadget
XmNbottomShadowPixmap	XmGadget	XmNmnemonicCharSet	XmLabelGadget
XmNborderWidth	RectObj	XmNnavigationType	XmGadget
XmNdestroyCallback	Object	XmNrecomputeSize	XmLabelGadget
XmNfontList	XmLabelGadget	XmNrenderTable	XmLabelGadget
XmNforeground	XmGadget	XmNsensitive	RectObj
XmNheight	RectObj	XmNshadowThickness	XmGadget
XmNhelpCallback	XmGadget	XmNstringDirection	XmLabelGadget
XmNhighlightColor	XmGadget	XmNtopShadowColor	XmGadget
XmNhighlightOnEnter	XmGadget	XmNtopShadowPixmap	XmGadget
XmNhighlightPixmap	XmGadget	XmNtraversalOn	XmGadget
XmNhighlightThickness	XmGadget	XmNunitType	XmGadget
XmNlabelInsensitivePixmap	XmLabelGadget	XmNuserData	XmGadget
XmNlabelPixmap	XmLabelGadget	XmNwidth	RectObj
XmNlabelType	XmLabelGadget	XmNx	RectObj
XmNlayoutDirection	XmGadget	XmNy	RectObj

Behavior

As a gadget subclass, CascadeButtonGadget has no translations associated with it. However, CascadeButtonGadget behavior corresponds to the action routines of the CascadeButton widget. See the CascadeButton action routines for more information.

Event	Action
BSelect Press	MenuBarSelect() (in a menu bar) StartDrag() (in a popup or pulldown menu)
BSelect Release	DoSelect()
MCtrl BSelect Press	MenuButtonTakeFocus()

Event	Action
MCtrl BSelect Release	MenuButtonTakeFocusUp()
KActivate	KeySelect()
KSelect	KeySelect()
KHelp	Help()
MAny KCancel	CleanupMenuBar()

In a menu bar that is armed, CascadeButtonGadget has additional behavior associated with <Enter>, which arms the CascadeButtonGadget and posts its submenu, and with <Leave>, which disarms the CascadeButtonGadget and unposts its submenu.

See Also

`XmCascadeButtonHighlight`(1), `XmCreateObject`(1), `XmOptionButtonGadget`(1), `Object`(2), `RectObj`(2), `XmCascadeButton`(2), `XmGadget`(2), `XmLabelGadget`(2), `XmRowColumn`(2).

Name

XmCheckBox – a RowColumn that contains ToggleButtons.

Synopsis

Public Header:
<Xm/RowColumn.h>

Class Name:
XmRowColumn

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmRowColumn

Class Pointer:
xmRowColumnWidgetClass

Instantiation:
widget = `XmCreateSimpleCheckBox` (parent, name,...)

Functions/Macros:
`XmCreateRowColumn`(), `XmCreateSimpleCheckBox`(), `XmIsRowColumn`(),
`XmVaCreateSimpleCheckBox`()

Description

An XmCheckBox is an instance of a RowColumn widget that contains ToggleButton or ToggleButtonGadget children, any number of which may be selected at a given time. A CheckBox is a RowColumn widget with its XmNrowColumnType resource set to XmWORK_AREA and XmNradioAlwaysOne set to False.

A CheckBox can be created by making a RowColumn with these resource values. When it is created in this way, a CheckBox does not automatically contain ToggleButton children; they are added by the application.

A CheckBox can also be created by a call to XmCreateSimpleCheckBox() or XmVaCreateSimpleCheckBox(). These routines automatically create the CheckBox with ToggleButtonGadgets as children. The routines use the RowColumn resources associated with the creation of simple menus. For a CheckBox, the only type allowed in the XmNbuttonType resource is XmCHECKBUTTON. The name of each ToggleButtonGadget is *button_n*, where *n* is the number of the button, ranging from 0 to 1 less than the number of buttons in the CheckBox.

Default Resource Values

A CheckBox sets the following default values for its resources:

Name	Default
XmNnavigationType	XmTAB_GROUP
XmNradioBehavior	False
XmNrowColumnType	XmWORK_AREA
XmNtraversalOn	True

See Also

`XmCreateObject`(2), `XmVaCreateSimpleCheckBox`(2), `XmRowColumn`(2), `XmToggleButton`(2), `XmToggleButtonGadget`(2).

Name

XmComboBox widget class – a composite widget which combines a text widget with a list of choices.

Synopsis

Public Header:
<Xm/ComboBox.h>

Class Name:
XmComboBox

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmComboBox

Class Pointer:
xmComboBoxWidgetClass

Instantiation:
widget = `XmCreateComboBox` (parent, name,...)
or
widget = `XmCreateDropDownComboBox` (parent, name,...)
or
widget = `XmCreateDropDownList` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmComboBoxWidgetClass,...)

Functions/Macros:
`XmComboBoxAddItem`(), `XmComboBoxDeletePos`(), `XmComboBoxSelectItem`(), `XmComboBoxSetItem`(), `XmComboBoxUpdate`(), `XmCreateComboBox`(), `XmCreateDropDownComboBox`(), `XmCreateDropDownList`(), `XmIsComboBox`()

Availability

Motif 2.0 and later.

Description

ComboBox is a composite widget that combines both text entry and scrolled list selection. The ComboBox can be configured in various ways, depending on whether the text field is to be editable, and whether the scrolled list is to be permanently visible. The text field contains the currently selected item; an item selected from the list is automatically placed into the text field. Whether entering data into the text field, or selecting from the list, the user can select only one item at any given time. Data typed directly into the text field does not, however, automatically select any matching item in the list.

By default, both the text field and the list are fully visible, and the list is placed underneath the text widget. The user can either enter characters directly into the text field, or select an item from the list. Alternatively, the ComboBox widget can be configured such that the list is hidden until required. In this case, the ComboBox widget draws an arrow button adjacent to the text field. Clicking on the arrow displays the hidden list underneath the text field. Selecting an item from the displayed popup list automatically pops down the list.

The XmNcomboBoxType resource configures the type of the ComboBox. You may specify one of the following types: combo box (XmCOMBO_BOX), drop-down combo box (XmDROP_DOWN_COMBO_BOX), or drop-down list (XmDROP_DOWN_LIST). The combo box type has a permanently visible list, and the text field is editable. A drop-down combo box has an editable text field, the list is hidden, and the arrow button is drawn. A drop-down list is identical to a drop-down combo box except that the text field is not editable: the user must select from the list to change the selection.

If the ComboBox widget has a non-editable text field, any characters typed do not appear directly in the text field. Instead, any characters entered may be compared against items in the list, depending on the value of the XmNmatchBehavior resource. When enabled, and the list has the input focus, characters typed are compared against the first character of each item in the list. If a match is found, the matched list item is automatically selected. Subsequently typing the same character progresses cyclically through the list to find any further matching item.

The position of the drawn arrow button relative to the text field depends upon the XmNlayoutDirection resource of the Shell ancestor of the ComboBox widget. If XmNlayoutDirection is XmLEFT_TO_RIGHT, the arrow button is drawn to the right of the text field. Otherwise, a value XmRIGHT_TO_LEFT draws the arrow button to the left.

Traits

ComboBox uses the XmQTaccessTextual and XmQTspecifyRenderTable traits.

New Resources

ComboBox defines the following resources:

Name	Class	Type	Default	Access
XmNarrowSize	XmCArrowSize	Dimension	dynamic	CSG
XmNarrowSpacing	XmCArrowSpacing	Dimension	dynamic	CSG
XmNcolumns	XmCColumns	short	dynamic	CSG
XmNcomboBoxType	XmCComboBoxType	unsigned char	XmCOMBO_BOX	CG
XmNfontList	XmCFontList	XmFontList	NULL	CSG

Name	Class	Type	Default	Access
XmNhighlightThickness	XmCHighlightThickness	Dimension	dynamic	CSG
XmNitemCount	XmCItemCount	int	dynamic	CSG
XmNitems	XmCItems	XmStringTable	dynamic	CSG
XmNlist	XmCList	Widget	dynamic	G
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNmatchBehavior	XmCMatchBehavior	unsigned char	dynamic	CSG
XmNpositionMode	XmCPositionMode	XtEnum	XmZERO_BASED	CG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNselectedItem	XmCSelectedItem	XmString	NULL	CSG
XmNselectedPosition	XmCSelectedPosition	int	0	CSG
XmNtextField	XmCTextField	Widget	dynamic	G
XmNvisibleItemCount	XmCVisibleItemCount	int	10	CSG

XmNarrowSize
: The size of the drawn arrow button used to display the hidden list. The default size depends upon the size of the text field, and the size of the ComboBox widget. Attempting to change the arrow size may result in a geometry request to the parent of the ComboBox widget. If this request is refused, the arrow size is set to the maximum size that fits into the space allowed by the parent of the ComboBox widget.

XmNarrowSpacing
: The horizontal spacing, in pixels, between the drawn arrow button and the text field. The default value is calculated from the value of the XmNmarginWidth resource.

XmNcolumns
: Specifies the number of columns in the text field. The default value is that of XmTextField.

XmNcomboBoxType
: Specifies the type of the ComboBox. Possible values:

```
XmCOMBO_BOX                 /* visible list; editable text field     */
XmDROP_DOWN_COMBO_BOX       /* hidden list; editable text field     */
XmDROP_DOWN_LIST            /* hidden list; non-editable text field */
```

XmNfontList
: The font list used for the items in the list and text field. In Motif 2.0 and later, the XmFontList is obsolete as a data type, and the resource is maintained for backwards compatibility through an implementation as an XmRenderTable. The

XmNrenderTable resource is the preferred method of specifying appearance. If both a render table and font list are specified, the render table takes precedence. The default font list is taken from the default render table.

XmNhighlightThickness
The thickness of the highlighting rectangle. In Motif 2.0, the default is 2. In Motif 2.1 and later, the default depends upon the value of the XmDisplay resource XmNenableThinThickness: if True, the default is 1, otherwise 2.

XmNitemCount
The total number of items. If unspecified, the value is taken from the internal list. The ComboBox widget updates this resource every time a list item is added or removed through the ComboBox convenience functions.

XmNitems
A pointer to an array of compound strings, representing the items to display in the list. A call to XtGetValues() returns the actual list items, not a copy. Applications should not directly free any items fetched in this manner. If the resource is unspecified, the value is taken from the internal list. The ComboBox widget updates this resource every time a list item is added or removed through the ComboBox convenience functions.

XmNlist
The list widget created by the ComboBox. Applications may not change the value of this resource, but may fetch the value to perform required operations on the internal list.

XmNmarginHeight
The minimum spacing between the ComboBox top or bottom edge and the child list and text field widgets.

XmNmarginWidth
The minimum spacing between the ComboBox left or right edge and the child list and text field widgets.

XmNmatchBehavior
Determines whether matching behavior is enabled, where characters typed are compared against items in the list. Possible values:

```
XmNONE                  /* No match behavior     */
XmQUICK_NAVIGATE        /* Match behavior enabled */
```

The value XmQUICK_NAVIGATE may only be specified if the XmNcomboBoxType resource has value XmDROP_DOWN_LIST. XmQUICK_NAVIGATE is the default when XmNcomboBoxType is XmDROP_DOWN_LIST. Otherwise, XmNONE is the default.

XmNpositionMode
Specifies the way in which the position of the selected item is reported in callbacks, and controls the initial index value of the XmNselectedPosition resource. Possible values:

```
XmZERO_BASED        /* first item in list is position zero */
XmONE_BASED         /* first item in list is position one */
```

A value of XmZERO_BASED configures callback data on the XmNselectionCallback list such that the item_position element of the XmComboBoxCallbackStruct is indexed from zero: selecting the first list item has item_position set to zero, selecting the second item has item_position as one, and so forth. Similarly, fetching the XmNselectedPosition resource when the first item in the list is selected will return the value zero.

A value of XmONE_BASED sets the item_position element such that selecting the first item in the list is reported at position one, the second item is reported at position two, and so on. By analogy, fetching the XmNselectedPosition resource when the first item in the list is selected will return the value one.

In all cases, changes to the text field are reported with item_position set to zero.

A XmNpositionMode of XmONE_BASED therefore makes it easier to distinguish between text field and list selection, since item_position set to zero is not ambiguous.

The ComboBox convenience functions for adding, deleting, or selecting items in the list are unaffected by the value of this resource: these functions always assume that the first item is at position one.

This resource is provided for CDE compatibility. In particular, setting the value to XmZERO_BASED makes the ComboBox selection behavior consistent with that of the DtComboBox widget.

XmNrenderTable
Specifies the render table for the ComboBox, and is used in both the text field and list children. If NULL, this is inherited from the nearest ancestor that has the XmQTspecifyRenderTable trait. The BulletinBoard, VendorShell, and MenuShell widgets and derived classes set this trait.
The render table resource takes precedence over any specified XmNfontList.

XmNselectedItem
A compound string representing the currently selected item contained within the ComboBox text field.

XmNselectedPosition
: Identifies the index of the XmNselectedItem in the list. The interpretation of the index depends upon the value of the XmNpositionMode resource. If XmNpositionMode is XmZERO_BASED, a XmNselectedPosition value of 0 (zero) indicates that the first list item is selected, a value of 1 indicates the second item is selected, and so forth. If XmNpositionMode is XmONE_BASED, the value 1 indicates that the first list item is selected, the value 2 indicates the second list item, and so on, with value 0 (zero) indicating that no list item is selected.

XmNtextField
: The text field widget created by the ComboBox. Applications may not change the value of this resource, but may fetch the value to perform required operations on the internal text field.

XmNvisibleItemCount
: The number of items to display in the work area of the list. If specified, this resource overrides the XmNvisibleItemCount resource of the internal list widget. The resource may affect the height of the list widget, and hence the ComboBox itself, depending upon whether the list is permanently visible. If the XmNvisibleItemCount value is less than the number of items in the list, the list is automatically configured with a vertical ScrollBar.

Callback Resources

ComboBox defines the following callback resources:

Callback	Reason Constant
XmNselectionCallback	XmCR_SELECT

XmNselectionCallback
: List of callbacks that are called when a selection occurs in the ComboBox widget.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* the reason that the callback was called         */
    XEvent      *event;         /* points to event structure that triggered callback */
    XmString    item_or_text;   /* the selected item                               */
    int         item_position;  /* the index of the item in the list               */
} XmComboBoxCallbackStruct;
```

The item_or_text element is a compound string representing the ComboBox selected item. It points to allocated memory that is reclaimed after the callbacks on the XmNselectionCallback list have returned. If the item is to be cached, the application should copy the item using XmStringCopy().

The item_position element specifies the index of the selected item within the XmNitems array of the list. The interpretation of the value will depend upon the XmNpositionMode resource of the ComboBox widget. With XmNpositionMode set to XmONE_BASED, an item_position of 1 refers to the first item in the list, and an item_position of zero indicates that the selected item has been entered directly into the text field (no list selection). With a XmNpositionMode of XmZERO_BASED, an item_position of zero could either mean that the first item in the list is selected, or that the selection is from direct text entry, and an item_position of 1 refers to the second list item.

Default Resource Values

A ComboBox sets the following default values for the scrolled list resources:

Name	Default
XmNborderWidth	0
XmNhighlightThickness	dynamic
XmNlistSizePolicy	XmVARIABLE
XmNnavigationType	XmNONE
XmNrenderTable	dynamic
XmNselectionPolicy	XmBROWSE_SELECT
XmNspacing	0
XmNtraversalOn	dynamic
XmNvisualPolicy	XmVARIABLE

If the ComboBox is a drop down list, XmNcursorPositionVisible and XmNeditable are set to False, and the XmNshadowThickness is set to zero. Otherwise, XmNcursorPositionVisible and XmNeditable are set True, and XmNeditMode is set to XmSINGLE_LINE_EDIT.

Inherited Resources

ComboBox inherits the resources shown below. The resources are listed alphabetically, along with the superclass that defines them. ComboBox sets the default values of XmNmarginWidth, and XmNmarginHeight to 2, and XmNnavigationType to XmSTICKY_TAB_GROUP. The default value of XmNborderWidth is reset to 0 (zero) by Manager. The default values for XmNhighlightThickness and XmNshadowThickness depend upon the XmDisplay XmNenableThinThickness resource: if True, the default is 1, otherwise 2.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Widget Hierarchy

The ComboBox creates the text field with the name Text, and the scrolled list is named List. If the ComboBox is of a drop down type, an XmGrabShell named GrabShell is created as parent to the scrolled list.

Translations

The translations for ComboBox include those of XmManager.

Event	Action
BSelect Press	CBArmAndDropDownList()
BSelect Release	CBDisarm()

ComboBox places the following translations upon the list:

Event	Action
KDown	CBDropDownList()
KUp	CBDropDownList()
KCancel	CBCancel()
KActivate	CBActivate()
MShift KBeginData	CBListAction(ListBeginData)
MShift KEndData	CBListAction(ListEndData)
KPageUp	CBListAction(ListPrevPage)
KPageDown	CBListAction(ListNextPage)

ComboBox places the following translations upon the text field:

Event	Action
<FocusOut>	CBTextFocusOut()

Action Routines

ComboBox defines the following action routines:

CBArmAndDropDownList()
: If the mouse is over the drawn arrow button, draws the arrow button as though selected, and posts the drop-down list.

CBDisarm()
: Draws the arrow button in unselected state.

CBDropDownList()
: Posts the drop-down list

CBFocusIn()
: Draws focus highlighting around the ComboBox widget.

CBFocusOut()
: Erases focus highlighting around the ComboBox widget. If the text field has changed, invokes the list of callbacks specified by XmNselectionCallback.

CBCancel()
: Pops down the drop-down list, and draws the arrow button in unselected state.

CBActivate()
: Fetches the value from the text field. If the XmNcomboBoxType is XmCOMBO_BOX, invokes the list of callbacks specified by XmNdefaultActionCallback for the internal list, passing the text field

value as the selected item within the callback data. Regardless of XmNcomboBoxType, if the value matches an item within the list, the list item is selected, otherwise all list items are deselected. Lastly, the list of callbacks specified by XmNselectionCallback is invoked.

CBListAction(type)

A generic action to perform operations on the internal list. The action type may be one of Up, Down, ListPrevPage, ListNextPage, ListBeginData, or ListEndData. The types Up and Down simply select the relevant item in the list in the required direction relative to the currently selected item. The remaining types directly invoke the ListPrevPage, ListNextPage, ListBeginData, or ListEndData actions of the list. The types Up and Down differ from the corresponding List actions in that the ComboBox actions will wrap around the items in the internal list.

CBTextFocusOut()

Turns off text field cursor blinking.

See Also

`XmComboBoxAddItem`(1), `XmComboBoxDeletePos`(1), `XmComboBoxSelectItem`(1), `XmComboBoxSetItem`(1), `XmComboBoxUpdate`(1), `XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmManager`(2).

Name

XmCommand widget class – a composite widget for command entry.

Synopsis

Public Header:
<Xm/Command.h>

Class Name:
XmCommand

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmBulletinBoard →
XmSelectionBox →XmCommand

Class Pointer:
xmCommandWidgetClass

Instantiation:
widget = `XmCreateCommand` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmCommandWidgetClass,...)

Functions/Macros:
`XmCommandAppendValue`(), `XmCommandError`(), `XmCommandGetChild`(),
`XmCommandSetValue`(), `XmCreateCommand`(), `XmIsCommand`()

Description

Command is a composite widget that handles command entry by providing a prompt, a command input field, and a history list region. Many of the Command widget's new resources are in fact renamed resources from SelectionBox.

New Resources

Command defines the following resources:

Name	Class	Type	Default	Access
XmNcommand	XmCTextString	XmString	NULL	CSG
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
The text currently displayed on the command line. Synonymous with the XmNtextString resource in SelectionBox. XmNcommand can be changed using the routines `XmCommandSetValue`() and `XmCommandAppendValue`().

XmNhistoryItems
The items in the history list. Synonymous with the XmNlistItems resource in SelectionBox. A call to `XtGetValues`() returns the actual list items (not a copy), so don't have your application free these items.

XmNhistoryItemCount
The number of strings in XmNhistoryItems. Synonymous with the XmNlistItemCount resource in SelectionBox.

XmNhistoryMaxItems
The history list's maximum number of items. When this number is reached, the first history item is removed before the new command is added to the list.

XmNhistoryVisibleItemCount
The number of history list commands that will display at one time. Synonymous with the XmNvisibleItemCount resource in SelectionBox.

XmNpromptString
The command-line prompt. Synonymous with the XmNselectionLabelString resource in SelectionBox.

Callback Resources

Command defines the following callback resources:

Callback	Reason Constant
XmNcommandEnteredCallback	XmCR_COMMAND_ENTERED
XmNcommandChangedCallback	XmCR_COMMAND_CHANGED

XmNcommandChangedCallback
List of callbacks that are called when the value of the command changes.

XmNcommandEnteredCallback
List of callbacks that are called when a command is entered in the widget.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;     /* the reason that the callback was called        */
    XEvent      *event;     /* points to event structure that triggered callback */
    XmString    value;      /* the string contained in the command area       */
    int         length;     /* the size of this string                        */
} XmCommandCallbackStruct;
```

Inherited Resources

Command inherits the resources shown below. The resources are listed alphabetically, along with the superclass that defines them. Command sets the default values of XmNautoUnmanage and XmNdefaultPosition to False, XmNdialogType to XmDIALOG_COMMAND, and XmNlistLabelString to NULL. In versions of Motif prior to 2.1.10, XmNresizePolicy is reset to XmRESIZE_NONE.

In Motif 2.1.10 and later, it is reset to XmRESIZE_ANY: this undocumented change is a bug which persists in Motif 2.1.20. The default value of XmNborderWidth is reset to 0 by Manager. BulletinBoard sets the value of XmNinitialFocus to XmNdefaultButton and resets the default XmNshadowThickness from 0 to 1 if the Command widget is a child of a DialogShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNlistItemCount	XmSelectionBox
XmNallowOverlap	XmBulletinBoard	XmNlistItems	XmSelectionBox
XmNancestorSensitive	Core	XmNlistLabelString	XmSelectionBox
XmNapplyCallback	XmSelectionBox	XmNlistVisibleItemCount	XmSelectionBox
XmNapplyLabelString	XmSelectionBox	XmNmapCallback	XmBulletinBoard
XmNautoUnmanage	XmBulletinBoard	XmNmappedWhenManaged	Core
XmNbackground	Core	XmNmarginHeight	XmBulletinBoard
XmNbackgroundPixmap	Core	XmNmarginWidth	XmBulletinBoard
XmNborderColor	Core	XmNminimizeButtons	XmSelectionBox
XmNborderPixmap	Core	XmNmustMatch	XmSelectionBox
XmNborderWidth	Core	XmNnavigationType	XmManager
XmNbottomShadowColor	XmManager	XmNnoMatchCallback	XmSelectionBox
XmNbottomShadowPixmap	XmManager	XmNnoResize	XmBulletinBoard
XmNbuttonFontList	XmBulletinBoard	XmNnumChildren	Composite
XmNbuttonRenderTable	XmBulletinBoard	XmNokCallback	XmSelectionBox
XmNcancelButton	XmBulletinBoard	XmNokLabelString	XmSelectionBox
XmNcancelCallback	XmSelectionBox	XmNpopupHandlerCallback	XmManager
XmNcancelLabelString	XmSelectionBox	XmNresizePolicy	XmBulletinBoard
XmNchildren	Composite	XmNscreen	Core
XmNchildPlacement	XmSelectionBox	XmNselectionLabelString	XmSelectionBox
XmNcolormap	Core	XmNsensitive	Core
XmNdefaultButton	XmBulletinBoard	XmNshadowThickness	XmManager
XmNdefaultPosition	XmBulletinBoard	XmNshadowType	XmBulletinBoard
XmNdepth	Core	XmNstringDirection	XmManager

Resource	Inherited From	Resource	Inherited From
XmNdestroyCallback	Core	XmNtextAccelerators	XmSelectionBox
XmNdialogStyle	XmBulletinBoard	XmNtextColumns	XmSelectionBox
XmNdialogTitle	XmBulletinBoard	XmNtextFontList	XmBulletinBoard
XmNdialogType	XmSelectionBox	XmNtextRenderTable	XmBulletinBoard
XmNfocusCallback	XmBulletinBoard	XmNtextString	XmSelectionBox
XmNforeground	XmManager	XmNtextTranslations	XmBulletinBoard
XmNheight	Core	XmNtopShadowColor	XmManager
XmNhelpCallback	XmManager	XmNtopShadowPixmap	XmManager
XmNhelpLabelString	XmSelectionBox	XmNtranslations	Core
XmNhighlightColor	XmManager	XmNtraversalOn	XmManager
XmNhighlightPixmap	XmManager	XmNunitType	XmManager
XmNinitialFocus	XmManager	XmNunmapCallback	XmBulletinBoard
XmNinitialResourcesPersistent	Core	XmNuserData	XmManager
XmNinsertPosition	Composite	XmNwidth	Core
XmNlabelFontList	XmBulletinBoard	XmNx	Core
XmNlabelRenderTable	XmBulletinBoard	XmNy	Core
XmNlayoutDirection	XmManager		

Translations

The translations for Command are inherited from XmSelectionBox.

Action Routines

Command defines the following action routines:

SelectionBoxUpOrDown(flag)

Selects a command from the history list, replaces the current command-line text with this list item, and invokes the callbacks specified by XmNcommandChangedCallback. The value of flag determines which history list command is selected. With a flag value of 0, 1, 2, or 3, this action routine selects the list's previous, next, first, or last item, respectively.

Additional Behavior

Command has the following additional behavior:

MAny KCancel

The event is passed to the parent if it is a manager widget.

KActivate
: In the Text widget, invokes the XmNactivateCallback callbacks, appends the text to the history list, and invokes the XmNcommandEnteredCallback callbacks.

<Key>
: In the Text widget, any keystroke that changes text invokes the XmNcommandChangedCallback callbacks.

KActivate or <DoubleClick>
: In the List widget, invokes the XmNdefaultActionCallback callbacks, appends the selected item to the history list, and invokes the XmNcommandEnteredCallback callbacks.

<FocusIn>
: Invokes the XmNfocusCallback callbacks.

<MapWindow>
: If the widget is a child of a DialogShell, invokes the XmNmapCallback callbacks when the widget is mapped.

<UnmapWindow>
: If the widget is a child of a DialogShell, invokes the XmNunmapCallback callbacks when the widget is unmapped.

See Also

`XmCommandAppendValue`(1), `XmCommandError`(1), `XmCommandGetChild`(1), `XmCommandSetValue`(1), `XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmBulletinBoard`(2), `XmManager`(2), `XmSelectionBox`(2).

Name

XmContainer widget class – a widget which controls the layout and selection of a set of items

Synopsis

Public Header:
<Xm/Container.h>

Class Name:
XmContainer

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmContainer

Class Pointer:
xmContainerWidgetClass

Instantiation:
widget = `XmCreateContainer` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmContainerWidgetClass,...)

Functions/Macros:
`XmContainerCopy`(), `XmContainerCopyLink`(), `XmContainerCut`(), `XmContainerGetItemChild`(), `XmContainerPaste`(), `XmContainerPasteLink`(), `XmContainerRelayout`(), `XmContainerReorder`(), `XmCreateContainer`()

Availability

Motif 2.0 and later.

Description

A Container is a constraint widget which controls the layout and selection of container items. Container is intended to provide an object-oriented view of the world: an application object can be represented in the Container as a container item. The user can subsequently manipulate the item by moving, copying, selecting, or deleting it, or perform drag and drop operations between applications using the item. New items can be dropped into the Container. At each stage, callbacks indicate to the application the operation performed upon each item, and hence the requested operation upon the object which it represents. The Container recognises as a container item any child widget which holds the XmQTcontainerItem trait. The IconGadget is the only standard Motif widget to hold this trait.

The Container provides three styles in which items can be displayed, specified through the XmNlayoutType resource. The resource can have the values XmOUTLINE, XmDETAIL, or XmSPATIAL.

The XmOUTLINE layout style provides a tree view onto the items, and is appropriate when application objects exist in a parent/child relationship to each other. The logical relationship between items is specified by setting the XmNentryParent constraint resource of an item to point to the logical parent. The order of items within the tree depends upon the XmNpositionIndex constraint value for each item. The Container draws connecting lines between items to indicate the relationships. The Container creates additional PushButtonGadgets which are used for folding and unfolding portions of the tree.

The XmDETAIL layout style gives a tabular format, where each row of the table represents an object, each cell within the row possibly representing a property of the object. The data attached to an object and displayed in the row is specified through the XmNdetail resources of the associated container item. Column headings can be specified through the XmNdetailColumnHeading resources of the Container.

The XmSPATIAL layout style provides generic layout, where the exact positioning of items is controlled through further resources. This can take the form of a grid layout, where items can span single or multiple cells of the grid, or a free format where items can be positioned at absolute x, y coordinates of the Container. At the simplest, the grid layout can be used to construct a general purpose icon box. The resources XmNspatialStyle, XmNspatialIncludeModel, and XmNspatialSnapModel control the positioning of items.

The Container controls the way in which items are selected, and provides selection notification. The widget supports single, browse, multiple, and extended selection. Selection of items within the Container can be performed by including them within a rubberband rectangle called a Marquee, which the user specifies using the mouse. Alternatively, selection can be performed by simply swiping the mouse over an item. The style of selection is specified through the XmNselectionTechnique resource.

The user can only move container items if the XmNlayoutType is XmSPATIAL.

Traits

Container holds the XmQTtransfer, XmQTtraversalControl, and XmQTcontainer traits, and uses the XmQTscrollFrame, XmQTcontainerItem, XmQTnavigator, XmQTspecifyRenderTable, and XmQTpointIn traits.

New Resources

Container defines the following resources:

Name	Class	Type	Default	Access
XmNautomaticSelection	XmCAutomaticSelection	unsigned char	XmAUTO_SELECT	CSG
XmNcollapsedStatePixmap	XmCCollapsedStatePixmap	Pixmap	dynamic	CSG
XmNdetailColumnHeading	XmCDetailColumnHeading	XmStringTable	NULL	CSG
XmNdetailColumnHeadingCount	XmCDetailColumnHeadingCount	Cardinal	0	CSG
XmNdetailOrder	XmCDetailOrder	Cardinal *	NULL	CSG
XmNdetailOrderCount	XmCDetailOrderCount	Cardinal	0	CSG
XmNdetailTabList	XmCDetailTabList	XmTabList	NULL	CSG
XmNentryViewType	XmCEntryViewType	unsigned char	XmANY_ICON	CSG
XmNexpandedStatePixmap	XmCExpandedStatePixmap	Pixmap	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNlargeCellHeight	XmCCellHeight	Dimension	0	CSG
XmNlargeCellWidth	XmCCellWidth	Dimension	0	CSG
XmNlayoutType	XmCLayoutType	unsigned char	XmSPATIAL	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	0	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	0	CSG
XmNoutlineButtonPolicy	XmCOutlineButtonPolicy	unsigned char	XmOUTLINE_BUTTON_PRESENT	CSG
XmNoutlineColumnWidth	XmCOutlineColumnWidth	Dimension	0	CSG
XmNoutlineIndentation	XmCOutlineIndentation	Dimension	40	CSG
XmNoutlineLineStyle	XmCOutlineLineStyle	unsigned char	XmSINGLE	CSG
XmNprimaryOwnership	XmCPrimaryOwnership	unsigned char	XmOWN_POSSIBLE_MULTIPLE	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNselectColor	XmCSelectColor	Pixel	XmREVERSED_GROUND_COLORS	CSG
XmNselectedObjectCount	XmCSelectedObjectCount	int	0	SG
XmNselectedObjects	XmCSelectedObjects	WidgetList	NULL	SG
XmNselectionPolicy	XmCSelectionPolicy	unsigned char	XmEXTENDED_SELECT	CSG
XmNselectionTechnique	XmCSelectionTechnique	unsigned char	XmTOUCH_OVER	CSG
XmNsmallCellHeight	XmCCellHeight	Dimension	0	CSG
XmNsmallCellWidth	XmCCellWidth	Dimension	0	CSG
XmNspatialIncludeModel	XmCSpatialIncludeModel	unsigned char	XmAPPEND	CSG
XmNspatialResizeModel	XmCSpatialResizeModel	unsigned char	XmGROW_MINOR	CSG
XmNspatialSnapModel	XmCSpatialSnapModel	unsigned char	XmNONE	CSG
XmNspatialStyle	XmCSpatialStyle	unsigned char	XmGRID	CSG

XmNautomaticSelection
Specifies whether selection callbacks are invoked immediately and each time that an item is selected, or whether callbacks are invoked when the user has completed the selection action. Possible values:

```
XmAUTO_SELECT       /* callbacks called immediately on selection    */
XmNO_AUTO_SELECT    /* callbacks delayed until user action completed
*/
```

XmNcollapsedStatePixmap
Specifies the pixmap to display on the PushButtonGadget to indicate that logical child items are folded (hidden) within an XmOUTLINE layout. The resource has no effect unless resource XmNoutlineButtonPolicy is XmOUTLINE_BUTTON_PRESENT. Otherwise, if the resource is unspecified, a default pixmap with an upwards pointing arrow is displayed.

XmNdetailColumnHeading
Specifies an XmString array to use as column headings in an XmDETAIL layout. No column headings are displayed if the value is NULL.

XmNdetailColumnHeadingCount
Specifies the length of the array associated with the XmNdetailColumnHeading resource.

XmNdetailOrder
Specifies an array of Cardinal values that represents which column, and in which order, the detail data associated with container items is to be displayed. The resource has no effect unless XmNlayoutType is XmDETAIL. If NULL, the XmNdetailOrderCount resource is used to determine the column detail data associated with each item.

XmNdetailOrderCount
Specifies the length of the array associated with the XmNdetailOrder resource. If XmNdetailOrder is NULL, and XmNdetailOrderCount is not zero, each container item displays any detail information in order starting from column 1, up to the value of XmNdetailOrderCount. Otherwise, with a value of zero, a default algorithm inspects the XmQTcontainerItem trait of each item to determine the columnar data.

XmNdetailTabList
Specifies an XmTabList which indicates the start of each column in an XmDETAIL layout. If NULL, the Container calculates a default XmTabList.

XmNentryViewType
Specifies the view type for all Container children. If the value is XmANY_ICON, then the XmQTcontainerItem trait of each child specifies the individual view type. Possible values:

```
XmANY_ICON      /* children use their own view type    */
XmLARGE_ICON    /* all children forced to XmLARGE_ICON */
XmSMALL_ICON    /* all children forced to XmSMALL_ICON */
```

XmNexpandedStatePixmap
Specifies the pixmap to display on the PushButtonGadget to indicate that logical child items are unfolded (displayed). The resource has no effect unless XmNoutlineButtonPolicy is XmOUTLINE_BUTTON_PRESENT. Otherwise, if the resource is unspecified, a default pixmap with a downwards pointing arrow is displayed.

XmNfontList
The XmFontList is considered obsolete in Motif 2.0 and later, and has been subsumed into the XmRenderTable. Any specified XmNrenderTable resource takes precedence.

XmNlargeCellHeight
Specifies the height of a cell when the Container is using a grid layout. The resource is not used when XmNentryViewType is XmSMALL_ICON.

XmNlargeCellWidth
Specifies the width of a cell when the Container is using a grid layout. The resource is not used when XmNentryViewType is XmSMALL_ICON.

XmNlayoutType
Specifies the way in which the Container lays out children. Possible values:

```
XmOUTLINE   /* items are displayed in a tree arrangement          */
XmSPATIAL   /* items displayed according to XmNspatialStyle resource */
XmDETAIL    /* items displayed in tabular row/column format       */
```

XmNmarginHeight
Specifies the spacing at the top and bottom of the Container widget.

XmNmarginWidth
Specifies the spacing at the left and right of the Container widget.

XmNoutlineButtonPolicy
Specifies whether a PushButtonGadget, used for folding/unfolding items, is displayed with each container item that has logical children, specified by the XmNentryParent resource. The resource has no effect if XmNspatialStyle is not XmOUTLINE. Possible values:

```
XmOUTLINE_BUTTON_ABSENT     /* display fold/unfold buttons */
XmOUTLINE_BUTTON_PRESENT    /* no PushButtonGadget buttons */
```

XmNoutlineColumnWidth
Specifies the width of the first column within an XmDETAIL layout, and the preferred width of the Container within an XmOUTLINE layout. If zero, the Container will deduce a default value based upon the width of the widest item pixmap and the XmNoutlineIndentation resource.

XmNoutlineIndentation
Specifies an indentation for container items. The resource has no effect when XmNlayoutType[1] is XmSPATIAL.

XmNoutlineLineStyle
Specifies whether to draw connecting lines between container items in an XmOUTLINE or XmDETAIL layout. Possible values:

```
XmNO_LINE                   /* no line is drawn between items      */
XmSINGLE                    /* a line one pixel wide connects items
*/
```

XmNprimaryOwnership
Specifies whether the Container takes possession of the primary selection when the user makes a selection from the items within the widget. Possible values:

```
XmOWN_NEVER                 /* never own the primary selection
*/
XmOWN_ALWAYS                /* always own the primary selection
*/
XmOWN_MULTIPLE              /* own if more than one item is selected
*/
XmOWN_POSSIBLE_MULTIPLE     /* own if multiple selection possible
*/
```

XmNrenderTable
Specifies the render table that is used for all children of the Container. If NULL, the nearest ancestor holding the XmQTspecifyRenderTable trait is searched, using the XmLABEL_FONTLIST value.

XmNselectColor
Specifies a color which container item children can use to indicate selected state. In addition to allocated Pixel values, the constant XmDEFAULT_SELECT_COLOR specifies a color between the XmNbackground and XmNbottomShadowColor, XmHIGHLIGHT_COLOR makes the select color the same as the XmNhighlightColor value, and XmREVERSED_GROUND_COLORS makes the XmNselectColor the same as the XmNforeground, using the XmNbackground color to render any text.

XmNselectedObjectCount
Specifies the number of widgets in the array of selected container items, represented by the XmNselectedObjects resource.

1.Erroneously listed as XmNlayoutStyle in 2nd Edition.

XmNselectedObjects
Specifies an array of widgets representing the set of container items currently selected.

XmNselectionPolicy
Specifies the way in which container items can be selected. Possible values:

```
XmSINGLE_SELECT          /* only one selected item permitted
*/
XmBROWSE_SELECT          /* as above, except items are selected by
dragging */
XmMULTIPLE_SELECT        /* items in contiguous range are selecta-
ble      */
XmEXTENDED_SELECT        /* items in discontinuous range are selecta-
ble    */
```

XmNselectionTechnique
Specifies the way in which items are selected. Possible values:

```
XmMARQUEE                /* items must be wholly enclosed
within Marquee          */ XmMARQUEE_EXTEND_START/* includes item
containing Marquee start coordinate      */ XmMARQUEE_EXTEND_BOTH/*
includes items containing Marquee start/end coordinates */ XmTOUCH_ONLY/*
select items between start and end location          */
XmTOUCH_OVER             /* select only items the mouse passes
through          */
```

XmNsmallCellHeight
Specifies the height of a cell when the Container spatial style is XmGRID, and the XmNentryViewType resource is XmSMALL_ICON.

XmNsmallCellWidth
Specifies the width of a cell when the Container spatial style is XmGRID, and the XmNentryViewType resource is XmSMALL_ICON.

XmNspatialIncludeModel
Specifies the layout of an item within a grid XmSPATIAL layout type, when the item is managed. Possible values:

```
XmAPPEND          /* place after the last occupied cell     */
                  /* according to XmNlayoutDirection */
XmCLOSEST         /* place in the free cell */
                  /* nearest the x, y coordinates of the item */
XmFIRST_FIT       /* place the item in the first free cell   */
                  /* according to XmNlayoutDirection */
```

XmNspatialResizeModel
Specifies how the Container will attempt to resize itself when there is insufficient space to contain a new item. The resource only has effect within a grid XmSPATIAL layout type. The definition of XmGROW_MAJOR and XmGROW_MINOR depend upon the value of the XmNlayoutDirection resource. The major dimension is width when the XmNlayoutDirection is horizontally oriented, and height when the direction is vertically oriented. Similarly, the minor dimension is height when the XmNlayoutDirection is horizontally oriented, and width when the direction is vertically oriented. Possible values:

```
XmGROW_BALANCED     /* request both width and height from parent */
XmGROW_MAJOR        /* request growth in the major dimension     */
XmGROW_MINOR        /* request growth in the minor dimension     */
```

XmNspatialSnapModel
Specifies how the Container will position an item within a cell, when the XmNlayoutType is XmSPATIAL. A value of XmSNAP_TO_GRID positions the item at the upper left or upper right of the cell, depending upon the value of XmNlayoutDirection. A value of XmNONE positions the item depending upon the value of XmNx, XmNy: if these fall outside the cell, then layout is performed according to the XmSNAP_TO_GRID method. A value of XmCENTER centers the item.

XmNspatialStyle
Specifies the layout of container items, when the XmNlayoutType is XmSPATIAL. Possible values:

```
XmCELLS     /* grid layout of same-sized cells. an item can occupy many cells
*/
XmGRID      /* grid layout of same-sized cells. an item can occupy one cell
*/
XmNONE      /* lay out according to XmNx, XmNy resources                 */
```

New Constraint Resources

Container defines the following constraint resources for its children:

Name	Class	Type	Default	Access
XmNentryParent	XmCEntryParent	Widget	NULL	CSG
XmNoutlineState	XmCOutlineState	unsigned char	XmCOLLAPSED	CSG
XmNpositionIndex	XmCPositionIndex	int	XmLAST_POSITION	CSG

XmNentryParent
Specifies a logical parent for the item. The root of a hierarchy has the value NULL. Used when the XmNlayoutType is XmOUTLINE or XmDETAIL.

XmNoutlineState
Specifies whether to display logical child items when the XmNlayoutType is XmOUTLINE or XmDETAIL. Possible values:

```
XmCOLLAPSED     /* does not display child items */
XmEXPANDED      /* displays child items         */
```

XmNpositionIndex
Specifies the order of the item within the Container, when XmNlayoutType is XmOUTLINE or XmDETAIL. Items are firstly ordered by XmNentryParent, and by XmNpositionIndex within those items sharing the same XmNentryParent. If unspecified, the highest such index for all other items sharing the same logical parent is calculated, and then incremented. If no other item shares the same logical parent, the default is zero.

Callback Resources

Container defines the following callback resources:

Callback	Reason Constant
XmNconvertCallback	XmCR_OK
XmNdefaultActionCallback	XmCR_DEFAULT_ACTION
XmNdestinationCallback	XmCR_OK
XmNoutlineChangedCallback	XmCR_COLLAPSED XmCR_EXPANDED
XmNselectionCallback	XmCR_SINGLE_SELECT XmCR_BROWSE_SELECT XmCR_MULTIPLE_SELECT XmCR_EXTENDED_SELECT

XmNconvertCallback
List of callbacks called when a request is made to convert a selection.

XmNdefaultActionCallback
List of callbacks called when an item is double-clicked, or KActivate is pressed.

XmNdestinationCallback
List of callbacks called when the Container is the destination of a transfer.

XmNoutlineChangedCallback
List of callbacks called when a change is made to the XmNoutlineState value of an item.

XmNselectionCallback
List of callbacks called when an item is selected.

Callback Structure

Each callback on the XmNoutlineChangedCallback list is passed the following structure:

```
typedef struct
    int             reason;          /* the reason that the callback was called */
    XEvent          *event;          /* points to event that triggered callback */
    Widget          item;            /* container item associated with event    */
    unsigned char   new_outline_state;       /* the requested state */
} XmContainerOutlineCallbackStruct;
```

new_outline_state specifies an XmNoutlineState for item. The value may be changed within the callback to force a particular state.

Each callback on the XmNselectionCallback and XmNdefaultActionCallback list is passed the following structure:

```
typedef struct {
    int             reason;          /* the reason that the callback was called */
    XEvent          *event;          /* points to event that triggered callback */
    WidgetList      selected_items;          /* the list of selected items          */
    int             selected_item_count;     /* the number of selected items     */
    unsigned char       auto_selection_type;         /* type of selection event    */
} XmContainerSelectCallbackStruct;
```

selected_items is the array of container items selected by the event. The number of such items is specified by selected_item_count. auto_selection_type indicates the type of automatic selection event. If XmNautomaticSelection is False, auto_selection_type has the value XmAUTO_UNSET. Otherwise, the range of possible values is given by:

```
XmAUTO_BEGIN        /* event is the beginning of automatic selection    */
XmAUTO_CANCEL       /* current selection is cancelled                   */
XmAUTO_CHANGE       /* current selection differs from initial selection */
XmAUTO_MOTION       /* current selection is caused by button drag */
XmAUTO_NO_CHANGE    /* current selection same as the initial selection */
```

Convert callbacks are fully described within the sections covering the Uniform Transfer Model. See *XmTransfer*(1) for more details. For quick reference, a pointer to the following structure is passed to callbacks on the XmNconvertCallback list:

```
typedef struct {
    int             reason;          /* reason that the callback is invoked     */
    XEvent          *event;          /* points to event that triggered callback */
    Atom            selection;       /* requested conversion selection          */
```

```
    Atom          target;          /* the conversion target                */
    XtPointer     source_data;     /* selection source information         */
    XtPointer     location_data;   /* information on data to be transferred */
    int           flags;           /* input status of the conversion       */
    XtPointer     parm;            /* parameter data for the target        *
    int           parm_format;     /* format of parameter data             */
    unsigned long     parm_length; /* the number of elements               */
                                   /*       in parameter data              */
    Atom          parm_type;       /* the type of the parameter data       */
    int           status;          /* output status of the conversion      */
    XtPointer     value;           /* returned conversion data             */
    Atom          type;            /* type of conversion data returned     */
    int           format;          /* format of the conversion data        */
    unsigned long     length;      /* number of elements in conversion data */
} XmConvertCallbackStruct;
```

Destination callbacks are fully described within the sections covering the Uniform Transfer Model. For quick reference, a pointer to the following structure is passed to callbacks on the XmNdestinationCallback list:

```
typedef struct {
    int           reason;              /* reason that the callback is invoked    */
    XEvent        *event;              /* points to event that triggered callback */
    Atom          selection;           /* requested selection type, as an Atom   */
    XtEnum        operation;           /* the type of transfer requested         */
    int           flags;               /* whether destination and source are same */
    XtPointer     transfer_id;         /* unique identifier for the request      */
    XtPointer     destination_data;    /* information about the destination      */
    XtPointer     location_data;       /* information about the data             */
    Time          time;                /* time when transfer operation started   */
} XmDestinationCallbackStruct;
```

Inherited Resources

Container inherits the resources shown below. The resources are listed alphabetically, along with the superclass that defines them.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite

Resource	Inherited From	Resource	Inherited From
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Widget Hierarchy

The PushButtonGadget children created by an outline style Container are all named OutlineButton.

Translations

The translations for Container include those of XmManager.

Event	Action
BSelect Press	ContainerHandleBtn1Down(ContainerBeginSelect, Copy)
BToggle Press	ContainerHandleBtn1Down(ContainerBeginToggle, Copy)
MLink BSelect Press	ContainerHandleBtn1Down(ContainerNoop, Link)
BExtend Press	ContainerHandleBtn1Down(ContainerBeginExtend Move)
BExtend Motion	ContainerHandleBtn1Motion(ContainerButtonMotion)
BSelect Release	ContainerHandleBtn1Up(ContainerEndSelect)
BToggle Release	ContainerHandleBtn1Up(ContainerEndToggle)
BExtend Release	ContainerHandleBtn1Up(ContainerEndExtend)
BTransfer Press	ContainerHandleBtn2Down(ContainerStartTransfer, Copy)
MLink BTransfer Press	ContainerHandleBtn2Down(ContainerStartTransfer, Link)
MMove BTransfer Press	ContainerHandleBtn2Down(ContainerStartTransfer, Move)
BTransfer Motion	ContainerHandleBtn2Motion(ContainerButtonMotion)

Event	Action
BTransfer Release	ContainerHandleBtn2Up(ContainerEndTransfer)
MShift KPrimaryCopy	ContainerPrimaryLink()
KPrimaryCopy	ContainerPrimaryCopy()
KPrimaryCut	ContainerPrimaryMove()
KCancel	ContainerCancel()
KExtend	ContainerExtend()
KSelect	ContainerSelect()
KSelectAll	ContainerSelectAll
KDeselectAll	ContainerDeselectAll()
KAddMode	ContainerToggleMode()
KActivate	ContainerActivate()
MShift KBeginData	ContainerExtendCursor(First)
MShift KEndData	ContainerExtendCursor(Last)
KBeginData	ContainerMoveCursor(First)
KEndData	ContainerMoveCursor(Last)
MCtrl KLeft	ContainerExpandOrCollapse(Left)
MCtrl KRight	ContainerExpandOrCollapse(Right)
MShift KUp	ContainerExtendCursor(Up)
MShift KDown	ContainerExtendCursor(Down)
MShift KLeft	ContainerExtendCursor(Left)
MShift KRight	ContainerExtendCursor(Right)
KUp	ContainerMoveCursor(Up)
KDown	ContainerMoveCursor(Down)
KLeft	ContainerMoveCursor(Left)
KRight	ContainerMoveCursor(Right)

Action Routines

Container defines the following action routines:

ContainerActivate()

Calls the procedures associated with the XmNdefaultActionCallback resource.

ContainerBeginExtend()

The action has no effect if the XmNlayoutType is XmSPATIAL, or if the XmNselectionPolicy is either XmSINGLE_SELECT or XmBROWSE_SELECT.

Otherwise, the location cursor is set to the object under the pointer, and if no object is there, or if there is no anchor, the action returns. Any items between the anchor and the location cursor are selected. Finally, if automatic selection is enabled, the list of callbacks specified by the XmNselectionCallback resource is invoked.

ContainerBeginSelect()

Single selection: if the object under the pointer is the anchor item, the selected state of the object is reversed. Otherwise, all items are deselected, the object at the pointer is made the anchor item, and the location cursor is set to it.

Browse selection: if the object under the pointer is not the anchor item, all items are deselected, the object is made the anchor item and selected, and the location cursor is set to it. If automatic selection is enabled, the list of callbacks specified by the XmNselectionCallback resource is invoked.

Multiple selection: sets the anchor item to the object under the pointer, and sets the location cursor to it. The selected state of the item is reversed. If the selection technique is XmTOUCH_OVER, and the anchor item is NULL, the Marquee start point is initialized. If automatic selection is enabled, the list of callbacks specified by the XmNselectionCallback resource is invoked.

Extended selection: as for multiple selection, except initially all items are deselected.

ContainerBeginToggle()

The action has no effect if the XmNlayoutType is XmSPATIAL, or if the XmNselectionPolicy is either XmSINGLE_SELECT or XmBROWSE_SELECT.

Multiple or Extended selection: the anchor item is set to the object under the pointer, and the location cursor is set to it. The selected state of the item is reversed. If automatic selection is enabled, the list of callbacks specified by the XmNselectionCallback resource is invoked. Lastly, if XmNselectionTechnique is XmMARQUEE_EXTEND_START or XmMARQUEE_EXTEND_BOTH, the Marquee rectangle is drawn around the item.

ContainerButtonMotion()

The action has no effect if XmNselectionPolicy is XmSINGLE_SELECT.

Browse selection: if the action follows ContainerBeginExtend() *or ContainerBeginToggle()* action, or if the pointer is over the current anchor item, the routine simply returns. Otherwise, the selected state of the anchor item is reversed, the selected state of any item under the pointer is also reversed, and the anchor item is reset to point to it.

Multiple and extended selection: if a previous action has initiated the Marquee, the rectangle is redrawn around the start point and the current pointer location. The selected state of all items within the Marquee are set to that of the anchor item. For non-Marquee selection, in a spatial layout the selected state of the item under the pointer is reversed, and the anchor item is reset to it. In a non-spatial layout, the selected state of all items between the anchor item and the item under the pointer are set to match the selected state of the anchor item.

In all cases, if automatic selection is enabled, the list of callbacks specified by the XmNselectionCallback resource is invoke*d.*

ContainerCancel()

The selected state of all items reverts to the pre-selection state. If automatic selection is enabled, the list of callbacks specified by the XmNselectionCallback resource is invoked.

ContainerDeselectAll()

All items are deselected, and the callbacks on the XmNselectionCallback list are invoked.

ContainerEndExtend()

The action has no effect if XmNlayoutType is XmSPATIAL.

Multiple or Extended selection: the callbacks specified by XmNselectionCallback are invoked.

ContainerEndSelect()

Single selection: simply invokes the callbacks specified by the XmNselectionCallback *resource.*

Browse selection: if the pointer is not over the current anchor item, the selected state of the current anchor, and the selected state of any item under the pointer are reversed. Callbacks specified by XmNselectionCallback are invoked.

Multiple and extended selection: similar to the ContainerButtonMotion() action, except that the auto_selection_type element within XmNselectionCallback procedures is XmAUTO_CHANGE or XmAUTO_NO_CHANGE rather than XmAUTO_MOTION.

ContainerEndToggle()

The action has no effect if XmNselectionPolicy is XmSINGLE_SELECT or XmBROWSE_SELECT.

Multiple or extended selection: the procedure directly invokes the ContainerEndSelect() action.

ContainerEndTransfer()

If the current transfer operation is XmLINK, the ContainerPrimaryLink() action is called. If the transfer operation is XmMOVE, the procedure invokes ContainerPrimaryMove(). If the operation is XmCOPY, the ContainerPrimaryCopy() action is called.

ContainerExpandOrCollapse(type)

The action has no effect if layout type is XmSPATIAL().

Otherwise, the outline state of the container item which has the focus is changed. Possible values for type:

```
Collapse/* outline state set to XmCOLLAPSED */
Expand/* outline state set to XmEXPANDED */
Left/* depends upon layout direction    */
Right/* depends upon layout direction    */
```

If XmNlayoutDirection is XmLEFT_TO_RIGHT, Left is interpreted as XmCOLLAPSED, and Right as XmEXPANDED. This is reversed if the layout direction is XmRIGHT_TO_LEFT.

ContainerExtend()

The action has no effect if layout type is XmSPATIAL(), or if XmNselectionPolicy is XmSINGLE_SELECT or XmBROWSE_SELECT.

Multiple selection: the selected state of all items between the anchor item and the location cursor is set to that of the anchor item. The callbacks specified by XmNselectionCallback *are invoked.*

Extended selection: in Normal mode, all items are deselected, then any items between the anchor item and the location cursor are selected. In Add mode, the selected state of all items between the anchor item and the location cursor is set to that of the anchor item. The callbacks specified by XmNselectionCallback are invoked.

ContainerExtendCursor(type)

The action has no effect if layout type is XmSPATIAL(), or if XmNselectionPolicy is XmSINGLE_SELECT or XmBROWSE_SELECT.

The location cursor is moved. If type is Left, Right, Up, Down, First, or Last, the cursor is moved one item in the specified direction if possible (or to the first/last item).
Thereafter, the ContainerExtend() procedure is directly invoked.

ContainerHandleBtn1Down(string)

The XmDisplay resource XmNenableBtn1Transfer configures the integration of selection and transfer operations on Button 1.

If XmNenableBtn1Transfer is not XmOFF, and the pointer is over an unselected item, the actions ContainerBeginSelect() and ContainerEndSelect() are invoked in order to select the item. If thereafter there is no selected item, the Marquee start point is initialized, otherwise the action becomes a data transfer operation, and the ContainerStartTransfer() action is invoked.

If XmNenableBtn1Transfer is XmOFF, and if no data transfer has been initialized, the action specified by string is invoked to initiate selection. Possible values for string:

ContainerBeginSelect,Copy
ContainerBeginToggle,Copy
ContainerNoop,Link
ContainerBeginExtend,Move

ContainerHandleBtn1Motion(string)

If the XmDisplay XmNenableBtn1Transfer resource is not XmOFF, and a selection is in progress, a drag action is initiated. Otherwise, the action as specified by string is invoked, typically ContainerButtonMotion.

ContainerHandleBtn1Up(string)

If a Button1 transfer is in progress, the transfer is cancelled. Otherwise, the action as specified by string is invoked. Possible values for string:

ContainerEndSelect
ContainerEndToggle
ContainerEndExtend

ContainerHandleBtn2Down(string)

If the XmDisplay XmNenableBtn1Transfer resource is XmBUTTON2_ADJUST, the action ContainerBeginExtend is directly invoked. Otherwise, the action as specified by string is invoked. Possible values for string:

ContainerStartTransfer, Copy
ContainerStartTransfer, Link
ContainerStartTransfer, Move

ContainerHandleBtn2Motion(string)

If the XmDisplay XmNenableBtn1Transfer resource is not XmBUTTON2_ADJUST, and a selection is in progress, a drag action is initiated. Otherwise, the action as specified by string is invoked, typically ContainerButtonMotion.

ContainerHandleBtn2Up(string)

If the XmDisplay XmNenableBtn1Transfer resource is XmBUTTON2_ADJUST, the action directly invokes the ContainerEndExtend() action. Otherwise, the action as specified by string is invoked, typically ContainerEndTransfer.

ContainerMoveCursor(string)

Moves the location cursor to the container item in a given direction, if possible. Valid values of type: Up, Down, Left, Right, First, Last.

If the number of selected items is greater than 1, all items are deselected. The item at the location cursor is selected, and callbacks associated with the XmNselectionCallback resource are invoked.

ContainerPrimaryCopy()

Requests a primary selection copy to the Container. Any XmNdestinationCallback procedures are invoked. By default, the Container performs no data transfer: the programmer must provide a callback for the task.

ContainerPrimaryLink()

Requests a primary selection link to the Container. Any XmNdestinationCallback procedures are invoked. By default, the Container performs no data transfer: the programmer must provide a callback for the task.

ContainerPrimaryMove()

Requests a primary selection copy to the Container. Any XmNdestinationCallback procedures are invoked. By default, the Container performs no data transfer: the programmer must provide a callback

for the task. Subsequently, the selection owner's XmNconvertCallback procedures are notified for the primary selection, with the target DELETE.

ContainerSelect()

Single or browse selection: deselects any selected item, and selects the item at the location cursor.

Multiple selection: reverses the selected state of the item at the location cursor, and makes this the anchor for any further operations.

Extended selection: in Normal mode, deselects all items, and selects the item at the location cursor. In Add mode, reverses the selected state of the item, which becomes the anchor for further operations.

In each case, callbacks associated with the XmNselectionCallback resource are invoked.

ContainerSelectAll()

Single or browse selection: deselects any selected item, and selects the item at the location cursor.

Multiple or extended selection: selects all container items.

In all cases, callbacks associated with the XmNselectionCallback resource are invoked.

ContainerStartTransfer(type)

The action saves the value of the parameter type for reference by later transfer operations. In XmSPATIAL layout, a DragContext is created, and a transfer operation is initiated. By default, unless overridden by a customized XmNconvertCallback procedure, if the drop occurs within the Container, then any dragged unselected item is moved to the pointer location, or if the item is selected, then all selected items are relocated to the pointer. Possible values for type are: Copy, Link, Move.

ContainerToggleMode()

Extended selection: toggles between Normal and Add mode.

Additional Behavior

Container has the following additional behavior:

<Double Click>

Calls the XmNdefaultActionCallback callbacks.

<FocusIn>
: If the keyboard focus policy is explicit, sets the focus and draws the location cursor.

<FocusOut>
: If the keyboard focus policy is explicit, removes the focus and erases the location cursor.

See Also

`XmContainerCopy`(1), `XmContainerCopyLink`(1), `XmContainerCut`(1), `XmContainerGetItemChildren`(1), `XmContainerPaste`(1), `XmContainerPasteLink`(1), `XmContainerRelayout`(1), `XmContainerReorder`(1), `XmCreateObject`(1), `XmTransfer`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmIconGadget`(2), `XmManager`(2).

Name

XmDialogShell widget class – the Shell parent for dialog boxes.

Synopsis

Public Header:
<Xm/DialogS.h>

Class Name:
XmDialogShell

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell →TransientShell →
XmDialogShell

Class Pointer:
xmDialogShellWidgetClass

Instantiation:
widget = `XmCreateDialogShell` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmDialogShellWidgetClass,...)

Functions/Macros:
`XmCreateDialogShell`(), `XmIsDialogShell`()

Description

DialogShell is the parent for dialog boxes. A DialogShell cannot be iconified separately, but only when the main application shell is iconified. The child of a DialogShell is typically a subclass of BulletinBoard and much of the functionality of DialogShell is based on this assumption.

Traits

DialogShell uses the XmQTdialogShellSavvy trait.

New Resources

DialogShell does not define any new resources.

Inherited Resources

DialogShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. DialogShell sets the default values of XmNdeleteResponse to XmUNMAP and XmNinput and XmNtransient to True. The default value of XmNborderWidth is reset to 0 by VendorShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNmaxAspectX	WMShell
XmNallowShellResize	Shell	XmNmaxAspectY	WMShell

Resource	Inherited From	Resource	Inherited From
XmNancestorSensitive	Core	XmNmaxHeight	WMShell
XmNaudibleWarning	VendorShell	XmNmaxWIdth	WMShell
XmNbackground	Core	XmNminAspectX	WMShell
XmNbackgroundPixmap	Core	XmNminAspectY	WMShell
XmNbaseHeight	WMShell	XmNminHeight	WMShell
XmNbaseWidth	WMShell	XmNminWidth	WMShell
XmNborderColor	Core	XmNmwmDecorations	VendorShell
XmNborderPixmap	Core	XmNmwmFunctions	VendorShell
XmNborderWidth	Core	XmNmwmInputMode	VendorShell
XmNbuttonFontList	VendorShell	XmNmwmMenu	VendorShell
XmNbuttonRenderTable	VendorShell	XmNnumChildren	Composite
XmNchildren	Composite	XmNoverrideRedirect	Shell
XmNcolormap	Core	XmNpopdownCalback	Shell
XmNcreatePopupChildProc	Shell	XmNpopupCallback	Shell
XmNdefaultFontList	VendorShell	XmNpreeditType	VendorShell
XmNdeleteResponse	VendorShell	XmNsaveUnder	Shell
XmNdepth	Core	XmNscreen	Core
XmNdestroyCallback	Core	XmNsensitive	Core
XmNgeometry	Shell	XmNshellUnitType	VendorShell
XmNheight	Core	XmNtextFontList	VendorShell
XmNheightInc	WMShell	XmNtextRenderTable	VendorShell
XmNiconMask	WMShell	XmNtitle	WMShell
XmNiconPixmap	WMShell	XmNtitleEncoding	WMShell
XmNiconWindow	WMShell	XmNtransient	WMShell
XmNinitialResourcesPersistent	Core	XmNtransientFor	TransientShell
XmNinitialState	WMShell	XmNtranslations	Core
XmNinput	WMShell	XmNvisual	Shell
XmNinputMethod	VendorShell	XmNwaitForWm	WMShell
XmNinputPolicy	VendorShell	XmNwidth	Core
XmNinsertPosition	Composite	XmNwidthInc	WMShell
XmNkeyboardFocusPolicy	VendorShell	XmNwindowGroup	WMShell
XmNlabelFontList	VendorShell	XmNwinGravity	WMShell
XmNlabelRenderTable	VendorShell	XmNwmTimeout	WMShell
XmNlayoutDirection	VendorShell	XmNx	Core
XmNmappedWhenManaged	Core	XmNy	Core

See Also

XmCreateObject(1), Composite(2), Core(2), Shell(2), TransientShell(2), VendorShell(2), WMShell(2), XmBulletinBoardDialog(2), XmErrorDialog(2), XmFileSelectionDialog(2), XmFormDialog(2), XmInformationDialog(2), XmMessageDialog(2), XmPromptDialog(2), XmQuestionDialog(2), XmSelectionDialog(2), XmTemplateDialog(2), XmWarningDialog(2), XmWorkingDialog(2).

Name

XmDisplay widget class – an object to store display-specific information.

Synopsis

Public Header:
<Xm/Display.h>

Class Name:
XmDisplay

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell →TopLevelShell →
ApplicationShell →XmDisplay

Class Pointer:
xmDisplayClass

Instantiation:
widget = `XtAppInitialize`(...)

Functions/Macros:
`XmGetXmDisplay`(), `XmIsDisplay`()

Availability

Motif 1.2 and later.

Description

The Display object stores display-specific information for use by the toolkit. An application has a Display object for each display it accesses. When an application creates its first shell on a display, typically by calling `XtAppInitialize`() or `XtAppCreateShell`(), a Display object is created automatically. There is no way to create a Display independently. The function `XmGetXmDisplay`() can be used to get the widget ID of the Display object.

The XmNdragInitiatorProtocolStyle and XmNdragReceiverProtocolStyle resources specify the drag protocol for an application that performs drag and drop operations. The two protocol styles are Dynamic and Preregister. Under the dynamic protocol, the initiator and receiver pass messages back and forth to handle drag and drop visuals. Under the Preregister protocol, the initiator handles drag and drop visuals by reading information that is preregistered and stored in properties. The actual protocol that is used by a specific initiator and receiver is based on the requested protocol styles of the receiver and initiator:

Drag Initiator	**Drag Receiver Protocol Style**			
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

New Resources

Display defines the following resources:

Name	Class	Type	Default	Access
XmNdefaultButtonEmphasis	XmCDefaultButtonEmphasis	XtEnum	XmEXTERNAL_HIGHLIGHT	SG
XmNdefaultVirtualBindings	XmCDefaultVirtualBindings	String	dynamic	SG
XmNdragInitiatorProtocolStyle	XmCDragInitiatorProtocolStyle	unsigned char	XmDRAG_PREFER_RECEIVER	SG
XmNdragReceiverProtocolStyle	XmCDragReceiverProtocolStyle	unsigned char	XmDRAG_PREFER_PREREGISTER	SG
XmNenableBtn1Transfer	XmCEnableBtn1Transfer	XtEnum	XmOFF	CG
XmNenableButtonTab	XmCEnableButtonTab	Boolean	False	CG
XmNenableDragIcon	XmCEnableDragIcon	Boolean	False	CG
XmNenableEtchedInMenu	XmCEnableEtchedinMenu	Boolean	False	CG
XmNenableMultiKeyBindings	XmCEnableMultiKeyBindings	Boolean	False	CG
XmNenableThinThickness	XmCEnableThinThickness	Boolean	False	CG
XmNenableToggleColor	XmCEnableToggleColor	Boolean	False	CG
XmNenableToggleVisual	XmCEnableToggleVisual	Boolean	False	CG
XmNenableUnselectableDrag	XmCEnableUnselectableDrag	Boolean	True	CG
XmNenableWarp	XmCEnableWarp	XtEnum	True	CG
XmNmotifVersion	XmCMotifVersion	int	XmVersion	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

XmNdefaultButtonEmphasis

In Motif 2.0 and later, specifies the manner in which button widgets and gadgets which have the XmNshowAsDefault resource set are displayed. A button which is the default has a double border. If XmNdefaultButtonEmphasis is XmINTERNAL_HIGHLIGHT, the location cursor is drawn between the double border. Otherwise, with a value of XmEXTERNAL_HIGHLIGHT, the location

cursor is drawn outside of the double border. An internal indication uses less space for the button.

XmNdefaultVirtualBindings
In Motif 2.0 and later, specifies the default virtual bindings for the display.

XmNdragInitiatorProtocolStyle
The client's drag and drop protocol requirements or preference when it is the initiator of a drag and drop operation. Possible values:

```
XmDRAG_PREREGISTER          /* can only use the preregister protocol */
XmDRAG_DYNAMIC              /* can only use the dynamic protocol     */
XmDRAG_NONE                 /* drag and drop is disabled             */
XmDRAG_DROP_ONLY            /* only supports dragging                */
XmDRAG_PREFER_DYNAMIC       /* supports both but prefers dynamic     */
XmDRAG_PREFER_PREREGISTER
                            /* supports both but prefers preregister */
XmDRAG_PREFER_RECEIVER      /* supports both; prefers receiver's     */
                            /* protocol */
```

XmNdragReceiverProtocolStyle
The client's drag and drop protocol requirements or preference when it is the receiver. Possible values:

```
XmDRAG_PREREGISTER          /* can only use the preregister protocol */
XmDRAG_DYNAMIC              /* can only use the dynamic protocol     */
XmDRAG_NONE                 /* drag and drop is disabled             */
XmDRAG_DROP_ONLY            /* only supports dropping                */
XmDRAG_PREFER_DYNAMIC       /* supports both but prefers dynamic     */
XmDRAG_PREFER_PREREGISTER
                            /* supports both but prefers preregister */
```

XmNenableBtn1Transfer
In Motif 2.0 and later, configures selection and transfer actions for Button1. The Container, Text, TextField, and List actions are affected by this resource. Possible values:

```
XmOFF                   /* selection and transfer disabled for button 1 */
XmBUTTON2_TRANSFER      /* selection on button 1, transfer on button 2  */
XmBUTTON2_ADJUST        /* selection on button 1, adjust on button 2    */
```

XmNenableButtonTab
In Motif 2.0 and later, configures the action of the Tab key with respect to keyboard navigation. If True, KNextField and KPrevField will behave like an Arrow key, moving the focus between widgets within a Tab Group, until the boundary of a Tab group is reached, at which point a subsequent navigation will move the focus into the next or previous Tab group. If False, KNextField and KPrevField move the focus to the next or previous tab group respectively.

XmNenableDragIcon
: In Motif 2.0 and later, a set of alternative icons representing the drag and drop default cursors is available. A value of True specifies that the newer icons are the default.

XmNenableEtchedInMenu
: In Motif 2.0 and later, specifies the way in which buttons within menus are shadowed when the widget is activated. The value False results in an etched out appearance, True gives an etched in shadowing, which is consistent with the appearance of activated buttons outside of a menu system. The resource affects PushButton, ToggleButton, CascadeButton widgets and gadget counterparts.

XmNenableMultiKeyBindings
: In Motif 2.1, merges an additional set of translations into the resource database which are compatible with CDE cancel translations.

XmNenableThinThickness
: Introduced in Motif 1.2.5 to provide CDE style shadowing and highlighting, used originally only by the ScrollBar. In Motif 2.1, the number of widgets sensitive to the resource is considerably expanded. If True, the default shadow thickness is 1, otherwise the default is 2.

XmNenableToggleColor
: In Motif 2.0 and later, specifies how the default value of a toggle's XmNselectColor is determined. True means that the default is taken from the XmNhighlightColor value, False uses the XmNbackground. XmNenableToggleColor is ignored if an explicit XmNselectColor is supplied to the toggle widget or gadget. The resource only takes effect if the indicator type of the toggle is XmONE_OF_MANY or XmONE_OF_MANY_ROUND.

XmNenableToggleVisual
: In Motif 2.0 and later, controls the default appearance of toggles. If False, a toggle with the indicator type of XmONE_OF_MANY is drawn as a diamond, and a toggle with indicator type XmN_OF_MANY is drawn square. If True, a toggle within a radio box has the default indicator type XmONE_OF_MANY_ROUND, which is rendered as a circle. A toggle outside of a radio box has the default indicator type XmN_OF_MANY, which is rendered square, and a check mark is displayed when XmNindicatorOn is True.

XmNenableUnselectableDrag
: In Motif 2.0 and later, specifies whether it is possible to initiate a drag operation from Label, LabelGadget, or Scale widgets. The value True enables drag operations from the widgets.

XmNenableWarp
In Motif 2.0 and later, specifies if the application is permitted to warp the pointer away from the user. The value True enables warping.

XmNmotifVersion
In Motif 2.0 and later, specifies the current version of Motif.

XmNuserData
In Motif 2.0 and later, specifies a pointer to data that the application can attach to the XmDisplay object. The resource is unused internally.

Callback Resources

In Motif 2.0 and later, Display defines the following callback resources:

Callback	Reason Constant
XmNdragStartCallback	XmCR_DRAG_START
XmNnoFontCallback	XmCR_NO_FONT
XmNnoRenditionCallback	XmCR_NO_RENDITION

XmNdragStartCallback
List of callbacks that are called when the procedure `XmDragStart`() is invoked.

XmNnoFontCallback
List of callbacks that are called when an XmRendition object fails in an attempt to load a font. This may happen if the object is created with an XmNloadModel of XmLOAD_IMMEDIATE, and the font cannot be loaded there and then, or if the XmNloadModel is XmLOAD_DEFERRED and a later attempt is made to render a compound string using an unloadable font. A callback can be supplied to rectify the situation: it can find or specify an alternative font, and invoke the function `XmRenditionUpdate`() upon the rendition object.

XmNnoRenditionCallback
List of callbacks that are called when an attempt is made to render using a rendition tag which does not match any entry within a given render table. A callback can be supplied to rectify the problem: it can create a new rendition with the problematic tag, and augment the render table.

Callback Structure

Each XmNnoFontCallback or XmNnoRenditionCallback procedure is passed the following structure:

```
typedef struct {
    int         reason;     /* the reason that the callback was called   */
    XEvent      *event;     /* points to event that triggered callback   */
    XmRendition rendition;  /* the rendition with a missing font         */
```

```
    char            *font_name;     /* the font which is not loadable          */
    XmRenderTable   render_table;   /* the render table with a missing rendition */
    XmString        tag;            /* the tag of the missing rendition        */
} XmDisplayCallbackStruct;
```

The render_table and tag elements are only applicable to callbacks on the XmNnoRenditionCallback list. rendition and font_name are valid only for XmNnoFontCallback callbacks.

In addition, an XmNdragStartCallback procedure is passed the following structure:

```
typedef struct {
    int             reason;     /* the reason that the callback was called     */
    XEvent          *event;     /* points to event structure that triggered callback */
    Widget          widget;     /* the ID of the widget where the drag initiated */
    Boolean         doit;       /* do the action (True) or undo it (False)     */
} XmDragStartCallbackStruct;
```

Inherited Resources

None of the resources inherited by Display can be set by the programmer or user.

See Also

`XmGetXmDisplay`(1), `ApplicationShell`(2), `Composite`(2), `Core`(2), `Shell`(2), `TopLevelShell`(2), `VendorShell`(2), `WMShell`(2), `XmScreen`(2).

Name

XmDragContext widget class – an object used to store information about a drag transaction.

Synopsis

Public Header:
<Xm/DragDrop.h>

Class Name:
XmDragContext

Class Pointer:
xmDragContextClass

Class Hierarchy:
Core →DragContext

Instantiation:
widget = `XmDragStart`(...)

Functions/Macros:
`XmDragCancel`(), `XmDragStart`()

Availability

Motif 1.2 and later.

Description

The DragContext object stores information that the toolkit needs to process a drag transaction. An application does not explicitly create a DragContext widget, but instead initiates a drag and drop operation by calling `XmDragStart`(), which initializes and returns a DragContext widget. The DragContext stores information about the types of data and operations of the drag source, the drag icons that are used during the drag, and the callbacks that are called during different parts of the drag. These characteristics can be specified as resources when the DragContext is created using `XmDragStart`().

Each drag operation has a unique DragContext that is freed by the toolkit when the operation is complete. The initiating and receiving clients in a drag and drop operation both use the DragContext to keep track of the state of the operation. The drag-over visual effects that are used during a drag operation depend on the drag protocol that is being used. Under the preregister protocol, either a cursor or a pixmap can be used, since the server is grabbed. Under the dynamic protocol, the X cursor is used.

New Resources

DragContext defines the following resources:

Name	Class	Type	Default	Access
XmNblendModel	XmCBlendModel	unsigned char	XmBLEND_ALL	CG
XmNclientData	XmCClientData	XtPointer	NULL	CSG
XmNconvertProc	XmCConvertProc	XtConvertSelectionIncrProc	NULL	CSG
XmNcursorBackground	XmCCursorBackground	Pixel	dynamic	CSG
XmNcursorForeground	XmCCursorForeground	Pixel	dynamic	CSG
XmNdragOperations	XmCDragOperations	unsigned char	XmDROP_COPY | XmDROP_MOVE	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
XmNoperationCursorIcon	XmCOperationCursorIcon	Widget	dynamic	CSG
XmNsourceCursorIcon	XmCSourceCursorIcon	Widget	dynamic	CSG
XmNsourcePixmapIcon	XmCSourcePixmapIcon	Widget	dynamic	CSG
XmNstateCursorIcon	XmCStateCursorIcon	Pixel	dynamic	CSG
XmNvalidCursorForeground	XmCCursorForeground	Pixel	dynamic	CSG

XmNblendModel

The combination of DragIcons that are blended to produce a drag-over visual. Possible values:

```
XmBLEND_ALL             /* source, state, and operation */
XmBLEND_STATE_SOURCE    /* source and state             */
XmBLEND_JUST_SOURCE     /* source only                  */
XmBLEND_NONE            /* no drag-over visual          */
```

XmNclientData

The client data that is passed to the XmNconvertProc.

XmNconvertProc

A procedure of type XtConvertSelectionIncrProc that converts the data to the format(s) specified by the receiving client. The widget argument passed to this procedure is the DragContext widget and the selection atom is _MOTIF_DROP. If XmNincremental is False, the conversion procedure should process the conversion atomically and ignore the max_length, client_data, and request_-id argu-

ments. Allocate any data returned by XmNconvertProc using `XtMalloc`() and it will be freed automatically by the toolkit after the transfer.

XmNcursorBackground
The background color of the cursor.

XmNcursorForeground
The foreground color of the cursor when the state icon is not blended. The default value is the foreground color of the widget passed to XmDragStart().

XmNdragOperations
The valid operations for the drag. The value is a bit mask that is formed by combining one or more of these possible values:

```
XmDROP_COPY          /* copy operations are valid */
XmDROP_LINK          /* link operations are valid */
XmDROP_MOVE          /* move operations are valid */
XmDROP_NOOP          /* no operations are valid   */
```

For Text and TextField widgets, the default value is XmDROP_COPY | XmDROP_MOVE. For List widgets and Label and subclasses, the default is XmDROP_COPY.

XmNexportTargets
The list of target atoms that the source data can be converted to.

XmNincremental
If True, the initiator uses the Xt incremental selection transfer mechanism. If False (default), the initiator uses atomic transfer.

XmNinvalidCursorForeground
The foreground color of the cursor when the state is invalid. The default value is the value of the XmNcursorForeground resource.

XmNnoneCursorForeground
The foreground color of the cursor when the state is none. The default value is the value of the XmNcursorForeground resource.

XmNnumExportTargets
The number of atoms in the XmNexportTargets list.

XmNoperationCursorIcon
The drag icon used to show the type of drag operation being performed. If the value is NULL, the default Screen icons are used.

XmNsourceCursorIcon
The drag icon used to represent the source data under the dynamic protocol. If the value is NULL, the default Screen icon is used.

XmNsourcePixmapIcon
The drag icon used to represent the source data under the preregister protocol. If the value is NULL, XmNsourceCursorIcon is used.

XmNstateCursorIcon
The drag icon used to show the state of a drop site. If the value is NULL, the default Screen icons are used.

XmNvalidCursorForeground
The foreground color of the cursor when the state is valid. The default value is the value of the XmNcursorForeground resource.

Callback Resources

DragContext defines the following callback resources:

Callback	Reason Constant
XmNdragDropFinishCallback	XmCR_DRAG_DROP_FINISH
XmNdragMotionCallback	XmCR_DRAG_MOTION
XmNdropFinishCallback	XmCR_DROP_FINISH
XmNdropSiteEnterCallback	XmCR_DROP_SITE_ENTER
XmNdropSiteLeaveCallback	XmCR_DROP_SITE_LEAVE
XmNdropStartCallback	XmCR_DROP_START
XmNoperationChangedCallback	XmCR_OPERATION_CHANGED
XmNtopLevelEnterCallback	XmCR_TOP_LEVEL_ENTER
XmNtopLevelLeaveCallback	XmCR_TOP_LEVEL_LEAVE

XmNdragDropFinishCallback
List of callbacks that are called when the entire transaction is finished.

XmNdragMotionCallback
List of callbacks that are called when the pointer moves during a drag.

XmNdropFinishCallback
List of callbacks that are called when the drop is finished.

XmNdropSiteEnterCallback
List of callbacks that are called when the pointer enters a drop site.

XmNdropSiteLeaveCallback
List of callbacks that are called when the pointer leaves a drop site.

XmNdropStartCallback
List of callbacks that are called when a drop is started.

XmNoperationChangedCallback
List of callbacks that are called when the user changes the operation during a drag.

XmNtopLevelEnterCallback
List of callbacks that are called when the pointer enters a top-level window or root window.

XmNtopLevelLeaveCallback
List of callbacks that are called when the pointer leaves a top-level window or the root window.

Callback Structure

The XmNdragDropFinishCallback is passed the following structure:

```
typedef struct {
    int          reason;          /* the reason the callback was called     */
    XEvent       *event;          /* event structure that triggered callback */
    Time         timeStamp;       /* time at which operation completed      */
} XmDragDropFinishCallbackStruct, *XmDragDropFinishCallback;
```

The XmNdragMotionCallback is passed the following structure:

```
typedef struct {
    int             reason;          /* reason the callback was called */
    XEvent          *event;          /* event that triggered callback */
    Time            timeStamp;       /* timestamp of logical event    */
    unsigned char   operation;       /* current operation             */
    unsigned char   operations;      /* supported operations          */
    unsigned char   dropSiteStatus;  /* valid, invalid, or none       */
    Position        x;               /* x-coordinate of pointer       */
    Position        y;               /* y-coordinate of pointer       */
} XmDragMotionCallbackStruct, *XmDragMotionCallback;
```

The XmNdropFinishCallback is passed the following structure:

```
typedef struct {
    int             reason;          /* reason the callback was called   */
    XEvent          *event;          /* event that triggered callback    */
    Time            timeStamp;       /* time at which drop completed     */
    unsigned char   operation;       /* current operation                */
    unsigned char   operations;      /* supported operations             */
    unsigned char   dropSiteStatus;  /* valid, invalid, or none          */
    unsigned char   dropAction;      /* drop, cancel, help, or interrupt */
    unsigned char   completionStatus; /* success or failure              */
} XmDropFinishCallbackStruct, *XmDropFinishCallback;
```

The XmNdropSiteEnterCallback is passed the following structure:

```
typedef struct {
    int             reason;          /* reason the callback was called */
    XEvent          *event;          /* event that triggered callback */
    Time            timeStamp;       /* time of crossing event       */
    unsigned char   operation;       /* current operation           */
    unsigned char   operations;      /* supported operations         */
    unsigned char   dropSiteStatus;  /* valid, invalid, or none      */
    Position        x;               /* x-coordinate of pointer      */
    Position        y;               /* y-coordinate of pointer      */
} XmDropSiteEnterCallbackStruct, *XmDropSiteEnterCallback;
```

The XmNdropSiteLeaveCallback is passed the following structure:

```
typedef struct {
    int             reason;          /* reason the callback was called */
    XEvent          *event;          /* event that triggered callback */
    Time            timeStamp;       /* time of crossing event       */
} XmDropSiteLeaveCallbackStruct, *XmDropSiteLeaveCallback;
```

The XmNdropStartCallback is passed the following structure:

```
typedef struct {
    int             reason;          /* reason the callback was called  */
    XEvent          *event;          /* event that triggered callback   */
    Time            timeStamp;       /* time at which drag completed    */
    unsigned char   operation;       /* current operation           */
    unsigned char   operations;      /* supported operations         */
    unsigned char   dropSiteStatus;  /* valid, invalid, or none      */
    unsigned char   dropAction;      /* drop, cancel, help, or interrupt */
    Position        x;               /* x-coordinate of pointer      */
    Position        y;               /* y-coordinate of pointer      */
} XmDropStartCallbackStruct, *XmDropStartCallback;
```

The XmNoperationChangedCallback is passed the following structure:

```
typedef struct {
    int             reason;          /* reason the callback was called */
    XEvent          *event;          /* event that triggered callback */
    Time            timeStamp;       /* timestamp of logical event    */
    unsigned char   operation;       /* current operation           */
    unsigned char   operations;      /* supported operations         */
    unsigned char   dropSiteStatus;  /* valid, invalid, or none      */
} XmOperationChangedCallbackStruct, *XmOperationChangedCallback;
```

The XmNtopLevelEnterCallback is passed the following structure:

```
typedef struct {
    int             reason;             /* reason callback was called    */
    XEvent          *event;             /* event that triggered callback */
    Time            timestamp;          /* timestamp of logical event    */
    Screen          screen;             /* screen of top-level window    */
    Window          window;             /* window being entered          */
    Position        x;                  /* x-coordinate of pointer       */
    Position        y;                  /* y-coordinate of pointer       */
    unsigned char   dragProtocolStyle;  /* drag protocol of initiator    */
} XmTopLevelEnterCallbackStruct, *XmTopLevelEnterCallback;
```

The XmNtopLevelLeaveCallback is passed the following structure:

```
typedef struct {
    int             reason;             /* reason the callback was called */
    XEvent          *event;             /* event that triggered callback */
    Time            timestamp;          /* timestamp of logical event    */
    Screen          screen;             /* screen of top-level window    */
    Window          window;             /* window being left             */
} XmTopLevelLeaveCallbackStruct, *XmTopLevelLeaveCallback;
```

The operations field in these structures specifies the set of operations supported for the data being dragged. The toolkit initializes the value based on the operations field of the XmDragProcCallbackStruct, the XmNdropSiteOperations resource of the DropSite, the XmNdragOperations resource of the DragContext and the operation selected by the user. The operation field in these structures specifies the current operation. The toolkit initializes the value based on the value of the operation field of the XmDragProcCallbackStruct, operations, and the XmNdropSiteOperations resource of the Drop Site.

The dropSiteStatus field in these structures specifies whether or not the drop site is valid. The toolkit initializes the value based on the XmNimportTargets resource of the DropSite and the XmNexportTargets resource of the DragContext and the location of the pointer. The possible values are XmDROP_SITE_VALID, XmDROP_SITE_INVALID, and XmNO_DROP_SITE.

The dropAction field in these structures specifies the action associated with the drop. The possible values are XmDROP, XmDROP_CANCEL, XmDROP_INTERRUPT, and XmDROP_HELP[1]. XmDROP_INTERRUPT is unsupported and is interpreted as XmDROP_CANCEL.

The completionStatus field in the XmDropFinishCallbackStruct specifies the status of the drop transaction, which determines the drop visual effect. The value of this field can be changed by the XmNdropFinishCallbackStruct. The possible values are XmDROP_SUCCESS[2] and XmDROP_FAILURE[3].

Inherited Resources

DragContext inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. DragContext sets the default value of XmNborderWidth to 0.

Name	Inherited From	Name	Inherited From
XmNaccelerators	Core	XmNheight	Core
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNscreen	Core
XmNborderColor	Core	XmNsensitive	Core
XmNborderPixmap	Core	XmNtranslations	Core
XmNborderWidth	Core	XmNwidth	Core
XmNcolormap	Core	XmNx	Core
XmNdepth	Core	XmNy	Core
XmNdestroyCallback	Core		

1.Erroneously given as DROP_HELP in 1st and 2nd edition.

2.Erroneously given as XmSUCCESS in 1st and 2nd edition.

3.Erroneously given as XmFAILURE in 1st and 2nd edition.

Translations

Event	Action
BDrag Motion	DragMotion()
BDrag Release	FinishDrag()
KCancel	CancelDrag()
KHelp	HelpDrag()

Action Routines

DragContext defines the following action routines:

CancelDrag()
: Cancels the drag operation and frees the associated DragContext.

DragMotion()
: Drags the selected data as the pointer is moved.

FinishDrag()
: Completes the drag operation and initiates the drop operation.

HelpDrag()
: Starts a conditional drop that allows the receiving client to provide help information to the user. The user can cancel or continue the drop operation in response to this information.

See Also

`XmDragCancel`(1), `XmDragStart`(1), `XmGetDragContext`(1), `Core`(2), `XmDisplay`(2), `XmDragIcon`(2), `XmDropSite`(2), `XmDropTransfer`(2), `XmScreen`(2).

Name

XmDragIcon widget class – an object used to represent the data in a drag and drop operation.

Synopsis

Public Header:
<Xm/DragDrop.h>

Class Name:
XmDragIcon

Class Pointer:
xmDragIconObjectClass

Class Hierarchy:
Object →DragIcon

Instantiation:
widget = `XmCreateDragIcon`(...)

Functions/Macros:
`XmCreateDragIcon`(), `XmIsDragIconObjectClass`()

Availability

Motif 1.2 and later.

Description

A DragIcon is an object that represents the source data in a drag and drop transaction. During a drag operation, the cursor changes into a visual that is created by combining the various DragIcons specified in the DragContext associated with the operation. A DragIcon is created using the `XmCreateDragIcon`() function or from entries in the resource database.

A drag-over visual can have both a static and a dynamic part. The static part of the visual is the DragIcon that represents the source data. The dynamic parts can be DragIcons that change to indicate the type of operation that is being performed and whether the pointer is over a valid or an invalid drop site. The XmNblendModel resource of the DragContext for a drag and drop operation specifies which icons are blended to produce the drag-over visual. DragIcon resources specify the relative positions of the operation and state icons if they are used. When a DragIcon is not specified, the default DragIcons from the appropriate Screen object are used.

New Resources

DragIcon defines the following resources:

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
: The relative location on the source icon where the state or operation icon is attached. Possible values:

XmATTACH_NORTH_WEST	XmATTACH_NORTH
XmATTACH_NORTH_EAST	XmATTACH_EAST
XmATTACH_SOUTH_EAST	XmATTACH_SOUTH
XmATTACH_SOUTH_WEST	XmATTACH_WEST
XmATTACH_CENTER	XmATTACH_HOT

XmNdepth
: The depth of the pixmap.

XmNheight
: The height of the pixmap.

XmNhotX
: The x-coordinate of the hotspot of the cursor.

XmNhotY
: The y-coordinate of the hotspot of the cursor.

XmNmask
: The mask for the DragIcon pixmap.

XmNoffsetX
: The horizontal offset in pixels of the origin of the state or operation icon relative to the attachment point on the source icon.

XmNoffsetY
The vertical offset in pixels of the origin of the state or operation icon relative to the attachment point on the source icon.

XmNpixmap
The pixmap for the DragIcon.

XmNwidth
The width of the pixmap.

Inherited Resources

DragIcon inherits the following resource:

Resource	Inherited From
XmNdestroyCallback	Object

See Also

`XmCreateObject`(1), `Object`(2), `XmDisplay`(2), `XmDragContext`(2), `XmDropSite`(2), `XmDropTransfer`(2), `XmScreen`(2).

Name

XmDrawingArea widget class – a simple manager widget for interactive drawing.

Synopsis

Public Header:
<Xm/DrawingA.h>

Class Name:
XmDrawingArea

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmDrawingArea

Class Pointer:
xmDrawingAreaWidgetClass

Instantiation:
widget = `XmCreateDrawingArea` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmDrawingAreaWidgetClass,...)

Functions/Macros:
`XmCreateDrawingArea`(), `XmIsDrawingArea`()

Description

DrawingArea provides a blank canvas for interactive drawing. The widget does not do any drawing of its own. Since DrawingArea is a subclass of Manager, it can provide simple geometry management of multiple widget or gadget children. The widget does not define any behavior except for invoking callbacks that notify an application when it receives input events, exposure events, and resize events.

New Resources

DrawingArea defines the following resources:

Name	Class	Type	Default	Access
XmNmarginHeight	XmCMarginHeight	Dimension	10	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	10	CSG
XmNresizePolicy	XmCResizePolicy	unsigned char	XmRESIZE_ANY	CSG

XmNmarginHeight
The spacing between a DrawingArea's top or bottom edge and any child widget.

XmNmarginWidth
The spacing between a DrawingArea's right or left edge and any child widget.

XmNresizePolicy
How DrawingArea widgets are resized. Possible values:

```
XmRESIZE_NONE       /* remain at fixed size       */
XmRESIZE_GROW       /* expand only                */
XmRESIZE_ANY        /* shrink or expand, as needed */
```

Callback Resources

DrawingArea defines the following callback resources:

Callback	Reason Constant
XmNconvertCallback	XmCR_OK
XmNdestinationCallback	XmCR_OK
XmNexposeCallback	XmCR_EXPOSE
XmNinputCallback	XmCR_INPUT
XmNresizeCallback	XmCR_RESIZE

XmNconvertCallback
In Motif 2.0 and later, specifies the list of callbacks called when a request is made to convert a selection.

XmNdestinationCallback
In Motif 2.0 and later, specifies the list of callbacks called when the DrawingArea is the destination of a data transfer.

XmNexposeCallback
List of callbacks that are called when the DrawingArea receives an exposure event.

XmNinputCallback
List of callbacks that are called when the DrawingArea receives a keyboard or mouse event.

XmNresizeCallback
List of callbacks that are called when the DrawingArea receives a resize event.

Callback Structure

Each expose, resize, and input callback function is passed the following structure:

```
typedef struct {
    int         reason;     /* the reason that the callback was called */
    XEvent      *event;     /* event structure that triggered callback; */
                            /* for XmNresizeCallback, this is NULL     */
    Window      window;     /* the widget's window                     */
} XmDrawingAreaCallbackStruct;
```

Convert callbacks are fully described within the sections covering the Uniform Transfer Model. See `XmTransfer`(1) for more details. For quick reference, a pointer to the following structure is passed to callbacks on the XmNconvertCallback list:

```
typedef struct {
    int           reason;          /* reason that the callback is invoked     */
    XEvent        *event;          /* points to event that triggered callback */
    Atom          selection;       /* requested conversion selection          */
    Atom          target;          /* the conversion target                   */
    XtPointer     source_data;     /* selection source information            */
    XtPointer     location_data;   /* information about data to be transferred*/
    int           flags;           /* input status of the conversion          */
    XtPointer     parm;            /* parameter data for the target           */
    int           parm_format;     /* format of parameter data                */
    unsigned long parm_length;     /* number of elements in                   */
                                   /*       parameter data                    */
    Atom          parm_type;       /* the type of the parameter data          */
    int           status;          /* output status of the conversion         */
    XtPointer     value;           /* returned conversion data                */
    Atom          type;            /* type of conversion data returned        */
    int           format;          /* format of the conversion data           */
    unsigned long length;          /* number of elements in conversion data */
} XmConvertCallbackStruct;
```

Destination callbacks are fully described within the sections covering the Uniform Transfer Model. See `XmTransfer`(1) for more details. For quick reference, a pointer to the following structure is passed to callbacks on the XmNdestinationCallback list:

```
typedef struct {
    int           reason;          /* reason that the callback is invoked     */
    XEvent        *event;          /* points to event that triggered callback */
    Atom          selection;       /* the requested selection type, as an Atom */
    XtEnum        operation;       /* the type of transfer requested          */
    int           flags;           /* whether destination and source are same */
    XtPointer     transfer_id;     /* unique identifier for the request       */
    XtPointer     destination_data; /* information about the destination */
    XtPointer     location_data;   /* information about the data        */
    Time          time;            /* the time when transfer operation started */
} XmDestinationCallbackStruct;
```

Inherited Resources

DrawingArea inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of XmNborderWidth is reset to 0 by Manager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Translations

The translations for DrawingArea include those of Manager. All of the events in the inherited translations except <BtnMotion>, <EnterWindow>, <LeaveWindow>, <FocusIn>, and <FocusOut> call the DrawingAreaInput() action before calling the Manager actions.

DrawingArea has the following additional translations:

Event	Action
MAny Bany Press	DrawingAreaInput()
MAny Bany Release	DrawingAreaInput()
Many KAny Press	DrawingAreaInput() ManagerGadgetKeyInput()
MAny KAny Release	DrawingAreaInput()

Action Routines

DrawingArea defines the following action routines:

DrawingAreaInput()
: When a widget child of a DrawingArea receives a keyboard or mouse event, this action routine invokes the list of callbacks specified by XmNinputCallback.

ManagerGadgetKeyInput()
: When a gadget child of a DrawingArea receives a keyboard or mouse event, this action routine processes the event.

Additional Behavior

DrawingArea has the following additional behavior:

<Expose>
: Invokes the XmNexposeCallback callbacks.

<WidgetResize>
: Invokes the XmNresizeCallback callbacks.

See Also

`XmCreateObject`(1), `XmTransfer`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmManager`(2).

Name

XmDrawnButton widget class – a button widget that provides a graphics area.

Synopsis

Public Header:
<Xm/DrawnB.h>

Class Name:
XmDrawnButton

Class Hierarchy:
XmPrimitive →XmLabel →XmDrawnButton

Class Pointer:
xmDrawnButtonWidgetClass

Instantiation:
widget = `XmCreateDrawnButton` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmDrawnButtonWidgetClass,...)

Functions/Macros:
`XmCreateDrawnButton`(), `XmIsDrawnButton`()

Description

DrawnButton is an empty widget window, surrounded by a shaded border. The widget provides a graphics area that can act like a PushButton. The graphics can be dynamically updated by the application.

Traits

DrawnButton holds the XmQTactivatable trait, which is inherited by any derived classes, and uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

New Resources

DrawnButton defines the following resources:

Name	Class	Type	Default	Access
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	CSG
XmNpushButtonEnabled	XmCPushButtonEnabled	Boolean	False	CSG
XmNshadowType	XmCShadowType	unsigned char	XmSHADOW_ETCHED_IN	CSG

XmNmultiClick
A flag that determines whether successive button clicks are processed or ignored. Possible values:

```
XmMULTICLICK_DISCARD    /* ignore successive button clicks; */
                        /* default value in a menu system   */
```

```
XmMULTICLICK_KEEP       /* count successive button clicks; */
                        /* default value when not in a menu */
```

XmNpushButtonEnabled
If False (default), the shadow drawing doesn't appear three dimensional; if True, the shading provides a pushed in or raised appearance as for the PushButton widget.

XmNshadowType
The style in which shadows are drawn. Possible values:

```
XmSHADOW_IN             /* widget appears inset          */
XmSHADOW_OUT            /* widget appears outset         */
XmSHADOW_ETCHED_IN      /* double line; widget appears inset */
XmSHADOW_ETCHED_OUT     /* double line; widget appears raised */
```

Callback Resources

DrawnButton defines the following callback resources:

Callback	Reason Constant
XmNactivateCallback	XmCR_ACTIVATE
XmNarmCallback	XmCR_ARM
XmNdisarmCallback	XmCR_DISARM
XmNexposeCallback	XmCR_EXPOSE
XmNresizeCallback	XmCR_RESIZE

XmNactivateCallback
List of callbacks that are called when BSelect is pressed and released inside of the widget.

XmNarmCallback
List of callbacks that are called when BSelect is pressed while the pointer is inside the widget.

XmNdisarmCallback
List of callbacks that are called when BSelect is released after it has been pressed inside of the widget.

XmNexposeCallback
List of callbacks that are called when the widget receives an exposure event.

XmNresizeCallback
List of callbacks that are called when the widget receives a resize event.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;      /* the reason that the callback was called */
    XEvent      *event;      /* event structure that triggered callback */
    Window      window;      /* ID of window in which the event occurred */
    int         click_count; /* number of multi-clicks                   */
} XmDrawnButtonCallbackStruct;
```

event is NULL for XmNresizeCallback and is sometimes NULL for XmNactivateCallback.

click_count is meaningful only for XmNactivateCallback. Furthermore, if the XmNmultiClick resource has the value XmMULTICLICK_KEEP, then XmNactivateCallback is called for each click, and the value of *click_count* is the number of clicks that have occurred in the last sequence of multiple clicks. If the XmNmultiClick resource is set to XmMULTICLICK_DISCARD, then *click_count* always has a value of 1.

Inherited Resources

DrawnButton inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. DrawnButton sets default value of XmNlabelString to XmUNSPECIFIED[1]. The default value of XmNborderWidth is reset to 0 by Primitive. In Motif 2.0 and earlier, the default value of XmNhighlightThickness and XmNshadowThickness are reset to 2. In Motif 2.1 and later, the default values depend upon the XmDisplay XmNenableThinThickness resource: if True, the default is 1, otherwise 2.

Resource	Inherited From	Resource	Inherited From
XmNaccelerator	XmLabel	XmNlabelType	XmLabel
XmNaccelerators	Core	XmNlayoutDirection	XmPrimitive
XmNacceleratorText	XmLabel	XmNmappedWhenManaged	Core
XmNalignment	XmLabel	XmNmarginBottom	XmLabel
XmNancestorSensitive	Core	XmNmarginHeight	XmLabel
XmNbackground	Core	XmNmarginLeft	XmLabel
XmNbackgroundPixmap	Core	XmNmarginRight	XmLabel
XmNborderColor	Core	XmNmarginTop	XmLabel

1.Given as “” in 1st and 2nd editions. This is imprecise. The XmLabel superclass treats XmUNSPECIFIED as a special value, which maps to an empty XmString.

Resource	Inherited From	Resource	Inherited From
XmNborderPixmap	Core	XmNmarginWidth	XmLabel
XmNborderWidth	Core	XmNmnemonicCharSet	XmLabel
XmNbottomShadowColor	XmPrimitive	XmNmnemonic	XmLabel
XmNbottomShadowPixmap	XmPrimitive	XmNnavigationType	XmPrimitive
XmNcolormap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNrecomputeSize	XmLabel
XmNdepth	Core	XmNrenderTable	XmLabel
XmNdestroyCallback	Core	XmNscreen	Core
XmNfontList	XmLabel	XmNsensitive	Core
XmNforeground	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNheight	Core	XmNstringDirection	XmLabel
XmNhelpCallback	XmPrimitive	XmNtopShadowColor	XmPrimitive
XmNhighlightColor	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNhighlightOnEnter	XmPrimitive	XmNtranslations	Core
XmNhighlightPixmap	XmPrimitive	XmNtraversalOn	XmPrimitive
XmNhighlightThickness	XmPrimitive	XmNunitType	XmPrimitive
XmNinitialResourcesPersistent	Core	XmNuserData	XmPrimitive
XmNlabelInsensitivePixmap	XmLabel	XmNwidth	Core
XmNlabelPixmap	XmLabel	XmNx	Core
XmNlabelString	XmLabel	XmNy	Core

Translations

Event	Action
BSelect Press	Arm()
MCtrl BSelect Press	ButtonTakeFocus()
BSelect Click	Activate() Disarm()
BSelect Release	Activate() Disarm()
BSelect Press 2+	MultiArm()
BSelect Release 2+	MultiActivate()
KSelect	ArmAndActivate()
KHelp	Help()

Action Routines

DrawnButton defines the following action routines:

Activate()
: Displays the DrawnButton as unselected if XmNpushButtonEnabled is True or displays the shadow according to XmNshadowType. Invokes the list of callbacks specified by XmNactivateCallback.

Arm()
: Displays the DrawnButton as selected if XmNpushButtonEnabled is True or displays the shadow according to XmNshadowType. Invokes the list of callbacks specified by XmNarmCallback.

ArmAndActivate()
: Displays the DrawnButton as selected if XmNpushButtonEnabled is True or displays the shadow according to XmNshadowType. Invokes the list of callbacks specified by XmNarmCallback. After doing this, the action routine displays the DrawnButton as unselected if XmNpushButtonEnabled is True or displays the shadow according to XmNshadowType and invokes the list of callbacks specified by XmNactivateCallback and XmNdisarmCallback.

ButtonTakeFocus()
: In Motif 2.0 and later, moves the current keyboard focus to the DrawnButton, without activating the widget.

Disarm()
: Displays the DrawnButton as unselected and invokes the list of callbacks specified by XmNdisarmCallback.

Help()
: Invokes the list of callbacks specified by XmNhelpCallback. If the DrawnButton doesn't have any help callbacks, the Help() routine invokes those associated with the nearest ancestor that has them.

MultiActivate()
: Increments the click_count member of XmDrawnButtonCallbackStruct, displays the DrawnButton as unselected if XmNpushButtonEnabled is True or displays the shadow according to XmNshadowType, and invokes the list of callbacks specified by XmNactivateCallback and XmNdisarmCallback. This action routine takes effect only when the XmNmultiClick resource is set to XmMULTICLICK_KEEP.

MultiArm()
: Displays the DrawnButton as selected if XmNpushButtonEnabled is True or displays the shadow according to XmNshadowType, and invokes the list of callbacks specified by XmNarmCallback. This action routine takes effect only when the XmNmultiClick resource is set to XmMULTICLICK_KEEP.

Additional Behavior

DrawnButton has the following additional behavior:

<EnterWindow>
: Displays the DrawnButton as selected if XmNpushButtonEnabled is True and the pointer leaves and re-enters the window while BSelect is pressed.

<LeaveWindow>
: Displays the DrawnButton as unselected if XmNpushButtonEnabled is True and the pointer leaves the window while BSelect is pressed.

See Also

`XmCreateObject`(1), `Core`(2), `XmLabel`(2), `XmPrimitive`(2), `XmPushButton`(2).

Name

XmDropSite registry – an object that defines the characteristics of a drop site.

Synopsis

Public Header:
<Xm/DragDrop.h>

Class Hierarchy:
DropSite does not inherit from any widget class.

Instantiation:
`XmDropSiteRegister`(...)

Functions/Macros:
`XmDropSiteConfigureStackingOrder`(), `XmDropSiteEndUpdate`(), `XmDropSiteQueryStackingOrder`(), `XmDropSiteRegister`(), `XmDropSiteRetrieve`(), `XmDropSiteStartUpdate`(), `XmDropSiteUpdate`(), `XmDropSiteUnregister`()

Availability

Motif 1.2 and later.

Description

An XmDropSite is an object that stores data about a drop site for drag and drop operations. A DropSite is associated with a particular widget or gadget in an application. An application registers a widget or gadget as a DropSite using `XmDropSiteRegister`(). The DropSite stores information about the shape of the drop site, the animation effects used when the pointer enters the drop site, the types of data supported by the drop site, and the callback that is activated when a drop occurs. These characteristics can be specified as resources when the DropSite is created.

The functions `XmDropSiteUpdate`() and `XmDropSiteRetrieve`() set and get the drop site resources for a widget that is registered as a DropSite. Use these routines instead of `XtSetValues`() and `XtGetValues`().

New Resources

DropSite defines the following resources:

Name	Class	Type	Default	Access
XmNanimationMask	XmCAnimationMask	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNanimationPixmap	XmCAnimationPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNanimationPixmapDepth	XmCAnimationPixmapDepth	int	0	CSG
XmNanimationStyle	XmCAnimationStyle	unsigned char	XmDRAG_UNDER_HIGHLIGHT	CSG

Name	Class	Type	Default	Access
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
The mask for the XmNanimationPixmap when the animation style is XmDRAG_UNDER_PIXMAP.

XmNanimationPixmap
The pixmap used for drag-under animation when the animation style is XmDRAG_UNDER_PIXMAP.

XmNanimationPixmapDepth
The depth of the pixmap specified by XmNanimationPixmap.

XmNanimationStyle
The style of drag-under animation used when the pointer enters a valid drop site during a drag operation. Possible values:

```
XmDRAG_UNDER_HIGHLIGHT      /* drop site highlighted                */
XmDRAG_UNDER_SHADOW_OUT     /* drop site shown with outset shadow */
XmDRAG_UNDER_SHADOW_IN      /* drop site shown with inset shadow  */
XmDRAG_UNDER_PIXMAP         /* drop site displays pixmap            */
XmDRAG_UNDER_NONE       /* no animation effects unless in XmNdragProc */
```

XmNdropRectangles
A list of rectangles that specify the shape of the drop site. When the value is NULL, the drop site is the entire widget.

XmNdropSiteActivity
Specifies the state of the drop site. Possible values:

```
XmDROP_SITE_ACTIVE          /* participates in drop operations */
XmDROP_SITE_INACTIVE        /* does not participate in drop operations */
```

XmNdropSiteOperations
The valid operations for a drop site. The value is a bit mask that is formed by combining one or more of these possible values:

```
XmDROP_COPY         /* copy operations are valid */
XmDROP_LINK         /* link operations are valid */
XmDROP_MOVE         /* move operations are valid */
XmDROP_NOOP         /* no operations are valid   */
```

XmNdropSiteType
The type of the drop site. Possible values:

```
XmDROP_SITE_SIMPLE      /* no children are registered as drop sites */
XmDROP_SITE_COMPOSITE   /* has children registered as drop sites    */
```

XmNimportTargets
The list of target atoms that the drop site accepts.

XmNnumDropRectangles
The number of rectangles in the XmNdropRectangles list.

XmNnumImportTargets
The number of atoms in the XmNimportTargets list.

Callback Resources

DropSite defines the following callback resources:

Callback	Reason Constant
XmNdragProc	XmCR_DROP_SITE_ENTER_MESSAGE XmCR_DROP_SITE_LEAVE_MESSAGE XmCR_DROP_SITE_MOTION_MESSAGE XmCR_OPERATION_CHANGED_MESSAGE
XmNdropProc	XmCR_DROP_MESSAGE

XmNdragProc
The procedure that is called when the drop site receives a crossing, motion, or operation changed message under the dynamic protocol. The reason passed to the procedure depends on the type of message that is received.

XmNdropProc
The procedure that is called when a drop operation occurs on the drop site.

Callback Structure

The XmNdragProc is passed the following structure:

```
typedef struct {
    int         reason;         /* reason the callback was called      */
    XEvent      *event;         /* event that triggered callback       */
    Time        timeStamp;      /* timestamp of logical event          */
    Widget      dragContext;    /* DragContext associated with operation */
    Position    x;              /* x-coordinate of pointer             */
    Position    y;              /* y-coordinate of pointer             */
```

```
    unsigned char   dropSiteStatus;   /* valid or invalid                */
    unsigned char   operation;        /* current operation               */
    unsigned char   operations;       /* supported operations            */
    Boolean         animate;          /* toolkit or receiver does animation */
} XmDragProcCallbackStruct, *XmDragProcCallback;
```

The XmNdragProc can change the value of the dropSiteStatus, operation, and operations fields in this structure. When the drag procedure completes, the toolkit uses the resulting values to initialize the corresponding fields in the callback structure passed to the initiating client's callbacks.

The XmNdropProc is passed the following structure:

```
typedef struct {
    int             reason;           /* reason the callback was called  */
    XEvent          *event;           /* event that triggered callback   */
    Time            timeStamp;        /* timestamp of logical event      */
    Widget          dragContext;      /* DragContext associated with operation */
    Position        x;                /* x-coordinate of pointer         */
    Position        y;                /* y-coordinate of pointer         */
    unsigned char   dropSiteStatus;   /* valid or invalid                */
    unsigned char   operation;        /* current operation               */
    unsigned char   operations;       /* supported operations            */
    unsigned char   dropAction;       /* drop or help                    */
} XmDropProcCallbackStruct, *XmDropProcCallback;
```

The XmNdropProc can change the value of the dropSiteStatus, operation, operations, and dropAction fields in this structure. When the drop procedure completes, the toolkit uses the resulting values to initialize the corresponding fields in the XmDropProcCallbackStruct callback structure passed to the initiating client's drop start callbacks.

The dropSiteStatus field in these structures specifies whether or not the drop site is valid. The toolkit initializes the value based on the XmNimportTargets resource of the DropSite and the XmNexportTargets resource of the DragContext. The possible values are XmDROP_SITE_VALID and XmDROP_SITE_INVALID.

The operations field in these structure specifies the set of operations supported for the data being dragged. The toolkit initializes the value based on the XmNdragOperations resource of the DragContext and the operation selected by the user. The operation field in these structures specifies the current operation. The toolkit initializes the value based on the value of operations and the XmNdropSiteOperations resource.

The animate field in the XmDragProcCallbackStruct specifies whether the toolkit or the receiving client handles the drag-under effects for the drop site. If the value is True, the toolkit handles the effects based on the XmNanimationStyle resource. Otherwise the receiver is responsible for providing drag-under effects.

The dropAction field in the XmDropProcCallbackStruct specifies the action associated with the drop, which is either a normal drop or a help action. The possible values are XmDROP and XmDROP_HELP.

See Also

`XmDropSiteConfigureStackingOrder`(1), `XmDropSiteEndUpdate`(1), `XmDropSiteQueryStackingOrder`(1), `XmDropSiteRegister`(1), `XmDropSiteRetrieve`(1), `XmDropSiteStartUpdate`(1), `XmDropSiteUnregister`(1), `XmDropSiteUpdate`(1), `XmDisplay`(1), `XmDragContext`(1), `XmDragIcon`(2), `XmDropTransfer`(2), `XmScreen`(2).

Name

XmDropTransfer widget class – an object used to store information about a drop transaction.

Synopsis

Public Header:
<Xm/DragDrop.h>

Class Name:
XmDropTransfer

Class Pointer:
xmDropTransferObjectClass

Class Hierarchy:
Object →DropTransfer

Instantiation:
widget = `XmDropTransferStart`(...)

Functions/Macros:
`XmDropTransferAdd`(), `XmDropTransferStart`()

Availability

Motif 1.2 and later.

Description

The XmDropTransfer object stores information that the toolkit needs to process a drop transaction. An application does not explicitly create a DropTransfer widget, but instead initiates a data transfer by calling `XmDropTransfer-Start`(), which initializes and returns a DropTransfer widget. If `XmDrop-TransferStart`() is called within an XmNdropProc, the data transfer starts after the callback returns. If no data needs to be transferred or the drop transaction is a failure, an application still needs to call `XmDropTransferStart`() with a failure status, so that the toolkit can complete the drag and drop operation.

The XmNtransferProc resource specifies a procedure of type XtSelectionCallbackProc that handles transferring the requested selection data. This procedure performs in conjunction with the underlying Xt selection mechanisms and is called for each type of data being transferred. Target types can be added after a transfer has started by calling the `XmDropTransferAdd`().

New Resources

DropTransfer defines the following resources:

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
: Pointer to an array of XmDropTransferEntryRec structures, which specifies the requested target data types for the source data. A XmDropTransferEntryRec is defined as follows:

```
typedef struct {
    XtPointer    client_data;    /* any additional information necessary */
    Atom         target;         /* the selection target type            */
} XmDropTransferEntryRec, *XmDropTransferEntry;
```

The drop transfer is done when all of the entries have been processed.

XmNincremental
: If True, the receiver uses the Xt incremental selection transfer mechanism. If False (default), the receiver uses atomic transfer.

XmNnumDropTransfers
: The number of entries in XmNdropTransfers. The transfer is complete if the value is set to 0 at any time.

XmNtransferProc
: A procedure of type XtSelectionCallbackProc that provides the requested selection values. The widget argument passed to this procedure is the DropTransfer widget and the selection atom is _MOTIF_DROP.

XmNtransferStatus
: The current status of the drop transfer. The receiving client updates this value when the transfer ends and the value is communicated to the initiator. Possible values:

XmTRANSFER_SUCCESS XmTRANSFER_FAILURE

Inherited Resources

DropTransfer inherits the following resource:

Resource	Inherited From
XmNdestroyCallback	Object

See Also

XmDropTransferAdd(1), XmDropTransferStart(1), XmTargetsAreCompatible(1), Object(2), XmDisplay(2), XmDragContext(2), XmDragIcon(2), XmDropTransfer(2), XmScreen(2).

Name

XmErrorDialog – an unmanaged MessageBox as a child of a DialogShell.

Synopsis

Public Header:
<Xm/MessageB.h>

Instantiation:
widget = `XmCreateErrorDialog`(...)

Functions/Macros:
`XmCreateErrorDialog`(), `XmMessageBoxGetChild`()

Description

An XmErrorDialog is a compound object created by a call to XmCreateErrorDialog() that an application can use to inform the user about any type of error. An ErrorDialog consists of a DialogShell with an unmanaged MessageBox widget as its child. The MessageBox resource XmNdialogType is set to XmDIALOG_ERROR. An ErrorDialog includes four components: a symbol, a message, three buttons, and a separator between the message and the buttons. By default, the symbol is an octagon with a diagonal slash. In Motif 1.2, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Cancel**, and **Help** by default.

Default Resource Values

An ErrorDialog sets the following default values for MessageBox resources:

Name	Default
XmNdialogType	XmDIALOG_ERROR
XmNsymbolPixmap	xm_error

Widget Hierarchy

When an ErrorDialog is created with a specified name, the DialogShell is named name*_popup* and the MessageBox is called *name***.**

See Also

`XmCreateObject`(1), `XmMessageBoxGetChild`(1), `XmDialogShell`(2), `XmMessageBox`(2).

Name

XmFileSelectionBox widget class – a widget for selecting files.

Synopsis

Public Header:
<Xm/FileSB.h>

Class Name:
XmFileSelectionBox

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmBulletinBoard →
XmSelectionBox →XmFileSelectionBox

Class Pointer:
xmFileSelectionBoxWidgetClass

Instantiation:
widget = `XmCreateFileSelectionBox` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmFileSelectionBoxWidgetClass,...)

Functions/Macros:
`XmCreateFileSelectionBox`(), `XmCreateFileSelectionDialog`(), `XmFileSelectionBoxGetChild`(), `XmFileSelectionDoSearch`(), `XmIsFileSelectionBox`()

Description

FileSelectionBox is a composite widget that is used to traverse a directory hierarchy and select files. FileSelectionBox provides a directory mask input field, a scrollable list of subdirectories, a scrollable list of filenames, a filename input field, and a group of four PushButtons. The names for the filter text, directory list, and directory list label are Text, DirList, and Dir respectively. The other components have the same names as the components in a SelectionBox.

In Motif 1.2, the button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Filter**, **Cancel**, and **Help** by default.

You can customize a FileSelectionBox by removing existing children or adding new children. Use `XmFileSelectionBoxGetChild`() to retrieve the widget ID of an existing child and then unmanage the child. With Motif 1.2, multiple widgets can be added as children of a FileSelectionBox. Additional children are added in the same way as for a SelectionBox. In Motif 1.1, only a single widget can be added as a child of a FileSelectionBox. This child is placed below the filename input field and acts as a work area.

In Motif 2.0 and later, the search pattern and base directory can be displayed in two separate text fields, depending on the value of the XmNpathMode resource. If the value is XmPATH_MODE_FULL, the behavior is consistent with that of Motif 1.2, and the filter text field (Text) contains the XmNdirMask resource. If the value is XmPATH_MODE_RELATIVE, the XmNdirectory resource is displayed in an additional text field which has the name **DirText**, with an accompanying label named **DirL**, and the filter text field **Text** contains the XmNpattern resource.

In some variants of CDE Motif, the directory pattern field may be replaced with an XmComboBox, providing a validset of directory locations. If the resource XmNenableFdbPickList is true, the FileSelectionBox creates an XmComboBox called **DirComboBox** in place of the **DirText** field.[1]

Traits

FileSelectionBox uses the XmQTactivatable trait.

New Resources

FileSelectionBox defines the following resources:

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
XmNdirTextLabelString	XmCDirTextLabelString	XmString	NULL	CSG
XmNfileFilterStyle	XmCFileFilterStyle	XtEnum	XmFILTER_NONE	SG
XmNfileListItems	XmCItems	XmStringTable	dynamic	SG
XmNfileListItemCount	XmCItemCount	int	dynamic	SG
XmNfileListLabelString	XmCFileListLabelString	XmString	dynamic	SG
XmNfileSearchProc	XmCFileSearchProc	XmSearchProc	default procedure	CSG
XmNfileTypeMask	XmCFileTypeMask	unsigned char	XmFILE_REGULAR	CSG
XmNfilterLabelString	XmCFilterLabelString	XmString	dynamic	CSG

1.XmNenableFdbPickList is implemented on Solaris 2.7 and above.

Name	Class	Type	Default	Access
XmNlistUpdated	XmCListUpdated	Boolean	dynamic	SG
XmNnoMatchString	XmCNoMatchString	XmString	XmUNSPECIFIED[a]	CSG
XmNpathMode	XmCPathMode	XtEnum	XmPATH_MODE_FULL	CSG
XmNpattern	XmCPattern	XmString	dynamic	CSG
XmNqualifySearchDataProc	XmCQualifySearchDataProc	XmQualifyProc	default procedure	CSG

a. Strictly speaking, more correct than the "[ ]" given in the 1st and 2nd editions. If the value is XmUNSPECIFIED, it defaults to this expression.

XmNdirectory
: The base directory that, in combination with XmNpattern, forms the directory mask (the XmNdirMask resource). The directory mask determines which files and directories to display.

XmNdirectoryValid
: A resource that can be set only by the directory search procedure (as specified by the XmNdirSearchProc resource). If the directory search procedure is unable to search the directory that was passed to it, then it will set XmNdirectoryValid to False, and as a result, the file search procedure won't be called.

XmNdirListItems
: The items in the directory list. This resource is set only by the directory search procedure. A call to XtGetValues() returns the actual list items (not a copy), so don't have your application free these items.

XmNdirListItemCount
: The number of items in XmNdirListItems. This resource is set only by the directory search procedure.

XmNdirListLabelString
: The string that labels the directory list. In Motif 1.2, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Directories".

XmNdirMask
: The directory mask that determines which files and directories to display. This value combines the values of the resources XmNdirectory and XmNpattern.

XmNdirSearchProc
: The procedure that performs directory searches. For most applications, the default procedure works just fine. The call to this procedure contains two arguments: the widget ID of the FileSelectionBox and a pointer to an XmFileSelectionBoxCallbackStruct.

XmNdirSpec
The complete specification of the file path. Synonymous with the XmNtextString resource in SelectionBox. It is the initial directory and file search that determines the default value for this resource.

XmNdirTextLabelString
In Motif 2.0 and later, specifies the label for the directory text field when the XmNpathMode resource is XmPATH_MODE_RELATIVE. The value is otherwise ignored.

XmNfileFilterStyle
In Motif 2.0 and later, controls the behaviour of the default file and directory search procedures in the way in which hidden files are displayed. If enabled, any file or directory beginning with '.' is filtered out. The exception to the rule is ".." which is not filtered out in the directory search procedure. Possible values:

```
XmFILTER_NONE          /* do not filter out any files or directories */
XmFILTER_HIDDEN_FILES  /* filter out file beginning with '.'         */
```

XmNfileListItems
The items in the file list. Synonymous with the XmNlistItems resource in SelectionBox. This resource is set only by the file search procedure. A call to `XtGetValues`() returns the actual list items (not a copy), so don't have your application free these items.

XmNfileListItemCount
The number of items in XmNfileListItems. Synonymous with the XmNlistItemCount resource in SelectionBox. This resource is set only by the file search procedure.

XmNfileListLabelString
The string that labels the file list. Synonymous with the XmNlistLabelString resource in SelectionBox. In Motif 1.2, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Files".

XmNfileSearchProc
The procedure that performs file searches. For most applications, the default procedure works just fine. The call to this procedure contains two arguments: the widget ID of the FileSelectionBox and a pointer to an XmFileSelectionBoxCallbackStruct.

XmNfileTypeMask
Determines whether the file list will display only regular files, only directories, or any type of file. Possible values are XmFILE_DIRECTORY, XmFILE_REGULAR, and XmFILE_ANY_TYPE.

XmNfilterLabelString
: The string that labels the field in which the directory mask is typed in by the user. In Motif 1.2, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Filter".

XmNlistUpdated
: A resource that can be set only by the directory search procedure or by the file search procedure. This resource is set to True if the directory or file list was updated by a search procedure.

XmNnoMatchString
: A string that displays in the file list when there are no filenames to display.

XmNpathMode
: In Motif 2.0 and later, specifies the way in which the filter string is presented in the file selection box. The layout can either contain a single text field for the filter, as specified by the XmNdirMask resource, or two separate text fields containing the XmNpattern and XmNdirectory resources. Possible values:

```
XmPATH_MODE_FULL      /* single text field for XmNdirMask          */
XmPATH_MODE_RELATIVE  /* 2 text fields for XmNpattern/XmNdirectory */
```

: When XmNpathMode is XmPATH_MODE_RELATIVE, the text field associated with the XmNpattern resource is labelled using the value of the XmNfilterLabelString resource, and the text field associated with the XmNdirectory resource is labelled from the XmNdirTextLabelString value.

XmNpattern
: The file search pattern that, in combination with XmNdirectory, forms the directory mask (the XmNdirMask resource). The directory mask determines which files and directories to display. If the XmNpattern resource defaults to NULL or is empty, a pattern for matching all files will be used.

XmNqualifySearchDataProc
: The procedure that generates a valid directory mask, base directory, and search pattern to be used by XmNdirSearchProc and XmNfileSearchProc (the search procedures for directories and files). For most applications, the default procedure works just fine. The call to this procedure contains three arguments: the widget ID of the FileSelectionBox, a pointer to an XmFileSelectionBoxCallbackStruct containing the input data, and a pointer to an XmFileSelectionBoxCallbackStruct that will contain the output data.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* reason that the callback was called */
    XEvent      *event;         /* event that triggered callback       */
    XmString    value;          /* current value of XmNdirSpec resource */
    int         length;         /* number of bytes in value member     */
    XmString    mask;           /* current value of XmNdirMask resource */
    int         mask_length;    /* number of bytes in mask member      */
    XmString    dir;            /* current base directory              */
    int         dir_length;     /* number of bytes in dir member       */
    XmString    pattern;        /* current search pattern              */
    int         pattern_length; /* number of bytes in pattern member   */
} XmFileSelectionBoxCallbackStruct;
```

Inherited Resources

FileSelectionBox inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. FileSelectionBox sets the default values of XmNautoUnmanage to False and XmNdialogType to XmDIALOG_FILE_SELECTION. It also sets the default values of XmNlistItems and XmNlistItemCount dynamically. The default value of XmNborderWidth is reset to 0 by Manager. BulletinBoard sets the value of XmNinitialFocus to XmNdefaultButton and resets the default XmNshadowThickness from 0 to 1 if the FileSelectionBox is a child of a DialogShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNlistItemCount	XmSelectionBox
XmNallowOverlap	XmBulletinBoard	XmNlistItems	XmSelectionBox
XmNancestorSensitive	Core	XmNlistLabelString	XmSelectionBox
XmNapplyCallback	XmSelectionBox	XmNlistVisibleItemCount	XmSelectionBox
XmNapplyLabelString	XmSelectionBox	XmNmapCallback	XmBulletinBoard
XmNautoUnmanage	XmBulletinBoard	XmNmappedWhenManaged	Core
XmNbackground	Core	XmNmarginHeight	XmBulletinBoard
XmNbackgroundPixmap	Core	XmNmarginWidth	XmBulletinBoard
XmNborderColor	Core	XmNminimizeButtons	XmSelectionBox
XmNborderPixmap	Core	XmNmustMatch	XmSelectionBox
XmNborderWidth	Core	XmNnavigationType	XmManager
XmNbottomShadowColor	XmManager	XmNnoMatchCallback	XmSelectionBox
XmNbottomShadowPixmap	XmManager	XmNnoResize	XmBulletinBoard

Resource	Inherited From	Resource	Inherited From
XmNbuttonFontList	XmBulletinBoard	XmNnumChildren	Composite
XmNbuttonRenderTable	XmBulletinBoard	XmNokCallback	XmSelectionBox
XmNcancelButton	XmBulletinBoard	XmNokLabelString	XmSelectionBox
XmNcancelCallback	XmSelectionBox	XmNpopupHandlerCallback	XmManager
XmNcancelLabelString	XmSelectionBox	XmNresizePolicy	XmBulletinBoard
XmNchildren	Composite	XmNscreen	Core
XmNchildPlacement	XmSelectionBox	XmNselectionLabelString	XmSelectionBox
XmNcolormap	Core	XmNsensitive	Core
XmNdefaultButton	XmBulletinBoard	XmNshadowThickness	XmManager
XmNdefaultPosition	XmBulletinBoard	XmNshadowType	XmBulletinBoard
XmNdepth	Core	XmNstringDirection	XmManager
XmNdestroyCallback	Core	XmNtextAccelerators	XmSelectionBox
XmNdialogStyle	XmBulletinBoard	XmNtextColumns	XmSelectionBox
XmNdialogTitle	XmBulletinBoard	XmNtextFontList	XmBulletinBoard
XmNdialogType	XmSelectionBox	XmNtextRenderTable	XmBulletinBoard
XmNfocusCallback	XmBulletinBoard	XmNtextString	XmSelectionBox
XmNforeground	XmManager	XmNtextTranslations	XmBulletinBoard
XmNheight	Core	XmNtopShadowColor	XmManager
XmNhelpCallback	XmManager	XmNtopShadowPixmap	XmManager
XmNhelpLabelString	XmSelectionBox	XmNtranslations	Core
XmNhighlightColor	XmManager	XmNtraversalOn	XmManager
XmNhighlightPixmap	XmManager	XmNunitType	XmManager
XmNinitialFocus	XmManager	XmNunmapCallback	XmBulletinBoard
XmNinitialResourcesPersistent	Core	XmNuserData	XmManager
XmNinsertPosition	Composite	XmNwidth	Core
XmNlabelFontList	XmBulletinBoard	XmNx	Core
XmNlabelRenderTable	XmBulletinBoard	XmNy	Core
XmNlayoutDirection	XmManager		

Translations

The translations for FileSelectionBox are inherited from SelectionBox.

Action Routines

FileSelectionBox defines the following action routines:

SelectionBoxUpOrDown(flag)

Replaces the selection text or the filter text, depending on which one has the keyboard focus. That is, this action replaces either: the text string in the selection area with an item from the file list, or the text string in the directory mask (filter) area with an item from the directory list.

The value of flag determines which file list item or which directory list item is selected as the replacement string. A flag value of 0, 1, 2, or 3 selects the previous, next, first, or last item, respectively, of the appropriate list.

SelectionBoxRestore()

Replaces the selection text or the filter text, depending on which one has the keyboard focus. That is, this action replaces either: the text string in the selection area with the currently selected item in the file list (clearing the selection area if no list item is selected), or the text string in the filter area with a new directory mask (which is formed by combining the values of the XmNdirectory and XmN--pattern resources).

Additional Behavior

FileSelectionBox has the following additional behavior:

MAny KCancel

If the **Cancel** button is sensitive, invokes its XmNactivateCallback callbacks. If there is no **Cancel** button, the event is passed to the parent if it is a manager.

KActivate

In the filename text input area, first invokes the XmNactivateCallback callbacks for the text and then invokes either the XmNnoMatchCallback or the XmNokCallback callbacks based on the value of XmNmustMatch**.**

In the directory mask text input area, first invokes the XmNactivateCallback callbacks for the text and then starts a directory and file search and invokes the XmNapplyCallback callbacks.

In the directory list, invokes the XmNdefaultActionCallback callback, begins a directory and file search, and invokes the XmNapplyCallback callbacks.

In the file list, invokes XmNdefaultActionCallback and XmNokCallback callbacks.

When neither of these areas nor any button has the keyboard focus, it invokes the callbacks in either XmNnoMatchCallback or XmNokCallback depending on the value of XmNmustMatch and whether or not the selection text matches a file in the file list.

<DoubleClick>
In the directory or file list, has the same behavior as KActivate.

<Single Select> or <Browse Select>
In the directory list, composes a directory mask using the selected directory item and the current pattern. In the file list, uses the selected file item to replace the selection text.

BTransfer
In Motif 1.2, in the file or directory list, starts a drag and drop operation using the selected items in the list. If BTransfer is pressed over an unselected item, only that item is used in the drag and drop operation.

<Apply Button Activated>
Starts a directory and file search and invokes the XmNapplyCallback callbacks.

<Ok Button Activated>
Invokes either the XmNnoMatchCallback or XmNokCallback callbacks based on the value of XmNmustMatch and whether or not the selection text matches a file in the file list.

<Cancel Button Activated>
Invokes the XmNcancelCallback callbacks.

<Help Button Activated>
Invokes the XmNhelpCallback callbacks.

See Also

`XmCreateObject`(1), `XmFileSelectionBoxGetChild`(1), `XmFileSelectionDoSearch`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmBulletinBoard`(2), `XmFileSelectionDialog`(2), `XmManager`(2), `XmSelectionBox`(2).

Name

XmFileSelectionDialog – an unmanaged FileSelectionBox as a child of a Dialog Shell.

Synopsis

Public Header:
<Xm/FileSB.h>

Instantiation:
widget = XmCreateFileSelectionDialog(...)

Functions/Macros:
`XmCreateFileSelectionBox`(), `XmFileSelectionBoxGetChild`(), `XmCreateFileSelectionDialog`(), `XmFileSelectionDoSearch`(), `XmIsFileSelectionBox`()

Description

An XmFileSelectionDialog is a compound object created by a call to XmCreate-FileSelectionDialog() that an application can use to allow a user to select a file from a dialog box. A FileSelectionDialog consists of a DialogShell with an unmanaged FileSelectionBox widget as its child. The SelectionBox resource XmNdialogType is set to XmDIALOG_FILE_SELECTION.

A FileSelectionDialog provides a directory mask input field, a scrollable list of subdirectories, a scrollable list of filenames, a filename input field, and a group of four PushButtons. In Motif 1.2, the button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Filter**, **Cancel**, and **Help** by default.

Default Resource Values

A FileSelectionDialog sets the following default values for its resources:

Name	Default
XmNdialogType	XmDIALOG_FILE_SELECTION

Widget Hierarchy

When a FileSelectionDialog is created with a specified name, the DialogShell is named name*_popup* and the FileSelectionBox is called *name*.

See Also

`XmCreateObject`(1), `XmFileSelectionBoxGetChild`(1), `XmFileSelectionDoSearch`(1), `XmFileSelectionBox`(2), `XmDialogShell`(2).

Name

XmForm widget class – a container widget that constrains its children.

Synopsis

Public Header:
<Xm/Form.h>

Class Name:
XmForm

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmBulletinBoard →
XmForm

Class Pointer:
xmFormWidgetClass

Instantiation:
widget = `XmCreateForm` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmFormWidgetClass,...)

Functions/Macros:
`XmCreateForm`(), `XmCreateFormDialog`(), `XmIsForm`()

Description

Form is a container widget that constrains its children so as to define their layout when the Form is resized. Constraints on the children of a Form specify the attachments for each of the four sides of a child. Children may be attached to each other, to edges of the Form, or to relative positions within the Form.

New Resources

Form defines the following resources:

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
The denominator part of the fraction that describes a child's relative position within a Form. The numerator of this fraction is one of the four positional constraint resources: XmNbottomPosition, XmNleftPosition, XmNrightPosition, or XmNtopPosition. For example, suppose you use the default XmNfractionBase of 100. Then, if you specify XmNtopPosition as 30, the top of the child will remain

invariably attached to a location that is 30/100 (or 30 percent) from the top of the Form. (In other words, resizing the Form's height might change the absolute position of the child's top, but not its position relative to the top of the Form.) Similarly, a value of 50 for XmNleftPosition ensures that the left side of the child is attached 50/100 from the left of the Form (or in this case, halfway between the left and right side). Note that these fractions are implemented only when the child's corresponding attachment constraint is set to XmATTACH_POSITION. (The attachment constraints are XmNbottomAttachment, XmNleftAttachment, XmNrightAttachment, and XmNtopAttachment.)

XmNhorizontalSpacing
The offset for right and left attachments.

XmNrubberPositioning
Defines the default behavior of a child's top and left side, in the absence of other settings. If this resource is False (default), the child's top and left sides are positioned using absolute values. If True, the child's top and left sides are positioned relative to the size of the Form.

XmNverticalSpacing
The offset for top and bottom attachments.

New Constraint Resources

Form defines the following constraint resources for its children:

Name	Class	Type	Default	Access
XmNbottomAttachment	XmCAttachment	unsigned char	XmATTACH_NONE	CSG
XmNbottomOffset	XmCOffset	int	0	CSG
XmNbottomPosition	XmCAttachment	int	0	CSG
XmNbottomWidget	XmCWidget	Widget	NULL	CSG
XmNleftAttachment	XmCAttachment	unsigned char	XmATTACH_NONE	CSG
XmNleftOffset	XmCOffset	int	0	CSG
XmNleftPosition	XmCAttachment	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	XmCAttachment	int	0	CSG
XmNrightWidget	XmCWidget	Widget	NULL	CSG
XmNtopAttachment	XmCAttachment	unsigned char	XmATTACH_NONE	CSG
XmNtopOffset	XmCOffset	int	0	CSG

Name	Class	Type	Default	Access
XmNtopPosition	XmCAttachment	int	0	CSG
XmNtopWidget	XmCWidget	Widget	NULL	CSG

XmNbottomAttachment
The method of attachment for the child's bottom side. Each of the four attachment resources (XmNtopAttachment, XmNbottomAttachment, XmNleftAttachment, and XmNrightAttachment) has the following possible values. The comments below refer to a corresponding edge (top, bottom, left, or right) of the child widget within the Form.

```
XmATTACH_NONE           /* remains unattached                    */
XmATTACH_FORM           /* attached to same edge of Form         */
XmATTACH_OPPOSITE_FORM  /* attached to other edge of Form        */
XmATTACH_WIDGET         /* abuts an adjacent widget              */
XmATTACH_OPPOSITE_WIDGET /* attached to other edge of adjacent   */
                        /*   widget                              */
XmATTACH_POSITION       /* relative to a dimension of Form       */
XmATTACH_SELF           /* relative to its current position and*/
                        /* to Form                               */
```

XmNbottomOffset
The distance between the child's bottom side and the object it's attached to. Offsets are absolute. Offsets are of type int and may not be resolution-independent.

XmNbottomPosition
Used in conjunction with XmNfractionBase to calculate the position of the bottom of a child, relative to the bottom of the Form. This resource has no effect unless the child's XmNbottomAttachment resource is set to XmATTACH_POSITION. (See XmNfractionBase for details.)

XmNbottomWidget
The name of the widget or gadget that serves as the attachment point for the bottom of the child. To use this resource, set the XmNbottomAttachment resource to either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.

XmNleftAttachment
The method of attachment for the child's left side.

XmNleftOffset
The distance between the child's left side and the object it's attached to. Offsets are absolute. Offsets are of type int and may not be resolution-independent.

XmNleftPosition
: Used in conjunction with XmNfractionBase to calculate the position of the left side of a child, relative to the left side of the Form. This resource has no effect unless the child's XmNleftAttachment resource is set to XmATTACH_POSITION. (See XmNfractionBase for details.)

XmNleftWidget
: The name of the widget or gadget that serves as the attachment point for the left side of the child. To use this resource, set the XmNleftAttachment resource to either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.

XmNresizable
: If True (default), a child's resize request is accepted by the Form, provided that the child isn't constrained by its attachments. That is, if both the left and right sides of a child are attached, or if both the top and bottom are attached, the resize request fails, whereas if the child has only one horizontal or one vertical attachment, the resize request is granted. If this resource is False, the child is never resized.

XmNrightAttachment
: The method of attachment for the child's right side.

XmNrightOffset
: The distance between the child's right side and the object it's attached to. Offsets are absolute. Offsets are of type int and may not be resolution-independent.

XmNrightPosition
: Used in conjunction with XmNfractionBase to calculate the position of the right side of a child, relative to the right side of the Form. This resource has no effect unless the child's XmNrightAttachment resource is set to XmATTACH_POSITION. (See XmNfractionBase for details.)

XmNrightWidget
: The name of the widget or gadget that serves as the attachment point for the right side of the child. To use this resource, set the XmNrightAttachment resource to either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.

XmNtopAttachment
: The method of attachment for the child's top side.

XmNtopOffset
: The distance between the child's top side and the object it's attached to. Offsets are absolute. Offsets are of type int and may not be resolution-independent.

XmNtopPosition
: Used in conjunction with XmNfractionBase to calculate the position of the top of a child, relative to the top of the Form. This resource has no effect unless the child's XmNtopAttachment resource is set to XmATTACH_POSITION. (See XmN-fraction-Base for details.)

XmNtopWidget

The name of the widget or gadget that serves as the attachment point for the top of the child. To use this resource, set the XmNtopAttachment resource to either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.

Inherited Resources

Form inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. Form sets the default values of XmN-marginWidth and XmN-margin-Height to 0. The default value of XmNborder-Width is reset to 0 by Manager. BulletinBoard sets the value of XmNinitialFocus to XmNdefaultButton and resets the default XmNshadowThickness from 0 to 1 if the Form widget is a child of DialogShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNlabelFontList	XmBulletinBoard
XmNallowOverlap	XmBulletinBoard	XmNlabelRenderTable	XmBulletinBoard
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNautoUnmanage	XmBulletinBoard	XmNmapCallback	XmBulletinBoard
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNmarginHeight	XmBulletinBoard
XmNborderColor	Core	XmNmarginWidth	XmBulletinBoard
XmNborderPixmap	Core	XmNnavigationType	XmManager
XmNborderWidth	Core	XmNnoResize	XmBulletinBoard
XmNbottomShadowColor	XmManager	XmNnumChildren	Composite
XmNbottomShadowPixmap	XmManager	XmNpopupHandlerCallback	XmManager
XmNbuttonFontList	XmBulletinBoard	XmNresizePolicy	XmBulletinBoard
XmNbuttonRenderTable	XmBulletinBoard	XmNscreen	Core
XmNcancelButton	XmBulletinBoard	XmNsensitive	Core
XmNchildren	Composite	XmNshadowThickness	XmManager
XmNcolormap	Core	XmNshadowType	XmBulletinBoard
XmNdefaultButton	XmBulletinBoard	XmNstringDirection	XmManager
XmNdefaultPosition	XmBulletinBoard	XmNtextFontList	XmBulletinBoard
XmNdepth	Core	XmNtextRenderTable	XmBulletinBoard
XmNdestroyCallback	Core	XmNtextTranslations	XmBulletinBoard
XmNdialogStyle	XmBulletinBoard	XmNtopShadowColor	XmManager
XmNdialogTitle	XmBulletinBoard	XmNtopShadowPixmap	XmManager
XmNfocusCallback	XmBulletinBoard	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager

Resource	Inherited From	Resource	Inherited From
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNunmapCallback	XmBulletinBoard
XmNhighlightColor	XmManager	XmNuserData	XmManager
XmNhighlightPixmap	XmManager	XmNwidth	Core
XmNinitialFocus	XmManager	XmNx	Core
XmNinitialResourcesPersistent	Core	XmNy	Core
XmNinsertPosition	Composite		

Translations

The translations for Form are inherited from XmBulletinBoard.

See Also

`XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmBulletinBoard`(2), `XmFormDialog`(2), `XmManager`(2).

Name

XmFormDialog – an unmanaged Form as a child of a DialogShell.

Synopsis

Public Header:
<Xm/Form.h>

Instantiation:
widget = XmCreateFormDialog(...)

Functions/Macros:
`XmCreateFormDialog`()

Description

An XmFormDialog is a compound object created by a call to `XmCreateFormDialog`() that is useful for creating custom dialogs. A FormDialog consists of a DialogShell with an unmanaged Form widget as its child. The FormDialog does not contain any labels, buttons, or other dialog components; these components are added by the application.

Widget Hierarchy

When a FormDialog is created with a specified name, the DialogShell is named name*_popup* and the Form is called *name*.

See Also

`XmCreateObject`(1), `XmDialogShell`(2), `XmForm`(2).

Name

XmFrame widget class – a manager widget that places a border around a single child.

Synopsis

Public Header:
<Xm/Frame.h>

Class Name:
XmFrame

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmFrame

Class Pointer:
xmFrameWidgetClass

Instantiation:
widget = `XmCreateFrame` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmFrameWidgetClass,...)

Functions/Macros:
`XmCreateFrame`(), `XmIsFrame`()

Description

Frame is a simple subclass of Manager that places a three-dimensional border around a single child. Frame is used to provide the typical Motif-style appearance for widget classes that do not have a visible frame, such as RowColumn.

As of Motif 1.2, a Frame can have two children: a work area child and a title child. The widget uses constraint resources to indicate the type of each child and to specify the alignment of the title child.

New Resources

Frame defines the following resources:

Name	Class	Type	Default	Access
XmNmarginHeight	XmCMarginHeight	Dimension	0	CSG
XmNmarginWidth	XmCMarginWidth	Dimension		CSG
XmNshadowType	XmCShadowType	unsigned char	dynamic	CSG

XmNmarginHeight
The spacing between the top or bottom of a Frame widget's child and the shadow of the Frame widget.

XmNmarginWidth
The spacing between the right or left side of a Frame widget's child and the shadow of the Frame widget.

XmNshadowType
The style in which Frame widgets are drawn. Possible values:

```
XmSHADOW_IN             /* widget appears inset          */
XmSHADOW_OUT            /* widget appears outset         */
XmSHADOW_ETCHED_IN      /* double line; widget appears inset */
XmSHADOW_ETCHED_OUT     /* double line; widget appears raised */
```

New Constraint Resources

As of Motif 1.2, Frame defines the following constraint resources for its children:

Name	Class	Type	Default	Access
XmNchildType	XmCChildType	unsigned char	XmFRAME_WORKAREA_CHILD	CSG
XmNchildHorizontalAlignment	XmCChildHorizontalAlignment	unsigned char	XmALIGNMENT_BEGINNING	CSG
XmNchildHorizontalSpacing	XmCChildHorizontalSpacing	Dimension	dynamic	CSG
XmNchildVerticalAlignment	XmCChildVerticalAlignment	unsigned char	XmALIGNMENT_CENTER	CSG
XmNframeChildType	XmCFrameChildType	unsigned char	XmFRAME_WORKAREA_CHILD	CSG

XmNchildType
The type of the child. Frame supports one title and one work area child. Possible values:

```
XmFRAME_TITLE_CHILD     /* child is the title    */
XmFRAME_WORKAREA_CHILD  /* child is the work area */
XmFRAME_GENERIC_CHILD   /* child is ignored      */
```

From Motif 2.0 and later, the XmNchildType resource is deprecated, and the XmNframeChildType resource is the preferred method.

XmNchildHorizontalAlignment
The alignment (left to right) for a Frame's title. Possible values are:

```
XmALIGNMENT_BEGINNING
XmALIGNMENT_CENTER
XmALIGNMENT_END
```

XmNchildHorizontalSpacing
The minimum distance between the title text and the Frame shadow. The title is clipped to maintain this distance. The value of XmNmarginWidth is used as the default value.

XmNchildVerticalAlignment
The alignment of the Frame's title relative to the top shadow of the Frame. Possible values:

XmALIGNMENT_BASELINE_BOTTOM
XmALIGNMENT_BASELINE_TOP
XmALIGNMENT_WIDGET_TOP
XmALIGNMENT_CENTER
XmALIGNMENT_WIDGET_BOTTOM

XmNframeChildType
Introduced in Motif 2.0 as part of a rationalization in the naming of constraint resources. The behavior of the XmNframeChildType resource is identical in all respects to the XmNchildType resource, which is now deprecated.

Inherited Resources

Frame inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. Frame sets the default value of XmNshadowThickness to 1 if the Frame is a child of a Shell and 2 otherwise. The default value of XmNborderWidth is reset to 0 by Manager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core

Resource	Inherited From	Resource	Inherited From
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Translations

The translations for Frame are inherited from XmManager.

See Also

`XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmManager`(2).

Name

XmGadget widget class – the fundamental class for windowless widgets.

Synopsis

Public Header:
<Xm/Xm.h>

Class Name:
XmGadget

Class Hierarchy:
Object →RectObj →XmGadget

Class Pointer:
xmGadgetClass

Instantiation:
Gadget is a meta-class and is not normally instantiated.

Functions/Macros:
`XmIsGadget`()

Description

Gadget is a supporting superclass for other gadget classes. Gadget takes care of drawing and highlighting border shadows as well as managing traversal.

In versions of Motif prior to 2.0, a gadget is drawn using pixmap and color resources taken from the Manager parent. Changing such a resource in a Manager (for example, XmNforeground) also affects all gadget children. Gadgets sharing the same parent therefore also share the same general appearance.

In Motif 2.0 and later, the Gadget class supports independent appearance resources. For example, Gadgets sharing the same parent can have different XmNforeground values. Where a particular appearance resource is unspecified for a Gadget, the default value is taken from the Manager parent.

Traits

Gadget holds the XmQTspecifyLayoutDirection, XmQTaccessColors, and XmQTspecifyUnitType traits, which are inherited by any derived classes, and uses the XmQTspecifyUnhighlight trait.

New Resources

Gadget defines the following resources:

Name	Class	Type	Default	Access
XmNbackground	XmCBackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	XmCPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNbottomShadowColor	XmCBottomShadowColor	Pixel	dynamic	CSG
XmNbottomShadowPixmap	XmCBottomShadowPixmap	Pixmap	dynamic	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOnEnter	Boolean	False	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	XmCHighlightThickness	Dimension	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	unsigned char	dynamic	CG
XmNnavigationType	XmCNavigationType	unsigned char	XmNONE	CSG
XmNshadowThickness	XmNShadowThickness	Dimension	dynamic	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

XmNbackground
: In Motif 2.0 and later, specifies the background color for the Gadget.

XmNbackgroundPixmap
: In Motif 2.0 and later, specifies the background pixmap for tiling the Gadget. The default is XmUNSPECIFIED_PIXMAP.

XmNbottomShadowColor
: Specifies the color used in drawing the border shadow’s bottom and right sides.

XmNbottomShadowPixmap
: In Motif 2.0 and later, specifies the pixmap for drawing the bottom and right sides of the Gadget border shadow.

XmNforeground
In Motif 2.0 and later, specifies the foreground color for the Gadget.

XmNhighlightColor
Specifies the color used in drawing the highlighting rectangle.

XmNhighlightOnEnter
Determines whether to draw a gadget's highlighting rectangle whenever the cursor moves into the gadget. This resource applies only when the shell has a focus policy of XmPOINTER. If the XmNhighlightOnEnter resource is True, highlighting is drawn; if False (default), highlighting is not drawn.

XmNhighlightPixmap
In Motif 2.0 and later, specifies the pixmap used for drawing the highlighting rectangle.

XmNhighlightThickness
The thickness of the highlighting rectangle. In Motif 2.0 and earlier, the default is 2. In Motif 2.1 and later, the default depends upon the XmDisplay XmNenableThinThickness resource: if True, the default is 1, otherwise 2.

XmNlayoutDirection
In Motif 2.0 and later, specifies the direction in which components (for example, strings) of the Gadget are laid out. If unspecified, the value is inherited from the Manager parent, or from the nearest ancestor which has the XmQTspecifyLayoutDirection trait. Possible values:

XmLEFT_TO_RIGHT XmRIGHT_TO_LEFT

XmNnavigationType
Determines the way in which gadgets are to be traversed during keyboard navigation. Possible values:

```
XmNONE                  /* exclude from keyboard navigation  */
                        /*    (default for non-shell parent)     */
XmTAB_GROUP             /* include in keyboard navigation     */
                        /*    (default when parent is a shell)   */
XmSTICKY_TAB_GROUP      /* include in keyboard navigation, even if  */
                        /* XmAddTabGroup() was called          */
XmEXCLUSIVE_TAB_GROUP   /* application defines order of navigation */
```

XmNshadowThickness
The thickness of the shadow border. In Motif 2.0 and earlier, the default is 2. In Motif 2.1 and later, the default depends upon the XmDisplay XmNenableThinThickness resource: if True, the default is 1, otherwise 2.

XmNtopShadowColor
Specifies the color used in drawing the border shadow's top and left sides.

XmNtopShadowPixmap
In Motif 2.0 and later, specifies the pixmap used in drawing the border shadow's top and left sides.

XmNtraversalOn
If True (default), traversal of this gadget is made possible.

XmNunitType
The measurement units to use in resources that specify a size or position--for example, any resources of data type Dimension (whose names generally include one of the words "Margin" or "Thickness"). For a gadget whose parent is a XmManager subclass, the default value is copied from this parent (provided the value hasn't been explicitly set by the application); otherwise, the default is XmPIXELS. Possible values:

XmPIXELS		Xm100TH_POINTS
Xm100TH_MILLIMETERS		Xm100TH_FONT_UNITS
Xm1000TH_INCHES		
XmINCHES	(2.0)	
XmPOINTS	(2.0)	
XmFONT_UNITS	(2.0)	

XmNuserData
A pointer to data that the application can attach to the gadget. This resource is unused internally.

Callback Resources

Gadget defines the following callback resources:

Callback	Reason Constant
XmNhelpCallback	XmCR_HELP

XmNhelpCallback
List of callbacks that are called when help is requested.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int      reason;     /* the reason that the callback was called */
    XEvent   *event;     /* event structure that triggered callback */
} XmAnyCallbackStruct;
```

Inherited Resources

Gadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. Gadget resets the default value of XmNborderWidth from 1 to 0.

Name	Inherited From
XmNancestorSensitive	RectObj
XmNborderWidth	RectObj
XmNdestroyCallback	Object
XmNheight	RectObj
XmNsensitive	RectObj
XmNwidth	RectObj
XmNx	RectObj
XmNy	RectObj

Behavior

Since Gadgets cannot have translations associated with them, a Gadget's behavior is controlled by the XmManager widget that contains the Gadget. If a Gadget has the keyboard focus, the XmManager handles passing events to the Gadget.

See Also

`Object`(2), `RectObj`(2), `XmManager`(2), `XmScreen`(2).

Name

XmGrabShell widget class – a popup shell that grabs the keyboard and pointer when mapped

Synopsis

Public Header:
<Xm/GrabShell.h>

Class Name:
XmGrabShell

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell →XmGrabShell

Class Pointer:
xmGrabShellWidgetClass

Instantiation:
widget = `XmCreateGrabShell` (parent, name,...)
or
widget = `XtCreatePopupShell`[1] (name, xmGrabShellWidgetClass,...)

Functions/Macros:
`XmCreateGrabShell`(), `XmIsGrabShell`()

Availability

Motif 2.0 and later.

Description

GrabShell is a shell widget which grabs the pointer and keyboard when it is mapped. The purpose of this is to provide a popup which immediately directs focus to its child. The ComboBox widget utilizes this feature in implementing its popup List.

Although GrabShell is an internal widget used by the ComboBox, it has a public interface, and is therefore available for use.

New Resources

GrabShell defines the following resources:

Name	Class	Type	Default	Access
XmNbottomShadowColor	XmCBottomShadowColor	Pixel	dynamic	CSG
XmNbottomShadowPixmap	XmCBottomShadowPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG

1.More precise than XtCreateWidget as given in 2nd edition.

Name	Class	Type	Default	Access
XmNgrabStyle	XmCGrabStyle	int	GrabModeAsync	CSG
XmNownerEvents	XmCOwnerEvents	Boolean	False	CSG
XmNshadowThickness	XmCShadowThickness	Dimension	2	CSG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG

XmNbottomShadowColor
The color used in drawing the border shadow's bottom and right sides, but only if XmNbottomShadowPixmap is NULL.

XmNbottomShadowPixmap
The pixmap used in drawing the border shadow's bottom and right sides.

XmNgrabStyle

Controls the further processing of pointer events once the grab has been initiated. Possible values:

GrabModeSync GrabModeAsync

Refer to Xlib documentation on `XGrabKeyboard`() and `XGrabPointer`() for more information.

XmNownerEvents
Specifies whether pointer or keyboard events are reported normally within the application, or only to the GrabShell window. Refer to Xlib documentation on `XGrabKeyboard`() and `XGrabPointer`() for more information.

XmNshadowThickness
The thickness of the shadow border.

XmNtopShadowColor
The color used in drawing the border shadow's top and left sides, but only if XmNtopShadowPixmap is NULL.

XmNtopShadowPixmap
The pixmap used in drawing the border shadow's top and left sides.

Inherited Resources

GrabShell inherits the resources shown below. The resources are listed alphabetically, along with the superclass that defines them. GrabShell resets XmNallowShellResize to True, XmNoverrideRedirect to True, XmNtransient to True, XmNwaitForWm to False, and XmNsaveUnder to False.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNmaxAspectX	WMShell
XmNallowShellResize	Shell	XmNmaxAspectY	WMShell
XmNancestorSensitive	Core	XmNmaxHeight	WMShell
XmNaudibleWarning	VendorShell	XmNmaxWIdth	WMShell
XmNbackground	Core	XmNminAspectX	WMShell
XmNbackgroundPixmap	Core	XmNminAspectY	WMShell
XmNbaseHeight	WMShell	XmNminHeight	WMShell
XmNbaseWidth	WMShell	XmNminWidth	WMShell
XmNborderColor	Core	XmNmwmDecorations	VendorShell
XmNborderPixmap	Core	XmNmwmFunctions	VendorShell
XmNborderWidth	Core	XmNmwmInputMode	VendorShell
XmNbuttonFontList	VendorShell	XmNmwmMenu	VendorShell
XmNbuttonRenderTable	VendorShell	XmNnumChildren	Composite
XmNchildren	Composite	XmNoverrideRedirect	Shell
XmNcolormap	Core	XmNpopdownCalback	Shell
XmNcreatePopupChildProc	Shell	XmNpopupCallback	Shell
XmNdefaultFontList	VendorShell	XmNpreeditType	VendorShell
XmNdeleteResponse	VendorShell	XmNsaveUnder	Shell
XmNdepth	Core	XmNscreen	Core
XmNdestroyCallback	Core	XmNsensitive	Core
XmNgeometry	Shell	XmNshellUnitType	VendorShell
XmNheight	Core	XmNtextFontList	VendorShell
XmNheightInc	WMShell	XmNtextRenderTable	VendorShell
XmNiconMask	WMShell	XmNtitle	WMShell
XmNiconPixmap	WMShell	XmNtitleEncoding	WMShell
XmNiconWindow	WMShell	XmNtransient	WMShell
XmNinitialResourcesPersistent	Core	XmNtranslations	Core
XmNinitialState	WMShell	XmNvisual	Shell
XmNinput	WMShell	XmNwaitForWm	WMShell
XmNinputMethod	VendorShell	XmNwidth	Core
XmNinputPolicy	VendorShell	XmNwidthInc	WMShell
XmNinsertPosition	Composite	XmNwindowGroup	WMShell
XmNkeyboardFocusPolicy	VendorShell	XmNwinGravity	WMShell

Resource	Inherited From	Resource	Inherited From
XmNlabelFontList	VendorShell	XmNwmTimeout	WMShell
XmNlabelRenderTable	VendorShell	XmNx	Core
XmNlayoutDirection	VendorShell	XmNy	Core
XmNmappedWhenManaged	Core		

Translations

The translations for GrabShell include those of WMShell.

Event	Action
BSelect Press	GrabShellBtnDown()
BSelect Release	GrabShellBtnUp()

Action Routines

GrabShell defines the following action routines:

GrabShellBtnDown()

If the event occurs outside the coordinates of the GrabShell, the widget is popped up, otherwise the event is ignored.

GrabShellBtnUp()

If the event occurs within the time specified by the multi-click interval of the display, the action ignores the event. Otherwise, the GrabShell is popped down.

GrabShellPopdown()

Grabs placed upon the pointer and keyboard are released, the GrabShell is unmapped, and the focus is reverted to the previous owner.

See Also

`XmCreateObject`(1), `Composite`(2), `Core`(2), `Shell`(2), `VendorShell`(2), `WMShell`(2)

Name

XmIconGadget widget class – a gadget for displaying both text and a pixmap

Synopsis

Public Header:
<Xm/IconG.h>

Class Name:
XmIconGadget

Class Hierarchy:
Object →RectObj →XmGadget →XmIconGadget

Class Pointer:
xmIconGadgetClass

Instantiation:
widget = `XmCreateIconGadget` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmIconGadgetClass,...)

Functions/Macros:
`XmCreateIconGadget(), XmIsIconGadget`()

Availability

Motif 2.0 and later.

Description

IconGadget is a gadget which can display both textual and pixmap information simultaneously. The textual data can be either centered below the pixmap, or placed to the side, depending upon the value of the XmNviewType resource. The value XmLARGE_ICON centers below, and XmSMALL_ICON horizontally aligns, the pixmap and textual information.

IconGadget is intended for use with the Container, which lays out IconGadget children in various styles, in order to represent application objects of some kind. In addition to the textual labelling, an IconGadget can be associated with an array of detail information, which presumably represents attributes of the application object, and which the Container parent can lay out relative to the IconGadget.

By default, the IconGadget will use its widget name for the XmNlabelString resource. To display only an image, specify the appropriate pixmap resource, and set the XmNlabelString resource to XmUNSPECIFIED. Alternatively, apply an XmNlabelString resource which specifies a compound string with no text component.

Traits

IconGadget holds the XmQTcontainerItem, XmQTcareParentVisual, XmQTpointIn, and XmQTaccessColors traits, and uses the XmQTcontainer and XmQTspecifyRenderTable traits. The XmQTpointIn trait is undocumented.

New Resources

IconGadget defines the following resources:

Name	Class	Type	Default	Access
XmNalignment	XmCAlignment	unsigned char	XmALIGNMENT_CENTER	CSG
XmNdetail	XmCDetail	XmStringTable	NULL	CSG
XmNdetailCount	XmCDetailCount	Cardinal	0	CSG
XmNfontList	XmCFontList	XmFontList	NULL	CSG
XmNlabelString	XmCLabelString	XmString	dynamic	CSG
XmNlargeIconMask	XmCIconMask	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNlargeIconPixmap	XmCIconPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNsmallIconMask	XmCIconMask	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNsmallIconPixmap	XmCIconPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNspacing	XmCSpacing	Dimension	4	CSG
XmNviewType	XmCViewType	unsigned char	XmLARGE_ICON	CSG
XmNvisualEmphasis	XmCVisualEmphasis	unsigned char	XmNOT_SELECTED	CSG

XmNalignment
: In Motif 2.1, specifies the horizontal alignment of the textual and pixmap data. Possible values:

  XmALIGNMENT_BEGINNING
  XmALIGNMENT_CENTER
  XmALIGNMENT_END

XmNdetail
: Specifies an array of compound strings, representing the detail information associated with the IconGadget.

XmNdetailCount
: Specifies the number of compound strings in XmNdetail.

XmNfontList
: Specifies the font list associated with the IconGadget. In Motif 2.0 and later, the XmFontList is an obsolete data type, and has been replaced by the XmRenderTable. The resource is maintained for backwards compatibility but implemented as a render table. Any specified XmNrenderTable resource takes priority.

XmNlabelString
: The compound string representing the textual data for labelling the IconGadget. If unspecified, the value is constructed out of the name of the gadget.

XmNlargeIconMask
: Specifies the icon mask used when XmNviewType is XmLARGE_ICON. This resource must be a bitmap (a pixmap of depth 1).

XmNlargeIconPixmap
: Specifies the pixmap used when XmNviewType is XmLARGE_ICON.

XmNmarginHeight
: In Motif 2.1, specifies the vertical distance in pixels between the highlight rectangle and the IconGadget contents.

XmNmarginWidth
: In Motif 2.1, specifies the horizontal distance in pixels between the highlight rectangle and the IconGadget contents.

XmNrenderTable
: Specifies the XmRenderTable used for displaying textual data associated with the IconGadget. If unspecified, the value is taken from the nearest ancestor which holds the XmQTspecifyRenderTable trait, using the XmLABEL_RENDER_TABLE value of any ancestor so found.

XmNsmallIconMask
: Specifies the icon mask used when XmNviewType is XmSMALL_ICON. This resource must be a bitmap (a pixmap of depth 1).

XmNsmallIconPixmap
: Specifies the pixmap used when XmNviewType is XmSMALL_ICON.

XmNspacing
: In Motif 2.1, specifies the distance in pixels between the textual and pixmap components of the IconGadget.

XmNviewType
Specifies the IconGadget layout style. If the parent of the IconGadget is a Container, the view type is overridden by the value of the XmNentryViewType resource of the parent if the value is not XmANY_ICON. If XmNviewType is XmLARGE_ICON, the pixmap specified by XmNlargeIconPixmap is displayed above the textual label, with the text centered upon the pixmap. If XmNviewType is XmSMALL_ICON, the label is displayed either to the left or the right of the pixmap, depending upon the XmNlayoutDirection resource.

XmNvisualEmphasis
Specifies whether the IconGadget is displayed in normal or selected state.

If the value is XmSELECTED, the gadget is rendered using the XmNselectColor resource of the Container parent. XmNOT_SELECTED displays in normal state.

Inherited Resources

IconGadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them.

Resource	Inherited From	Resource	Inherited From
XmNancestorSensitive	RectObj	XmNhighlightThickness	XmGadget
XmNbackground	XmGadget	XmNlayoutDirection	XmGadget
XmNbackgroundPixmap	XmGadget	XmNnavigationType	XmGadget
XmNbottomShadowColor	XmGadget	XmNsensitive	RectObj
XmNbottomShadowPixmap	XmGadget	XmNshadowThickness	XmGadget
XmNborderWidth	RectObj	XmNtopShadowColor	XmGadget
XmNdestroyCallback	Object	XmNtopShadowPixmap	XmGadget
XmNforeground	XmGadget	XmNtraversalOn	XmGadget
XmNheight	RectObj	XmNunitType	XmGadget
XmNhelpCallback	XmGadget	XmNuserData	XmGadget
XmNhighlightColor	XmGadget	XmNwidth	RectObj
XmNhighlightOnEnter	XmGadget	XmNx	RectObj
XmNhighlightPixmap	XmGadget	XmNy	RectObj

See Also

`XtCreateObject(1), Object`(2), `RectObj`(2), `XmGadget`(2).

Name

XmInformationDialog – an unmanaged MessageBox as a child of a DialogShell.

Synopsis

Public Header:
<Xm/MessageB.h>

Instantiation:
widget = `XmCreateInformationDialog`(...)

Functions/Macros:
`XmCreateInformationDialog`(), `XmMessageBoxGetChild`()

Description

An XmInformationDialog is a compound object created by a call to XmCreateInformationDialog() that an application can use to provide the user with information. An InformationDialog consists of a DialogShell with an unmanaged MessageBox widget as its child. The MessageBox resource XmNdialogType is set to XmDIALOG_INFORMATION. An InformationDialog includes four components: a symbol, a message, three buttons, and a separator between the message and the buttons. By default, the symbol is a lowercase *i*. In Motif 1.2, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Cancel**, and **Help** by default.

Default Resource Values

An InformationDialog sets the following default values for MessageBox resources:

Name	Default
XmNdialogType	XmDIALOG_INFORMATION
XmNsymbolPixmap	xm_information[a]

a.Erroneously given as Xm_information in 2nd edition.

Widget Hierarchy

When an InformationDialog is created with a specified name, the DialogShell is named name*_popup* and the MessageBox is called *name***.**

See Also

`XmCreateObject`(1), `XmMessageBoxGetChild`(1), `XmDialogShell`(2), `XmMessageBox`(2).

Name

XmLabel widget class – a simple widget that displays a non-editable label.

Synopsis

Public Header:
<Xm/Label.h>

Class Name:
XmLabel

Class Hierarchy:
Core →XmPrimitive →XmLabel

Class Pointer:
xmLabelWidgetClass

Instantiation:
widget = `XmCreateLabel` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmLabelWidgetClass,...)

Functions/Macros:
`XmCreateLabel`(), `XmIsLabel`()

Description

Label provides a text string or a pixmap for labelling other widgets in an application. Label is also a superclass for the various button widgets. Label does not accept any button or key events, but it does receive enter and leave events.

Traits

Label holds the XmQTmenuSavvy, XmQTtransfer, and XmQTaccessTextual traits, which are inherited by any derived classes, and uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

New Resources

Label defines the following resources, where the access for every resource is CSG:

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

Name	Class	Type	Default	Access
XmNlabelString	XmCLabelString	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	XmFONTLIST_DEFAULT_TAG	CSG
XmNrecomputeSize	XmCRecomputeSize	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CSG

XmNaccelerator
A string that describes a button widget's accelerator (the modifiers and key to use as a shortcut in selecting the button). The string's format is like that of a translation but allows only a single key press event to be specified.

XmNacceleratorText
The text that is displayed for an accelerator.

XmNalignment
The alignment (left to right) for a label's text or pixmap. Possible values are XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, and XmALIGNMENT_END. In Motif 2.0 and later, the interpretation of alignment depends upon the value of any inherited XmNlayoutDirection resource.

XmNfontList
The font list used for the widget's text. From Motif 2.0 and later, the XmfontList is an obsolete data type, and the Rendition Table is the preferred method of setting appearance. Although maintained for backwards compatibility, the resource is implemented through a render table. Any XmNrenderTable resource takes precedence.

XmNlabelInsensitivePixmap
The pixmap label for an insensitive button (when XmNlabelType is XmPIXMAP).

XmNlabelPixmap
The pixmap used when XmNlabelType is XmPIXMAP.

XmNlabelString
The compound string used when XmNlabelType is XmSTRING. If this resource is NULL, the application uses the widget's name (converted to compound string format).

XmNlabelType
The type of label (either string or pixmap). Possible values:

```
XmPIXMAP    /* use XmNlabelPixmap or XmNlabelInsensitivePixmap */
XmSTRING    /* use XmNlabelString                              */
```

XmNmarginTop, XmNmarginBottom,

XmNmarginLeft, XmNmarginRight
The amount of space between one side of the label text and the nearest margin.

XmNmarginHeight, XmNmarginWidth
The spacing between one side of the label and the nearest edge of a shadow.

XmNmnemonic
A keysym that gives the user another way to select a button. In the label string, the first character matching this keysym will be underlined.

XmNmnemonicCharSet
The character set for the label's mnemonic.

XmNrecomputeSize
If True (default), the Label widget changes its size so that the string or pixmap fits exactly.

XmNrenderTable
In Motif 2.0 and later, specifies the render table for the Label. If NULL, this is inherited from the nearest ancestor that has the XmQTspecifyRenderTable trait, taking the XmLABEL_RENDER_TABLE value from the ancestor. The BulletinBoard, VendorShell, and MenuShell widgets and derived classes set this trait.

XmNstringDirection
In Motif 2.0 and later, XmNstringDirection is superseded by the inherited XmNlayoutDirection resource. The direction in which to draw the string. Possible values are:

```
XmSTRING_DIRECTION_L_TO_R
XmSTRING_DIRECTION_R_TO_L
XmDEFAULT_DIRECTION
```

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;     /* set to XmCR_HELP                    */
    XEvent      *event;     /* points to event that triggered callback */
} XmAnyCallbackStruct;
```

Inherited Resources

Label inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. Label sets the default values of XmNhighlightThickness and XmNshadowThickness to 0 and XmNtraversalOn to False. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNhighlightThickness	XmPrimitive
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNlayoutDirection	XmPrimitive
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnavigationType	XmPrimitive
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmPrimitive	XmNsensitive	Core
XmNbottomShadowPixmap	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNcolormap	Core	XmNtopShadowColor	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNdepth	Core	XmNtranslations	Core
XmNdestroyCallback	Core	XmNtraversalOn	XmPrimitive
XmNforeground	XmPrimitive	XmNunitType	XmPrimitive
XmNheight	Core	XmNuserData	XmPrimitive
XmNhelpCallback	XmPrimitive	XmNwidth	Core
XmNhighlightColor	XmPrimitive	XmNx	Core
XmNhighlightOnEnter	XmPrimitive	XmNy	Core
XmNhighlightPixmap	XmPrimitive		

Translations

Event	Action
BTransfer Press	ProcessDrag()
KHelp	Help()

For subclasses of Label:

Event	Action
KLeft	MenuTraverseLeft()
KRight	MenuTraverseRight()
KUp	MenuTraverseUp()
KDown	MenuTraverseDown()
MAny KCancel	MenuEscape()

Action Routines

Label defines the following action routines:

Help()

Unposts menus, restores keyboard focus, and invokes the callbacks from XmNhelpCallback, if there are any.

MenuEscape()

Unposts the menu, disarms the associated CascadeButton, and restores keyboard focus.

MenuTraverseDown()

In a MenuBar, if the current menu item has a submenu, posts the submenu, disarms the current menu item, and arms the first item in the submenu. In a menu pane, disarms the current menu item and arms the item below it, wrapping around to the top if necessary.

MenuTraverseLeft()

In a MenuBar, disarms the current menu item and arms the next item to the left, wrapping if necessary. In a menu pane, disarms the current item and arms the item to the left if there is such an item. Otherwise, unposts the current submenu and, if that submenu is attached to a MenuBar item, traverses to the MenuBar item to the left (wrapping if necessary), posts the submenu, and arms the first item in the submenu. In a PopupMenu or a torn-off menu pane, traverses to the menu item to the left, wrapping to the right if necessary.

MenuTraverseRight()

In a MenuBar, disarms the current menu item and arms the next item to the right, wrapping if necessary. In a menu pane, if the current item is a CascadeButton, posts the associated submenu. Otherwise, disarms the current item and arms the item to the right if there is such an item or unposts all submenus, traverses to the MenuBar

item to the right (wrapping if necessary), posts the submenu, and arms the first item in the submenu. In a PopupMenu or a torn-off menu pane, traverses to the menu item to the right, wrapping to the left if necessary.

MenuTraverseUp()
: In a menu pane, disarms the current menu item and arms the item above it, wrapping around to the bottom if necessary.

ProcessDrag()
: In Motif 1.2, initiates a drag and drop operation using the contents of the Label. In Motif 2.0 and later, this indirectly invokes any procedures specified by the inherited XmNconvertCallback resource.

See Also

`XmCreateObject`(1), `Core`(2), `XmPrimitive`(2).

Name

XmLabelGadget widget class – a simple gadget that displays a non-editable label.

Synopsis

Public Header:
<Xm/LabelG.h>

Class Name:
XmLabelGadget

Class Hierarchy:
Object →RectObj →XmGadget →XmLabelGadget

Class Pointer:
xmLabelGadgetClass

Instantiation:
widget = `XmCreateLabelGadget` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmLabelGadgetClass,...)

Functions/Macros:
`XmCreateLabelGadget`(), `XmOptionLabelGadget`(), `XmIsLabelGadget`()

Description

LabelGadget is the gadget variant of Label.

In Motif 2.0, the LabelGadget cached resource set is expanded to include XmNforeground, XmNbackground, XmNtopShadowColor, XmNtopShadowPixmap, XmNbottomShadowColor, XmNbottomShadowPixmap, XmNhighlightColor and XmNhighlightPixmap resources. LabelGadgets sharing the same Manager parent can therefore have independent appearance.

Traits

LabelGadget holds the XmQTmenuSavvy, XmQTtransfer, XmQTaccessTextual, XmQTcareParentVisual, and XmQTaccessColors traits, which are inherited by any derived classes, and uses the traits XmQTmenuSystem and XmQTspecifyRenderTable.

New Resources

LabelGadget's new resources, callback resources, and callback structure are the same as those for Label.

Inherited Resources

LabelGadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. LabelGadget sets the default values of XmNhighlightThickness and XmNshadowThickness to 0 (zero) and XmNtraversalOn to False. The default value of XmNborderWidth is reset to 0 by Gadget.

Resource	Inherited From	Resource	Inherited From
XmNancestorSensitive	RectObj	XmNhighlightThickness	XmGadget
XmNbackground	XmGadget	XmNlayoutDirection	XmGadget
XmNbackgroundPixmap	XmGadget	XmNnavigationType	XmGadget
XmNbottomShadowColor	XmGadget	XmNsensitive	RectObj
XmNbottomShadowPixmap	XmGadget	XmNshadowThickness	XmGadget
XmNborderWidth	RectObj	XmNtopShadowColor	XmGadget
XmNdestroyCallback	Object	XmNtopShadowPixmap	XmGadget
XmNforeground	XmGadget	XmNtraversalOn	XmGadget
XmNheight	RectObj	XmNunitType	XmGadget
XmNhelpCallback	XmGadget	XmNuserData	XmGadget
XmNhighlightColor	XmGadget	XmNwidth	RectObj
XmNhighlightOnEnter	XmGadget	XmNx	RectObj
XmNhighlightPixmap	XmGadget	XmNy	RectObj

Behavior

As a Gadget subclass, LabelGadget has no translations associated with it. However, LabelGadget behavior corresponds to the action routines of the Label widget. See the Label action routines for more information.

Behavior	Equivalent Label Action
BTransfer Press	ProcessDrag()
KHelp	Help()

For subclasses of LabelGadget:

Behavior	Equivalent Label Action
KLeft	MenuTraverseLeft()
KRight	MenuTraverseRight()
KUp	MenuTraverseUp()

Behavior	Equivalent Label Action
KDown	MenuTraverseDown()
MAny KCancel	MenuEscape()

See Also

XmCreateObject(1), XmOptionLabelGadget(1), Object(2), RectObj(2), XmGadget(2), XmLabel(2).

Name

XmList widget class – a widget that allows a user to select from a list of choices.

Synopsis

Public Header:
<Xm/List.h>

Class Name:
XmList

Class Hierarchy:
Core →XmPrimitive →XmList

Class Pointer:
xmListWidgetClass

Instantiation:
widget = `XmCreateList` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmListWidgetClass,...)

Functions/Macros:
`XmCreateList`(), `XmCreateScrolledList`(), `XmList`... routines, `XmIsList`()

Description

List provides a list of choices from which a user can select one or more items, based on the selection policy. List supports four selection policies: single select, browse select, multiple select, and extended select.

In single select mode, only one item can be selected at a time; a button press on an item selects it and deselects the previously selected item. In browse select mode, only one item can be selected at a time; a button press works as in single select mode and, additionally, a button drag moves the selection with the pointer. In multiple select mode, any number of items can be selected at a time; a button press toggles the selection state of an item and does not change the selection state of any other items. In extended select mode, any number of items can be selected at a time; discontinuous ranges of items can be selected by combining button presses and button drags.

Selections can be made by using either the pointer or the keyboard. Keyboard selection has two modes: normal mode and add mode. In normal mode, keyboard navigation operations affect the selection; the item with the keyboard focus is always selected. In add mode, keyboard navigation operations are distinct from selection operations; the item with the keyboard focus can be disjoint from the selection. Browse select operates in normal mode; single select and multiple select operate in add mode; extended select can be made to operate in either

mode. Normal mode uses a solid location cursor while add mode uses a dashed location cursor.

In Motif 1.2 and later, List is a supported drag source for drag and drop operations. BTransfer Press starts a drag and drop operation using the selected items in the List. If BTransfer is pressed over an unselected item, that item is dragged instead of the selected items.

Traits

List holds the XmQTtransfer trait, which is inherited by any derived classes, and uses the XmQTnavigator, XmQTscrollFrame and XmQTspecifyRenderTable traits.

New Resources

List defines the following resources:

Name	Class	Type	Default	Access
XmNautomaticSelection	XmCAutomaticSelection	XtEnum	XmNO_AUTO_SELECT	CSG
XmNdoubleClickInterval	XmCDoubleClickInterval	int	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNhorizontalScrollBar	XmCHorizontalScrollBar	Widget	NULL	G
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	CSG
XmNlistSpacing	XmCListSpacing	Dimension	0	CSG
XmNmatchBehavior	XmCMatchBehavior	unsigned char	XmQUICK_NAVIGATE	CSG
XmNprimaryOwnership	XmCPrimaryOwnership	unsigned char	XmOWN_NEVER	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNscrollBarDisplayPolicy	XmCScrollBarDisplayPolicy	unsigned char	XmAS_NEEDED	CSG
XmNselectColor	XmCSelectColor	Pixel	dynamic	CSG
XmNselectedItemCount	XmCSelectedItemCount	int	0	CSG
XmNselectedItems	XmCSelectedItems	XmStringTable	NULL	CSG
XmNselectedPositionCount	XmCSelectedPositionCount	int	0	CSG
XmNselectedPositions	XmCSelectedPositions	int *	NULL	CSG
XmNselectionMode	XmCSelectionMode	unsigned char	XmNORMAL_MODE	CSG
XmNselectionPolicy	XmCSelectionPolicy	unsigned char	XmBROWSE_SELECT	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CSG

Name	Class	Type	Default	Access
XmNtopItemPosition	XmCTopItemPosition	int	1	CSG
XmNverticalScrollBar	XmCVerticalScrollBar	Widget	NULL	G
XmNvisibleItemCount	XmCVisibleItemCount	int	dynamic	CSG

XmNautomaticSelection
If True (and the widget's XmNselectionPolicy is either XmBROWSE_SELECT or XmEXTENDED_SELECT), then this resource calls XmNsingleSelection-Callback whenever the user moves into a new item.

If False, then the user must release the mouse button before any selection callbacks are called.

From Motif 2.0, the resource has changed type from a Boolean to an enumeration. Possible values of the enumerated type:

```
XmAUTO_SELECT       /* enables automatic selection */
XmNO_AUTO_SELECT    /* disables automatic selection */
```

XmNdoubleClickInterval
The time span (in milliseconds) within which two button clicks must occur to be considered a double click rather than two single clicks. By default, this value is the multiclick time of the display.

XmNfontList
The font list used for the widget's items. From Motif 2.0 and later, the XmfontList is considered obsolete, and the Rendition Table is the preferred method of setting appearance. Any XmNrenderTable value will take precedence.

XmNhorizontalScrollBar
When the List is part of a ScrolledList, specifies the widget ID of the ScrollBar created by the List to perform horizontal scrolling.

XmNitemCount
The total number of items. The widget updates this resource every time a list item is added or removed.

XmNitems
A pointer to an array of compound strings. The compound strings are the list items to display. A call to XtGetValues() returns the actual list items (not a copy), so don't have your application free these items.

XmNlistMarginHeight

XmNlistMarginWidth
The height or width of the margin between the border of the List and the items in the list.

XmNlistSizePolicy
The method for resizing the widget when a list item exceeds the width of the work area. This resizing policy must be set at creation time. Possible values:

```
XmVARIABLE          /* grow to fit; don't add ScrollBar          */
XmCONSTANT          /* don't grow to fit; add ScrollBar          */
XmRESIZE_IF_POSSIBLE /* grow or shrink; add ScrollBar if too large */
```

XmNlistSpacing
The spacing between items.

XmNmatchBehavior
In Motif 2.0 and later, specifies whether the widget navigates to items within the List by matching keyboard input against the first character of each item. If the value is XmNONE, no matching is performed. If the value is XmQUICK_NAVIGATE, any characters typed when the List has the focus are compared against the first character of each item. If a match is found, the matched list item is automatically made the current item. Subsequently typing the same character progresses cyclically through the list to find any further matching item.

XmNprimaryOwnership
Specifies how the list interacts with the primary selection when a user selects an item from the list. Possible values:

```
XmOWN_NEVER          /* never take ownership of the primary selection */
XmOWN_ALWAYS         /* always take ownership of the selection        */
XmOWN_MULTIPLE       /* take ownership if more than one item selected */
XmOWN_POSSIBLE_MULTIPLE  /* take ownership if selection policy        */
                         /* is multiple or extended                   */
```

XmNrenderTable
In Motif 2.0 and later, specifies the render table for the list. If unspecified, the value of the resource is inherited from the nearest ancestor which holds the XmQTspecifyRenderTable trait, using the XmTEXT_RENDER_TABLE value of the ancestor so found. This resource, together with the XmNvisibleItem-Count resource, is used to calculate the List widget's height.

XmNscrollBarDisplayPolicy
Determines when to display vertical scrollbars in a ScrolledList widget. Possible values:

```
XmSTATIC             /* vertical ScrollBar always displays   */
XmAS_NEEDED          /* add ScrollBar when list is too large */
```

XmNselectColor
: In Motif 2.0 and later, specifies the color used to draw the background of selected items. In addition to allocated Pixel values, the constant XmDEFAULT_SELECT_COLOR specifies a color between the XmNbackground and XmNbottomShadowColor, XmHIGHLIGHT_COLOR makes the select color the same as the XmNhighlightColor value, and XmREVERSED_GROUND_COLORS makes the XmNselectColor the same as the XmNforeground, using the XmNbackground color to render any text.

XmNselectedItemCount
: The number of items in the list of selected items.

XmNselectedItems
: A pointer to an array of compound strings. The compound strings represent the currently selected list items. A call to `XtGetValues`() returns the actual list items (not a copy), so don't have your application free these items.

XmNselectedPositionCount
: In Motif 2.0 and later, specifies the number of positions in the list of selected positions.

XmNselectedPositions
: In Motif 2.0 and later, specifies a pointer to an array of integers, representing the currently selected list positions. A call to `XtGetValues`() returns the actual list position array, so don't free the array in application code. Compare with `XmListGetSelectedPos`() which returns a copy of the selected position array which should be freed after use.

XmNselectionMode
: In Motif 2.0 and later, specifies the effect which keyboard navigation has upon selection. If the value is XmNORMAL_MODE, navigation operations can select the item under the location cursor, deselecting any other items. In XmADD_MODE, navigation operations have no effect on selection. For XmNORMAL_MODE, the selection policy must be browse or extended selection, and for XmADD_MODE, the policy must not be browse.

XmNselectionPolicy
: Determines the effect of a selection action. Possible values:

    XmSINGLE_SELECT
    XmBROWSE_SELECT
    XmMULTIPLE_SELECT
    XmEXTENDED_SELECT

XmNstringDirection
The direction in which to draw the string. Possible values are:

XmSTRING_DIRECTION_L_TO_R
XmSTRING_DIRECTION_R_TO_L
XmDEFAULT_DIRECTION

In Motif 2.0 and later, the XmNstringDirection resource is obsolete, being subsumed into the XmNlayoutDirection resource. If the XmNlayoutDirection is NULL, and the XmNstringDirection is XmDEFAULT_DIRECTION, the value will be taken from the nearest ancestor which holds the XmQTspecifyLayoutDirection trait. Manager, MenuShell, and VendorShell support this trait.

XmNtopItemPosition
The position of the first item that will be visible in the list. Calling the `XmListSetPos`() routine is the same as setting this resource. In both cases, the first position is specified as 1 and the last position is specified as 0.

XmNverticalScrollBar
When the List is part of a ScrolledList, specifies the widget ID of the ScrollBar created by the List to perform vertical scrolling.

XmNvisibleItemCount
The number of items to display in the work area of the list. This value affects the widget's height. In Motif 1.2 and later, the default value of this resource is dynamic and based on the height of the List, while in Motif 1.1, the default value is 1.

Callback Resources

List defines the following callback resources:

Callback	Reason Constant
XmNbrowseSelectionCallback	XmCR_BROWSE_SELECT
XmNdefaultActionCallback	XmCR_DEFAULT_ACTION
XmNdestinationCallback	XmCR_OK
XmNextendedSelectionCallback	XmCR_EXTENDED_SELECT
XmNmultipleSelectionCallback	XmCR_MULTIPLE_SELECT
XmNsingleSelectionCallback	XmCR_SINGLE_SELECT

XmNbrowseSelectionCallback
List of callbacks that are called when a list item is selected using the browse selection policy.

XmNdefaultActionCallback
List of callbacks that are called when a list item is double clicked or KActivate is pressed.

XmNdestinationCallback
List of callbacks that are called when the List is the destination of a transfer operation.

XmNextendedSelectionCallback
List of callbacks that are called when list items are selected using the extended selection policy.

XmNmultipleSelectionCallback
List of callbacks that are called when a list item is selected using the multiple selection policy.

XmNsingleSelectionCallback
List of callbacks that are called when a list item is selected using the single selection policy.

Callback Structure

Each selection callback function is passed the structure below; however, some structure members might be unused because they aren't meaningful for particular callback reasons.

```
typedef struct {
        int         reason;                 /* reason that callback was called   */
        XEvent      *event;                 /* event that triggered callback     */
        XmString    item;                   /* item most recently selected at    */
                                            /* the time event occurred           */
        int         item_length;            /* number of bytes in item member    */
        int         item_position;          /* item's position in XmNitems array */
        XmString    *selected_items;        /* list of items selected at time    */
                                            /* event occurred                    */
        int         selected_item_count;    /* number of items in selected_items */
        int         *selected_item_positions; /* array of integers that mark     */
                                            /* selected items                    */
        char        selection_type;         /* type of the most recent selection */
        char        auto_selection_type;    /* the type of automatic selection   */
} XmListCallbackStruct;
```

The structure members *event*, *item*, *item_length*, and *item_position* are valid for any value of *reason*.

The structure members *selected_items*, *selected_item_count*, and *selected_item_positions* are valid when the *reason* field has a value of XmCR_MULTIPLE_SELECT or XmCR_EXTENDED_SELECT.

The structure member *selection_type* is valid only when the *reason* field is XmCR_EXTENDED_SELECT.

The structure member *auto_selection_type* is valid only when the resource XmNautomaticSelection is XmAUTO_SELECT, and has the value XmAUTO_UNSET otherwise.

For the strings pointed to by *item* and *selected_items*, as well as for the integers pointed to by *selected_item_positions*, storage is overwritten each time the callback is invoked. Applications that need to save this data should make their own copies of it.

selected_item_positions is an integer array. The elements of the array indicate the positions of each selected item within the List widget's XmNitems array.

selection_type specifies what kind of extended selection was most recently made. One of three values is possible:

```
XmINITIAL       /* selection was the initial selection          */
XmMODIFICATION  /* selection changed an existing selection      */
XmADDITION      /* selection added non-adjacent items to       */
                /*          existing selection */
```

auto_selection_type specifies at what point within the selection the user is. Possible values:

```
XmAUTO_UNSET
XmAUTO_BEGIN
XmAUTO_MOTION
XmAUTO_CANCEL
XmAUTO_NO_CHANGE
XmAUTO_CHANGE
```

Destination callbacks are fully described within the sections covering the Uniform Transfer Model. See `XmTransfer`(s1) for more details. For quick reference, a pointer to the following structure is passed to callbacks on the XmNdestinationCallback list*:*

```
typedef struct {
    int         reason;         /* reason that the callback is invoked    */
    XEvent      *event;         /* event that triggered callback          */
    Atom        selection;      /* requested selection type, as an Atom   */
    XtEnum      operation;      /* type of transfer requested             */
    int         flags;          /* whether destination and source are same */
```

```
    XtPointer   transfer_id;      /* unique identifier for the request      */
    XtPointer   destination_data; /* information about the destination      */
    XtPointer   location_data;    /* information about the data             */
    Time        time;             /* time when transfer operation started   */
} XmDestinationCallbackStruct;
```

Inherited Resources

List inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. List sets the default value of XmNnavigationType to XmTAB_GROUP, and sets the default value of XmNlayoutDirection. XmDEFAULT_DIRECTION. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNhighlightThickness	XmPrimitive
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNlayoutDirection	XmPrimitive
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnavigationType	XmPrimitive
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmPrimitive	XmNsensitive	Core
XmNbottomShadowPixmap	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNcolormap	Core	XmNtopShadowColor	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNdepth	Core	XmNtranslations	Core
XmNdestroyCallback	Core	XmNtraversalOn	XmPrimitive
XmNforeground	XmPrimitive	XmNunitType	XmPrimitive
XmNheight	Core	XmNuserData	XmPrimitive
XmNhelpCallback	XmPrimitive	XmNwidth	Core
XmNhighlightColor	XmPrimitive	XmNx	Core
XmNhighlightOnEnter	XmPrimitive	XmNy	Core
XmNhighlightPixmap	XmPrimitive		

Translations

Event	Action	Event	Action
BSelect Press	ListBeginSelect()	KPageRight	ListRightPage()
BSelect Motion	ListButtonMotion()	KAny	ListQuickNavigate()
BSelect Release	ListEndSelect()	KBeginLine	ListBeginLine()
BExtend Press	ListBeginExtend()	KEndLine	ListEndLine()
BExtend Motion	ListButtonMotion()	KBeginData	ListBeginData()
BExtend Release	ListEndExtend()	MShift KBeginData	ListBeginDataExtend()
BToggle Press	ListBeginToggle()	KEndData	ListEndData()
BToggleMotion	ListButtonMotion()	MShift KEndData	ListEndDataExtend()
BToggle Release	ListEndToggle()	KAddMode	ListAddMode()
BTransfer Press	ListProcessDrag()	KActivate	ListKbdActivate()
KUp	ListPrevItem()	KCopy Press	ListCopyToClipboard()
MShift KUp	ListExtendPrevItem()	KSelectPress	ListKbdBeginSelect()
KDown	ListNextItem()	KSelect Release	ListKbdEndSelect()
MShift KDown	ListExtendNextItem()	KExtend Press	ListKbdBeginExtend()
KLeft	ListLeftChar()	KExtend Release	ListKbdEndExtend()
MCtrl KLeft	ListLeftPage()	Many KCancel	ListKbdCancel()
KRight	ListRightChar	KSelectAll	ListKbdSelectAll()
MCtrl KRight	ListRightPage()	KDeselectAll	ListKbdDeSelectAll()
KPageUp	ListPrevPage	KHelp	PrimitiveHelp()
KPageDown	ListNextPage()	KNextField	PrimitiveNextTabGroup()
KPageLeft	ListLeftPage()	KPrevField	PrimitivePrevTabGroup()

In addition, the translations of List are modified when the XmNenableBtn1Transfer() resource of the XmDisplay object is not XmOFF. The following translations apply under these circumstances:

Event	Action
BSelect Press	ListProcessBtn1(ListBeginSelect)
BSelect Motion	ListProcessBtn1(ListButtonMotion)
BSelect Release	ListProcessBtn1(ListEndSelect)

Event	Action
BExtend Press	ListProcessBtn1(ListBeginExtend)
BExtend Release	ListProcessBtn1(ListEndExtend)
BToggle Press	ListProcessBtn1(ListBeginToggle)
BToggle Release	ListProcessBtn1(ListEndToggle)
BTransfer Press	ListProcessBtn2(ListBeginExtend)
BTransfer Motion	ListProcessBtn2(ListButtonMotion)
BTransfer Release	ListProcessBtn2(ListEndExtend)

Action Routines

List defines the action routines below. The current selection always appears with its foreground and background colors reversed. Note that many List actions have different effects depending on the selection policy and also that some actions apply only for a particular selection policy.

ListAddMode()

Turns add mode on or off.

ListBeginData()

Moves the cursor to the first list item. If keyboard selection is in normal mode, this action also selects the first item after deselecting any earlier selection and invokes the callbacks specified either by XmNbrowseSelectionCallback or by XmNextendedSelectionCallback (as dictated by the selection policy).

ListBeginDataExtend()

Multiple selection: moves the cursor to the first list item.

Extended selection: moves the cursor to the first list item, cancels any current extended selection, selects (or deselects) all items from the first item to the current anchor, and invokes the callbacks specified by XmNextendedSelectionCallback.

ListBeginExtend()

Extended selection: cancels any current extended selection, selects (or deselects) all items from the pointer location to the current anchor, and invokes the callbacks specified by XmNextendedSelectionCallback *(if the XmNautomaticSelection* resource is True*).*

ListBeginLine()

Scrolls the List's viewing area horizontally to its beginning.

ListBeginSelect()

Single selection: selects or deselects the item under the pointer after deselecting any previous selection.

Browse selection: selects the item under the pointer after deselecting any previous selection and invokes the callbacks specified by XmNbrowseSelectionCallback *if the XmNautomaticSelection* resource is True.

Multiple selection: selects or deselects the item under the pointer, leaving previous selections unaffected.

Extended selection: selects the item under the pointer after deselecting any previous selection, marks this item as the current anchor, and invokes the callbacks specified by XmNextendedSelectionCallback *if the XmNautomaticSelection* resource is True.

ListBeginToggle()

Extended selection: keeps the current selection but shifts the anchor to the item under the pointer. This item's selection state is toggled, and if XmNautomaticSelection *is True*, the extended selection callbacks are invoked.

ListButtonMotion()

Browse selection: selects the item under the pointer after deselecting any previous selection and invokes the browse selection callbacks if XmNautomaticSelection *is True* and the pointer moved over a new item.

Extended selection: cancels any current extended selection, selects (or deselects) all items from the pointer location to the current anchor, and invokes the extended selection callbacks if XmNautomaticSelection *is True* and the pointer moved over a new item.

In addition, when the pointer moves outside a ScrolledList widget, the list scrolls in sync with the pointer motion.

ListCopyToClipboard()

In Motif 1.2 and later, this action copies the selected list items to the clipboard. The items are copied as a single compound string, with a new line between each item.

ListEndData()

Moves the cursor to the last list item. If keyboard selection is in normal mode, this action also selects the last item after deselecting any earlier selection and invokes the appropriate callbacks (browse selection or extended selection).

ListEndDataExtend()

Multiple selection: moves the cursor to the last list item.

Extended selection: moves the cursor to the last list item, cancels any current extended selection, selects (or deselects) all items from the last item to the current anchor, and invokes the extended selection callbacks.

ListEndExtend()

Extended selection: moves the cursor to the last item whose selection state was switched, and invokes the extended selection callbacks if XmNautomaticSelection is *False*.

ListEndLine()

Scrolls the List's viewing area horizontally to its beginning.

ListEndSelect()

Single selection or multiple selection: moves the cursor to the last item whose selection state was switched, and invokes the appropriate selection callbacks.

Browse selection or extended selection: same as above, except that the appropriate callbacks are called only if XmNautomaticSelection *is False*.

ListEndToggle()

Extended selection: moves the cursor to the last item whose selection state was switched, and invokes the extended selection callbacks if XmNautomaticSelection *is False*.

ListExtendNextItem()
ListExtendPrevItem()

Extended selection: adds the next/previous item to an extended selection and invokes the extended selection callbacks.

ListKbdActivate()

Invokes the default action callbacks.

ListKbdBeginExtend()

This action is the keyboard's complement to the mouse-activated ListBeginExtend() action.

Extended selection: cancels any current extended selection and selects (or deselects) all items from the cursor to the current anchor.

ListKbdBeginSelect()

This action is the keyboard's complement to the mouse-activated ListBeginSelect*()* action.

Single selection: selects or deselects the item at the cursor after deselecting any previous selection.

Browse selection: selects the item at the cursor after deselecting any previous selection and invokes the browse selection callbacks if XmNautomaticSelection *is True*.

Multiple selection: selects or deselects the item at the cursor, leaving previous selections unaffected.

Extended selection: shifts the anchor to the item at the cursor. In normal mode, this item is selected after any previous selection is deselected; in add mode, this item's state is toggled, and the current selection remains unaffected. This action calls the extended selection callbacks if XmNautomaticSelection *is True*.

ListKbdCancel()

Extended selection: cancels an extended selection and restores the items to their previous selection state.

ListKbdDeSelectAll()

Deselects all list items and calls the appropriate selection callbacks. This action applies to all selection modes except browse selection because this mode requires one item to remain selected at all times. In extended selection with keyboard Normal Mode and an XmNkeyboardFocusPolicy of XmEXPLICIT, the item at the cursor remains selected after this action is applied.

ListKbdEndExtend()

Extended selection: calls the extended selection callbacks if XmNautomaticSelection *is False*.

ListKbdEndSelect()

Single selection or multiple selection: calls the appropriate selection callbacks. If XmNautomaticSelection *is False*, this action applies under any of the four selection policies.

ListKbdSelectAll()

Single selection or browse selection: selects the item at the cursor and calls the appropriate selection callbacks.

Multiple selection or extended selection: selects all list items and calls the appropriate selection callbacks.

ListLeftChar()
ListLeftPage()

Scrolls the list either one character or one page to the left.

ListNextItem()

Moves the cursor to the next list item and has the following additional operations:

Browse selection: selects this item, deselects any previously selected item(s), and calls the browse selection callbacks.

Extended selection: in normal mode, selects this item and moves the anchor there, deselects any previously selected item(s), and calls the extended selection callbacks. In add mode, neither the selection nor the anchor is affected.

ListNextPage()

Moves the cursor by scrolling the list to the list item at the top of the next page and has the same additional operations as ListNextItem().

ListPrevItem()

Same as *ListNextItem()*, going back one item instead.

ListPrevPage() *Same as ListNextPage()*, going back one page instead.

ListProcessDrag()

In Motif 1.2 and later, this action initiates a drag and drop operation using the selected items, where each item is separated by a newline. If BTransfer is pressed over an unselected item, only that item is used in the drag and drop operation.

ListProcessBtn1(string*)*

In Motif 2.0 and later, the XmDisplay resource XmNenableBtn1Transfer configures the integration of selection and transfer operations on Button 1.

If XmNenableBtn1Transfer is XmOFF, if no data transfer has been initialised, the action specified by string is invoked to initiate selection. Possible values for string are:

ListBeginExtend
ListEndExtend
ListButtonMotion
ListBeginSelect
ListEndSelect
ListBeginToggle
ListEndToggle

ListProcessBtn2(string)
In Motif 2.0 and later, if the XmDisplay resource XmNenableBtn1Transfer has the value XmBUTTON2_TRANSFER, the actions for extending List selection are bound to Button 2, and data transfer is initiated. If the resource has the value XmBUTTON2_ADJUST, the action specified by string is invoked to extend selection. Possible values for string are:

ListBeginExtend
ListEndExtend
ListButtonMotion

ListQuickNavigate()
In Motif 2.0 and later, navigates to an item if XmNmatchBehavior is XmQUICK_NAVIGATE.

ListRightChar()
ListRightPage() Scrolls the list either one character or one page to the right.

ListScrollCursorVertically()
Makes the item with the keyboard focus visible.

PrimitiveHelp()
Calls the help callbacks for this widget.

PrimitiveNextTabGroup()
PrimitivePrevTabGroup()
Moves the keyboard focus to the beginning of the next or previous tab group, wrapping around if necessary.

Additional Behavior

List has the following additional behavior:

<Double Click>
: Calls the XmNdefaultActionCallback callbacks.

<FocusIn>
: If the keyboard focus policy is explicit, sets the focus and draws the location cursor.

<FocusOut>
: If the keyboard focus policy is explicit, removes the focus and erases the location cursor.

See Also

XmCreateObject(s1), XmListAddItem(s1), XmListAddItemUnselected(s1), XmListDeleteAllItems(s1), XmListDeleteItem(s1), XmListDeleteItemsPos(s1), XmListDeletePos(s1), XmListDeletePositions(s1), XmListDeselectAllItems(s1), XmListDeselectItem(s1), XmListDeselectPos(s1), XmListGetKbdItemPos(s1), XmListGetMatchPos(s1), XmListGetSelectedPos(s1), XmListItemExists(s1), XmListItemPos(s1), XmListPosSelected(s1), XmListPosToBounds(s1), XmListReplaceItems(s1), XmListReplaceItemsPos(s1), XmListReplaceItemsPosUnselected(s1), XmListReplaceItemsUnselected(s1), XmListReplacePositions(1), XmListSelectItem(1), XmListSelectPos(1), XmListSetAddMode(1), XmListSetBottomItem(1), XmListSetBottomPos(1), XmListSetHorizPos(1), XmListSetItem(1), XmListSetKbdItemPos(1), XmListSetPos(1), XmListUpdateSelectedList(1), XmListYToPos(1), Core(2), XmDisplay(2). XmPrimitive(2), XmTransfer(1).

Name

XmMainWindow widget class – the standard layout widget for an application's primary window.

Synopsis

Public Header:
<Xm/MainW.h>

Class Name:
XmMainWindow

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmScrolledWindow → XmMainWindow

Class Pointer:
xmMainWindowWidgetClass

Instantiation:
widget = `XmCreateMainWindow` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmMainWindowWidgetClass,...)

Functions/Macros:
`XmCreateMainWindow`(), `XmMainWindowSep1`(), `XmMainWindowSep2`(), `XmMainWindowSep3`(), `XmMainWindowSetAreas`(), `XmIsMainWindow`()

Description

MainWindow provides the standard appearance for the primary window of an application. MainWindow supports five standard areas: a MenuBar, a command window, a work region, a message window, and two ScrollBars (one horizontal and one vertical). An application can use as many or as few of these areas as necessary; they are all optional. A MainWindow can also display three Separator widgets for dividing one area from another.

Each of the MainWindow regions is associated with a MainWindow resource; `XmMainWindowSetAreas`() sets the associated resources. If an application does not call `XmMainWindowSetAreas`(), the widget may still set some of the standard regions. When a MenuBar child is added to a MainWindow, if XmNmenuBar has not been set, it is set to the MenuBar child. When a Command child is added to a MainWindow, if XmNcommand has not been set, it is set to the Command child. If ScrollBars are added as children, the XmNhorizontalScrollBar and XmNverticalScrollBar resources may be set if they have not already been specified. Any child that is not one of these types is used for the XmNworkWindow. If you want to be certain about which widgets are used for the different regions, it is wise to call `XmMainWindowSetAreas`() explicitly.

Traits

MainWindow uses the XmQTmenuSystem trait.

New Resources

MainWindow defines the following resources:

Name	Class	Type	Default	Access
XmNcommandWindow	XmCCommandWindow	Widget	NULL	CSG
XmNcommandWindowLocation	XmCCommandWindowLocation	unsigned char	XmCOMMAND_ABOVE_WORKSPACE	CG
XmNmainWindowMarginHeight	XmCMainWindowMarginHeight	Dimension	0	CSG
XmNmainWindowMarginWidth	XmCMainWindowMarginWidth	Dimension	0	CSG
XmNmenuBar	XmCMenuBar	Widget	NULL	CSG
XmNmessageWindow	XmCMessageWindow	Widget	NULL	CSG
XmNshowSeparator	XmCShowSeparator	Boolean	False	CSG

XmNcommandWindow
The widget ID of the command window child.

XmNcommandWindowLocation
One of two positions for the command window. Possible values:

```
XmCOMMAND_ABOVE_WORKSPACE   /* default; appears below menu bar */
XmCOMMAND_BELOW_WORKSPACE   /* appears between work and      */
                            /*     message windows           */
```

XmNmainWindowMarginHeight
The margin on the top and bottom of the MainWindow widget. This resource overrides the corresponding margin resource in the ScrolledWindow widget.

XmNmainWindowMarginWidth
The margin on the right and left of the MainWindow widget. This resource overrides the corresponding margin resource in the ScrolledWindow widget.

XmNmenuBar
The widget ID of the menu bar child.

XmNmessageWindow
The widget ID of the message window child.

XmNshowSeparator
If True, separators are displayed between components of the MainWindow widget. If False (default), separators are not displayed.

Inherited Resources

MainWindow inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of XmNborderWidth is reset to 0 by XmManager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNnavigationType	XmManager
XmNancestorSensitive	Core	XmNnumChildren	Composite
XmNautoDragModel	XmScrolledWindow	XmNpopupHandlerCallback	XmManager
XmNbackground	Core	XmNscreen	Core
XmNbackgroundPixmap	Core	XmNscrollBarDisplayPolicy	XmScrolledWindow
XmNborderColor	Core	XmNscrollBarPlacement	XmScrolledWindow
XmNborderPixmap	Core	XmNscrolledWindowMarginHeight	XmScrolledWindow
XmNborderWidth	Core	XmNscrolledWindowMarginWidth	XmScrolledWindow
XmNbottomShadowColor	XmManager	XmNscrollingPolicy	XmScrolledWindow
XmNbottomShadowPixmap	XmManager	XmNsensitive	Core
XmNchildren	Composite	XmNshadowThickness	XmManager
XmNclipWindow	XmScrolledWindow	XmNspacing	XmScrolledWindow
XmNcolormap	Core	XmNstringDirection	XmManager
XmNdepth	Core	XmNtopShadowColor	XmManager
XmNdestroyCallback	Core	XmNtopShadowPixmap	XmManager
XmNforeground	XmManager	XmNtranslations	Core
XmNheight	Core	XmNtraversalOn	XmManager
XmNhelpCallback	XmManager	XmNtraverseObscuredCallback	XmScrolledWindow
XmNhighlightColor	XmManager	XmNunitType	XmManager
XmNhighlightPixmap	XmManager	XmNuserData	XmManager
XmNhorizontalScrollBar	XmScrolledWindow	XmNverticalScrollBar	XmScrolledWindow
XmNinitialFocus	XmManager	XmNvisualPolicy	XmScrolledWindow
XmNinitialResourcesPersistent	Core	XmNwidth	Core
XmNinsertPosition	Composite	XmNworkWindow	XmScrolledWindow
XmNlayoutDirection	XmManager	XmNx	Core
XmNmappedWhenManaged	Core	XmNy	Core

Translations

The translations for MainWindow are inherited from ScrolledWindow.

See Also

XmCreateObject(1), XmMainWindowSep(1), XmMainWindowSetAreas(1), Composite(2), Constraint(2), Core(2), XmManager(2), XmScrolledWindow(2).

Name

XmManager widget class – the fundamental class for Motif widgets that manage children.

Synopsis

Public Header:
<Xm/Xm.h>

Class Name:
XmManager

Class Hierarchy:
Core →Composite →Constraint →XmManager

Class Pointer:
xmManagerWidgetClass

Instantiation:
Manager is a meta-class and is not normally instantiated.

Functions/Macros:
`XmIsManager`()

Description

Manager is a superclass for Motif widget classes that contain children. Manager supports geometry management by providing resources for visual shadows and highlights and for keyboard traversal mechanisms.

The default values of the color resources for the foreground, background, top and bottom shadows, and highlighting are set dynamically. If no colors are specified, they are generated automatically. On a monochrome system, black and white colors are selected. On a color system, four colors are selected that provide the appropriate shading for the 3-D visuals. When the background color is specified, the shadow colors are selected to provide the appropriate 3-D appearance and foreground and highlight colors are selected to provide the necessary contrast. The colors are generated when the widget is created; using `XtSetValues`() to change the background does not change the other colors. With Motif 1.2, use `XmChangeColor`() to change the associated colors when the background color is changed.

In Motif 2.0 and later, the way in which popup menus are posted for particular widgets is rationalized by including within Manager and Primitive popup handler callback resources. When a popup menu is created, an event handler may be automatically installed upon the parent Manager to catch posting events. Once in receipt of the posting event, the Manager's XmNpopupHandlerCallback list is invoked in order to determine which of the various popup menus available is cur-

rently required. Whether the handlers are automatically installed depends upon the value of the XmNpopupEnabled resource of the menus concerned.

Traits

Manager holds the XmNspecifyLayoutDirection, XmNspecifyUnitType, and XmNaccessColors traits, which are inherited by any derived classes.

New Resources

Manager defines the following resources:

Name	Class	Type	Default	Access
XmNbottomShadowColor	XmCBottomShadowColor	Pixel	dynamic	CSG
XmNbottomShadowPixmap	XmCBottomShadowPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNforeground	XmCForeground	Pixel	dynamic	CSG
XmNhighlightColor	XmCHighlightColor	Pixel	dynamic	CSG
XmNhighlightPixmap	XmCHighlightPixmap	Pixmap	dynamic	CSG
XmNinitialFocus	XmCInitialFocus	Widget	NULL	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG[a]
XmNnavigationType	XmCNavigationType	XmNavigationType	XmTAB_GROUP	CSG
XmNshadowThickness	XmCShadowThickness	Dimension	0	CSG
XmNstringDirection	XmCStringDirection	XmStringDirection	dynamic	CG
XmNtopShadowColor	XmCTopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	XmCTopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	XmCTraversalOn	Boolean	True	CSG
XmNunitType	XmCUnitType	unsigned char	dynamic	CSG
XmNuserData	XmCUserData	XtPointer	NULL	CSG

a.Erroneously given as CSG in 2nd edition.

XmNbottomShadowColor
: The color used in drawing the border shadow's bottom and right sides, but only if XmNbottomShadowPixmap is NULL.

XmNbottomShadowPixmap
: The pixmap used in drawing the border shadow's bottom and right sides.

XmNforeground
: The foreground color used by Manager widgets.

XmNhighlightColor
: The color used in drawing the highlighting rectangle, but only if XmNhighlight-Pixmap is XmUNSPECIFIED_PIXMAP.

XmNhighlightPixmap
The pixmap used in drawing the highlighting rectangle.

XmNinitialFocus
In Motif 1.2, the widget ID of the widget that receives the keyboard focus when the manager is a child of a shell and the shell receives the keyboard focus for the first time.

XmNlayoutDirection
In Motif 2.0 and later, specifies the direction in which components of the Manager are laid out. If unspecified, the value is inherited from the nearest ancestor holding the XmQTspecifyLayoutDirection trait. Manager, MenuShell, and VendorShell hold this trait. Possible values:

```
XmLEFT_TO_RIGHT                 XmRIGHT_TO_LEFT
XmBOTTOM_TO_TOP                 XmTOP_TO_BOTTOM
XmBOTTOM_TO_TOP_LEFT_TO_RIGHT
XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
XmTOP_TO_BOTTOM_LEFT_TO_RIGHT
XmTOP_TO_BOTTOM_RIGHT_TO_LEFT
XmLEFT_TO_RIGHT_BOTTOM_TO_TOP
XmRIGHT_TO_LEFT_BOTTOM_TO_TOP
XmLEFT_TO_RIGHT_TOP_TO_BOTTOM
XmRIGHT_TO_LEFT_TOP_TO_BOTTOM
```

XmNnavigationType
Determines the way in which a Manager widget is traversed during keyboard navigation. Possible values:

```
XmNONE                  /* exclude from keyboard navigation       */
XmTAB_GROUP             /* include in keyboard navigation         */
XmSTICKY_TAB_GROUP      /* include in keyboard navigation,        */
                        /* even if XmAddTabGroup() was called     */
XmEXCLUSIVE_TAB_GROUP   /* application defines order of navigation */
```

XmNshadowThickness
The thickness of the shadow border. This resource is dynamically set to 1 in a top-level window and 0 otherwise.

XmNstringDirection
The direction in which to draw the string. Possible values are:

```
XmSTRING_DIRECTION_L_TO_R
XmSTRING_DIRECTION_R_TO_L
XmSTRING_DIRECTION_DEFAULT
```

In Motif 2.0 and later, the XmNstringDirection resource is obsolete, and is subsumed into the XmNlayoutDirection resource. If the XmNlayoutDirection is

NULL, and the XmNstringDirection is XmSTRING_DIRECTION_DEFAULT, the value will be taken from the nearest ancestor which holds the XmQTspecify-LayoutDirection trait. Manager itself, MenuShell, and VendorShell hold this trait.

XmNtopShadowColor
The color used in drawing the border shadow's top and left sides. (Used only if XmNtopShadowPixmap is NULL.)

XmNtopShadowPixmap
The pixmap used in drawing the border shadow's top and left sides.

XmNtraversalOn
If True (default), traversal of this widget is made possible.

XmNunitType
The measurement units to use in resources that specify a size or position--for example, any resources of type Dimension (whose names generally include one of the words "Margin", "Height", "Width", "Thickness", or "Spacing"), as well as the offset resources defined by Form. For a widget whose parent is a manager, the default value is copied from this parent (provided the value hasn't been explicitly set by the application); otherwise, the default is XmPIXELS. Possible values:

XmPIXELS	
Xm100TH_POINTS	
Xm100TH_MILLIMETERS	
Xm100TH_FONT_UNITS	
Xm1000TH_INCHES	
XmINCHES	(2.0)
XmPOINTS	(2.0)
XmFONT_UNITS	(2.0)

XmNuserData
A pointer to data that the application can attach to the widget. This resource is unused internally.

Callback Resources

Manager defines the following callback resources:

Callback	Reason Constant
XmNhelpCallback	XmCR_HELP
XmNpopupHandlerCallback	XmCR_POST XmCR_REPOST

XmNhelpCallback
List of callbacks that are called when help is requested.

XmNpopupHandlerCallback
In Motif 2.0 and later, the list of callbacks invoked in order to determine which popup menu to display.

Callback Structure

With the exception of a popup handler callback, each callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* set to XmCR_HELP                 */
    XEvent      *event;         /* event structure that triggered callback */
} XmAnyCallbackStruct;
```

A popup handler callback is passed a pointer to the following structure:

```
typedef struct {
    int         reason;         /* the reason the callback is invoked   */
    XEvent      *event;         /* event structure that triggered callback */
    Widget      menuToPost;     /* the menu to post                 */
    Boolean     postIt;         /* whether to continue posting      */
    Widget      target;         /* the manager descendant issuing request */
} XmPopupHandlerCallbackStruct;
```

reason is either XmCR_POST[1] or XmCR_REPOST[2]. XmCR_POST is the normal menu post request. XmCR_REPOST is called when the menu is unposted because of event replay.

menuToPost is the suggested menu to be posted. Alter the element if a different menu is required.

1.Erroneously given as XmPOST in 2nd edition.

2.Erroneously given as XmREPOST in 2nd edition.

postIt is a flag indicating whether the posting operation is to continue once the callback has finished. The default value is True if reason is XmPOST, otherwise False.

target is the widget or gadget which the Manager believes best describes the source of the posting event. The algorithm performs a recursive descent, matching the received event against the location of managed children.

Inherited Resources

Manager inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. Manager resets the default value of XmNborderWidth from 1 to 0.

Name	Inherited From	Name	Inherited From
XmNaccelerators	Core	XmNheight	Core
XmNancestorSensitive	Core	XmNinsertPosition	Composite
XmNbackground	Core	XmNinitialResourcesPersistent	Core
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNscreen	Core
XmNborderWidth	Core	XmNsensitive	Core
XmNchildren	Composite	XmNtranslations	Core
XmNcolormap	Core	XmNwidth	Core
XmNdepth	Core	XmNx	Core
XmNdestroyCallback	Core	XmNy	Core

Translations

For Manager widgets that have gadget children:

Event	Action
BAny Motion	ManagerGadgetButtonMotion()
BSelect Press	ManagerGadgetArm()
BSelect Click	ManagerGadgetActivate()
BSelect Release	ManagerGadgetActivate()
BSelect Press 2+	ManagerGadgetMultiArm()

Event	Action
BSelect Release 2+	ManagerGadgetMultiActivate
BTransfer Press	ManagerGadgetDrag()
KActivate	ManagerParentActivate() (1.2)
KCancel	ManagerParentCancel()
KPrevField	ManagerGadgetPrevTabGroup()
KNextField	ManagerGadgetNextTabGroup()
KUp	ManagerGadgetTraverseUp()
KDown	ManagerGadgetTraverseDown()
KLeft	ManagerGadgetTraverseLeft()
KRight	ManagerGadgetTraverseRight()
KSelect	ManagerGadgetSelect()
KBeginLine	ManagerGadgetTraverseHome()
KHelp	ManagerGadgetHelp(), ManagerGadgetSelect() (1.2)
KAny	ManagerGadgetKeyInput()

Action Routines

The action routines for a Manager widget affect the gadget child that has the keyboard focus. The descriptions below refer to the gadget that has the focus.

ManagerGadgetActivate()
: Activates the gadget.

ManagerGadgetArm()
: Arms the gadget.

ManagerGadgetButtonMotion()
: Triggers the mouse motion event that the gadget received.

ManagerGadgetDrag()
: In Motif 1.2, initiates a drag and drop operation using the contents of a gadget's label.

ManagerGadgetHelp()
: Invokes the list of callbacks specified by the gadget's XmNhelpCallback resource. If the gadget doesn't have any help callbacks, the ManagerGadgetHelp() routine invokes those associated with the nearest ancestor that has them.

ManagerGadgetKeyInput()
: Triggers the keyboard event that the gadget received.

ManagerGadgetMultiActivate()
: Processes a multiple click of the mouse.

ManagerGadgetMultiArm()
: Processes a multiple press of the mouse button.

ManagerGadgetNextTabGroup()
ManagerGadgetPrevTabGroup()
: Traverses to the beginning of the next/previous tab group, wrapping if necessary.

ManagerGadgetSelect()
: Arms and activates the gadget.

ManagerGadgetTraverseDown()
ManagerGadgetTraverseUp()
: Within the same tab group, descends/ascends to the item below/above the gadget, wrapping if necessary.

ManagerGadgetTraverseHome()
: Changes the focus to the first item in the tab group.

ManagerGadgetTraverseLeft()
ManagerGadgetTraverseRight()
: Within the same tab group, traverses to the item on the left/right of the gadget, wrapping if necessary.

ManagerGadgetTraverseNext()
ManagerGadgetTraversePrev()
: Within the same tab group, traverses to the next/previous item, wrapping if necessary.

ManagerParentActivate()
: In Motif 1.2, passes the KActivate event to the parent if it is a manager.

ManagerParentCancel()
: In Motif 1.2, passes the KCancel event to the parent if it is a manager.

Additional Behavior

Manager has the following additional behavior:

<FocusIn>

If the event occurs in a gadget, highlights the gadget and gives it the focus under the explicit keyboard focus policy.

<FocusOut>

If the event occurs in a gadget, unhighlights the gadget and removes the focus under the explicit keyboard focus policy.

See Also

`Composite`(2), `Constraint`(2), *Core*(2), `XmGadget`(2).

Name

XmMenuBar – a type of RowColumn widget used as a menu bar.

Synopsis

Public Header:
<Xm/RowColumn.h>

Class Name:
XmRowColumn

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmRowColumn

Class Pointer:
xmRowColumnWidgetClass

Instantiation:
widget = `XmCreateMenuBar` (parent, name,...)

Functions/Macros:
`XmCreateMenuBar`(), `XmCreateSimpleMenuBar`(), `XmVaCreateSimpleMenuBar`(), `XmIsRowColumn`()

Description

An XmMenuBar is an instance of a RowColumn widget that is normally used for constructing a pulldown menu system. An application typically places a MenuBar across the top of the main application window. CascadeButtons are added to the MenuBar and pulldown menus are associated with each of the CascadeButtons.

MenuBar is a RowColumn widget whose XmNrowColumnType resource is set to XmMENU_BAR. The XmNentryClass resource is set to xmCascadeButtonWidgetClass and XmNisHomogeneous is set to True, so that only CascadeButtons can be added to the widget. The XmNmenuAccelerator resource is set to KMenuBar and XmNmenuPost is set to BSelect Press. The XmNmenuHelpWidget resource can be set to specify the CascadeButton for the Help menu. The XmNorientation resource is set to XmHORIZONTAL.

A MenuBar can be created using `XmCreateMenuBar`(). In this case, the MenuBar does not automatically contain any CascadeButtons; they are added by the application.

A MenuBar can also be created by XmCreateSimpleMenuBar(), which automatically creates the MenuBar with the specified CascadeButtonGadgets as children. This routine uses the RowColumn resources associated with the creation of simple menus. For a MenuBar, the only type allowed in the XmNbuttonType resource is XmCASCADEBUTTON. The name of each CascadeButtonGadget is button_*n*, where *n* is the number of the button, ranging from 0 to 1 less than the number of buttons in the MenuBar.

Default Resource Values

A MenuBar sets the following default values for RowColumn resources:

Name	Default
XmNentryClass	xmCascadeButtonWidgetClass
XmNisHomogenous	True
XmNmenuAccelerator	KMenuBar
XmNmenuPost	BSelect Press
XmNorientation	XmHORIZONTAL
XmNrowColumnType	XmMENU_BAR

See Also

XmCreateObject(1), XmVaCreateSimpleMenuBar(1), XmCascadeButton(2), XmRowColumn(2).

Name

XmMenuShell widget class – a shell widget meant to contain popup and pull-down menu panes.

Synopsis

Public Header:
<Xm/MenuShell.h>

Class Name:
XmMenuShell

Class Hierarchy:
Core →Composite →Shell →OverrideShell →XmMenuShell

Class Pointer:
xmMenuShellWidgetClass

Instantiation:
widget = `XmCreateMenuShell` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmMenuShellWidgetClass,...)

Functions/Macros:
`XmCreateMenuShell`(), `XmCreatePopupMenu`(), `XmCreatePulldownMenu`(),
`XmIsMenuShell`()

Description

MenuShell is a subclass of OverrideShell that is meant to contain only popup or pulldown menu panes. Most application writers do not need to create MenuShell widgets explicitly because they are created automatically by the convenience routines XmCreatePopupMenu() and XmCreatePulldownMenu().

If you do not use the convenience functions and create your own MenuShell widgets, the type of menu system being built determines the parent to specify for the MenuShell. For a top-level popup menu, specify the widget from which it will pop up. For a pulldown menu pane from the menu bar, specify the menu bar. For a pulldown menu pane from another pulldown menu or a popup menu, specify the menu pane from which it is pulled down. For pulldown menu in an option menu, specify the option menu's parent.

Traits

MenuShell holds the XmQTspecifyLayoutDirection and XmQTspecifyRenderTable traits, which are inherited by any derived classes, and uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

New Resources

MenuShell defines the following resource:

Name	Class	Type	Default	Access
XmNbuttonFontList	XmCButtonFontList	XmFontList	dynamic	CSG
XmNbuttonRenderTable	XmCButtonRenderTable	XmRenderTable	dynamic	CSG
XmNdefaultFontList	XmCDefaultFontList	XmFontList	dynamic	CG
XmNlabelFontList	XmCLabelFontList	XmFontList	dynamic	CSG
XmNlabelRenderTable	XmCLabelRenderTable	XmRenderTable	dynamic	CSG
XmNlayoutDirection	XmCLayoutDirection	XmDirection	dynamic	CG[a]

a.Erroneously given as CSG in 2nd edition.

XmNbuttonFontList
: The font list used for the button children of the MenuShell widget. In Motif 2.0 and later, the XmfontList is considered obsolete, and the Rendition Table is the preferred method of setting appearance. Any XmNbuttonRenderTable value will take precedence.

XmNbuttonRenderTable
: Specifies the render table used for button children of the MenuShell widget. If the value is NULL, this will be inherited from the nearest ancestor with the XmQTspecifyRenderTable trait, taking the XmBUTTON_RENDER_TABLE value from the ancestor so found. The BulletinBoard, VendorShell, and MenuShell widgets and derived classes set this trait.

XmNdefaultFontList
: The default font list for the children of the MenuShell widget. This resource is obsolete in Motif 1.2.

XmNlabelFontList
: The font list used for the label children of the MenuShell widget. In Motif 2.0 and later, the XmFontList is considered obsolete, and the Rendition Table is the preferred method of setting appearance. Any XmNlabelRenderTable value will take precedence.

XmNlabelRenderTable
: Specifies the render table used for label children of the MenuShell widget. If the value is NULL, this will be inherited from the nearest ancestor with the XmQTspecifyRenderTable trait, taking the XmLABEL_RENDER_TABLE value from the ancestor so found. The BulletinBoard, VendorShell, and MenuShell widgets and derived classes set this trait.

XmNlayoutDirection
In Motif 2.0 and later, specifies the default direction in which descendants of the MenuShell are laid out.

XmLEFT_TO_RIGHT
XmRIGHT_TO_LEFT
XmBOTTOM_TO_TOP
XmTOP_TO_BOTTOM
XmBOTTOM_TO_TOP_LEFT_TO_RIGHT
XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
XmTOP_TO_BOTTOM_LEFT_TO_RIGHT
XmTOP_TO_BOTTOM_RIGHT_TO_LEFT
XmLEFT_TO_RIGHT_BOTTOM_TO_TOP
XmRIGHT_TO_LEFT_BOTTOM_TO_TOP
XmLEFT_TO_RIGHT_TOP_TO_BOTTOM
XmRIGHT_TO_LEFT_TOP_TO_BOTTOM

Inherited Resources

MenuShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. MenuShell sets the default value of XmNallowShellResize to True and XmNborderWidth to 0. The default values of XmNoverrideRedirect and XmNsaveUnder are set to True by OverrideShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinitialResourcesPersistent	Core
XmNallowShellResize	Shell	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNmappedWhenManaged	Core
XmNbackground	Core	XmNnumChildren	Composite
XmNbackgroundPixmap	Core	XmNoverrideRedirect	Shell
XmNborderColor	Core	XmNpopdownCalback	Shell
XmNborderPixmap	Core	XmNpopupCallback	Shell
XmNborderWidth	Core	XmNsaveUnder	Shell
XmNchildren	Composite	XmNscreen	Core
XmNcolormap	Core	XmNsensitive	Core
XmNcreatePopupChildProc	Shell	XmNtranslations	Core
XmNdepth	Core	XmNvisual	Shell
XmNdestroyCallback	Core	XmNwidth	Core
XmNgeometry	Shell	XmNx	Core
XmNheight	Core	XmNy	Core

Translations

Event	Action
BSelect Press	ClearTraversal()
BSelect Release	MenuShellPopdownDone()

Action Routines

MenuShell defines the following action routines:

ClearTraversal()
: Shuts off keyboard traversal within this menu, turns on mouse traversal, and unposts any submenus that this menu posted.

MenuShellPopdownDone()
: Unposts the menu tree and restores the previous focus.

MenuShellPopdownOne()
: Like MenuShellPopdownDone() except that it unposts only one level of the menu tree. In a top-level pulldown menu pane attached to a menu bar, this action routine disarms the cascade button and the menu bar.

See Also

`XmCreateObject`(1), `Composite`(2), `Core`(2), `OverrideShell`(2), `Shell`(2), `XmRowColumn`(2).

Name

XmMessageBox widget class – a composite widget used for creating message dialogs.

Synopsis

Public Header:
<Xm/MessageB.h>

Class Name:
XmMessageBox

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmBulletinBoard → XmMessageBox

Class Pointer:
xmMessageBoxWidgetClass

Instantiation:
widget = `XmCreateMessageBox` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmMessageBoxWidgetClass,...)

Functions/Macros:
`XmCreateErrorDialog`(), `XmCreateInformationDialog`(), `XmCreateMessageBox`(), `XmCreateMessageDialog`(), `XmCreateQuestionDialog`(), `XmCreateTemplateDialog`(), `XmCreateWarningDialog`(), `XmCreateWorkingDialog`(), `XmIsMessageBox`(), `XmMessageBoxGetChild`()

Description

MessageBox is composite widget that is used for creating simple message dialog boxes, which normally present transient messages. A MessageBox usually contains a message symbol, a message, three PushButtons, and a separator between the message and the buttons. The names of the symbol and the separator gadgets are Symbol and Separator. In Motif 1.2, the default symbols and button labels can be localized. The XmNdialogType resource controls the type of message symbol that is displayed. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Cancel**, and **Help** by default.

You can customize a MessageBox by removing existing children or adding new children. Use `XmMessageBoxGetChild`() to retrieve the widget ID of an existing child and then unmanage the child[1]. With Motif 1.2, multiple widgets can be added as children of a MessageBox. If a menu bar is added, it is placed at the top of the window. Any buttons are placed after the **OK** button. Any addi-

tional children are placed below the message. In Motif 1.1, only a single widget can be added as a child of a MessageBox. This child is placed below the message and acts as a work area.

In Motif 1.2 and later, a XmNdialogType of XmDIALOG_TEMPLATE create a TemplateDialog which contains nothing but a separator by default. Specifying callback, label string, or pixmap symbol resources causes the appropriate children of the MessageBox to be created.

Traits

MessageBox uses the XmQTactivatable trait.

New Resources

MessageBox defines the following resources:

Name	Class	Type	Default	Access
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	""[a]	CSG
XmNminimizeButtons	XmCMinimizeButtons	Boolean	False	CSG
XmNokLabelString	XmCOkLabelString	XmString	dynamic	CSG
XmNsymbolPixmap	XmCPixmap	Pixmap	dynamic	CSG

a.Strictly speaking, NULL, which is internally mapped through the empty string.

XmNcancelLabelString
The string that labels the **Cancel** button. In Motif 1.2, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Cancel".

XmNdefaultButtonType
Specifies which PushButton provides the default action. Possible values:

XmDIALOG_CANCEL_BUTTON
XmDIALOG_OK_BUTTON
XmDIALOG_HELP_BUTTON

1. From Motif 2.0 onwards, use `XtNameToWidget`(): the various toolkit GetChild() routines are considered deprecated.

XmNdialogType
The type of MessageBox dialog, which also indicates the message symbol that displays by default. Possible values:

XmDIALOG_ERROR	XmDIALOG_TEMPLATE (1.2)
XmDIALOG_INFORMATION	XmDIALOG_WARNING
XmDIALOG_MESSAGE	XmDIALOG_WORKING
XmDIALOG_QUESTION	

XmNhelpLabelString
The string that labels the **Help** button. In Motif 1.2, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Help"**.**

XmNmessageAlignment
The type of alignment for the message label. Possible values:

XmALIGNMENT_BEGINNING
XmALIGNMENT_CENTER
XmALIGNMENT_END

XmNmessageString
The string to use as the message label.

XmNminimizeButtons
If False (default), all buttons are standardized to be as wide as the widest button and as high as the highest button. If True, buttons will keep their preferred size.

XmNokLabelString
The string that labels the **OK** button. In Motif 1.2, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "OK"**.**

XmNsymbolPixmap
The pixmap label to use as the message symbol.

Callback Resources

MessageBox defines the following callback resources:

Callback	Reason Constant
XmNcancelCallback	XmCR_CANCEL
XmNokCallback	XmCR_OK

XmNcancelCallback
List of callbacks that are called when the user selects the **Cancel** button.

XmNokCallback
List of callbacks that are called when the user selects the **OK** button.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;     /* the reason that the callback was called */
    XEvent      *event;     /* event structure that triggered callback */
} XmAnyCallbackStruct;
```

Inherited Resources

MessageBox inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of XmNborderWidth is reset to 0 by XmManager. BulletinBoard sets the value of XmNinitialFocus to XmNdefaultButton and resets the default XmNshadowThickness from 0 to 1 if the MessageBox is a child of a DialogShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNlabelFontList	XmBulletinBoard
XmNallowOverlap	XmBulletinBoard	XmNlabelRenderTable	XmBulletinBoard
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNautoUnmanage	XmBulletinBoard	XmNmapCallback	XmBulletinBoard
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNmarginHeight	XmBulletinBoard
XmNborderColor	Core	XmNmarginWidth	XmBulletinBoard
XmNborderPixmap	Core	XmNnavigationType	XmManager
XmNborderWidth	Core	XmNnoResize	XmBulletinBoard
XmNbottomShadowColor	XmManager	XmNnumChildren	Composite
XmNbottomShadowPixmap	XmManager	XmNpopupHandlerCallback	XmManager
XmNbuttonFontList	XmBulletinBoard	XmNresizePolicy	XmBulletinBoard
XmNbuttonRenderTable	XmBulletinBoard	XmNscreen	Core
XmNcancelButton	XmBulletinBoard	XmNsensitive	Core
XmNchildren	Composite	XmNshadowThickness	XmManager
XmNcolormap	Core	XmNshadowType	XmBulletinBoard
XmNdefaultButton	XmBulletinBoard	XmNstringDirection	XmManager
XmNdefaultPosition	XmBulletinBoard	XmNtextFontList	XmBulletinBoard
XmNdepth	Core	XmNtextRenderTable	XmBulletinBoard
XmNdestroyCallback	Core	XmNtextTranslations	XmBulletinBoard
XmNdialogStyle	XmBulletinBoard	XmNtopShadowColor	XmManager
XmNdialogTitle	XmBulletinBoard	XmNtopShadowPixmap	XmManager

Resource	Inherited From	Resource	Inherited From
XmNfocusCallback	XmBulletinBoard	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNunmapCallback	XmBulletinBoard
XmNhighlightColor	XmManager	XmNuserData	XmManager
XmNhighlightPixmap	XmManager	XmNwidth	Core
XmNinitialFocus	XmManager	XmNx	Core
XmNinitialResourcesPersistent	Core	XmNy	Core
XmNinsertPosition	Composite		

Translations

The translations for MessageBox include those from XmManager.

Additional Behavior

MessageBox has the following additional behavior:

MAny KCancel
: For a sensitive **Cancel** button, invokes the callbacks in XmNactivateCallback**.**

KActivate
: For the button that has keyboard focus, or the default button, invokes the callbacks in XmNactivateCallback.

<OK Button Activated>
: Invokes the callbacks for XmNokCallback.

<Cancel Button Activated>
: Invokes the callbacks for XmNcancelCallback.

<Help Button Activated>
: Invokes the callbacks for XmNhelpCallback.

<FocusIn>
: Invokes the callbacks for XmNfocusCallback.

<Map>
: Invokes the callbacks for XmNmapCallback if the parent is a DialogShell.

<Unmap>
: Invokes the callbacks for XmNunmapCallback if the parent is a DialogShell.

See Also

XmCreateObject(1), XmMessageBoxGetChild(1), Composite(2), Constraint(2), Core(2), XmBulletinBoard(2), XmErrorDialog(2), XmInformationDialog(2), XmManager(2), XmQuestionDialog(2), XmTemplateDialog(2), XmWarningDialog(2), XmWorkingDialog(2).

Name

XmMessageDialog – an unmanaged MessageBox as a child of DialogShell.

Synopsis

Public Header:
<Xm/MessageB.h>

Instantiation:
widget = XmCreateMessageDialog (parent, name,...)

Functions/Macros:
`XmCreateMessageDialog`(), `XmMessageBoxGetChild`()

Description

An XmMessageDialog is a compound object created by a call to `XmCreate-MessageDialog`() that an application can use to present a message to the user. A MessageDialog consists of a DialogShell with an unmanaged MessageBox widget as its child. The MessageBox resource XmNdialogType is set to XmDIALOG_MESSAGE. A MessageDialog includes four components: a symbol, a message, three buttons, and a separator between the message and the buttons. By default, there is no symbol. In Motif 1.2, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Cancel**, and **Help** by default.

Default Resource Values

A MessageDialog sets the following default values for MessageBox resources:

Name	Default
XmNdialogType	XmDIALOG_MESSAGE

Widget Hierarchy

When a MessageDialog is created with a specified name, the DialogShell is named name*_popup* and the MessageBox is called *name***.**

See Also

`XmCreateObject`(1), `XmMessageBoxGetChild`(1), `XmDialogShell`(2), `XmMessageBox`(2).

Name

XmNotebook widget class – a constraint widget which lays out its children like pages in a book

Synopsis

Public Header:
<Xm/Notebook.h>

Class Name:
XmNotebook

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmNotebook

Class Pointer:
xmNotebookWidgetClass

Instantiation:
widget = `XmCreateNotebook` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmNotebookWidgetClass,...)

Functions/Macros:
`XmNotebookGetPageInfo`(), `XmCreateNotebook`(), `XmIsNotebook`()

Availability

Motif 2.0 and later.

Description

Notebook is a constraint widget that organises its children into logical pages. It lays itself out so that the stack of logical pages look like the pages of an open notebook. The Notebook has visuals to simulate book binding and overlapping (back) page edges. Only one page is visible at any time.

The Notebook can be divided in sections and sub-sections by creating tabs which are displayed along the edge of the Notebook pages. A major tab divides the Notebook into sections, within each section there may be further minor tabs. A tab is associated with a page, so that activating a tab causes the relevant page to be displayed within the Notebook. The Notebook automatically creates a tab scroller, consisting of a set of four ArrowButtonGadgets. These are used for moving backwards and forwards between the major and minor tabs, when it is not possible to display all the tabs.

A status area can be associated with a page, and this is used for providing additional information, typically the page number.

In addition to associating tabs with pages, a page scroller can also be created for moving between pages of the Notebook. This can be any preferred method of navigating between the pages. If, however, there is no page scroller associated with the Notebook when it is realized, the Notebook creates a default page scroller, consisting of a SpinBox. The Notebook only requires one page scroller.

A fully programmed Notebook therefore consists of pages, tabs, status areas, a page scroller, and a tab scroller. The programmer does not create the tab scroller, and only optionally provides an alternative page scroller. Tabs and status areas are also optional.

An application adds a page to the Notebook by creating a child with the constraint resource XmNnotebookChildType set to XmPAGE. Any widget derived from RectObj may form a page of the Notebook. The default behavior is that if XmNnotebookChildType is unspecified (XmNONE), the child forms a page, with the following exceptions: a child with the XmQTactivatable trait (ArrowButton, DrawnButton, PushButton, and derived classes) is set up as a tab; a child with the XmQTaccessTextual trait (Label, Text, TextField, and derived classes) becomes a status area; a child with the XmQTnavigator trait (ScrollBar, SpinBox, and derivatives) is made into a page scroller.

Pages are ordered by setting the constraint resource XmNpageNumber appropriately for each child of type XmPAGE. A tab is attached to a page by adding an appropriate widget to the Notebook with XmNnotebookChildType set to either XmMAJOR_TAB or XmMINOR_TAB, and with the XmNpageNumber constraint resource set to that of the relevant page. Similarly, a status area can be attached to a page by adding the child with XmNnotebookChildType set to XmSTATUS_AREA, and again appropriately setting XmNpageNumber to the relevant page.

The resources XmNcurrentPageNumber, XmNfirstPageNumber, and XmNlastPageNumber control the range of pages which may be displayed. If a child has a XmNpageNumber constraint value which falls outside of the bounds set by XmNfirstPageNumber and XmNlastPageNumber, either the bounds must be altered to encompass the page constraint, or the page number itself must be altered, before the Notebook will display the child. The XmNlastPageNumber resource is dynamically maintained as the highest XmNpageNumber supplied, unless the application itself changes the value of XmNlastPageNumber. Once set by the application, the XmNlastPageNumber resource is no longer maintained by the Notebook, even if higher XmNpageNumber values are set.

When a child page is managed and no XmNpageNumber has been explicitly assigned, the Notebook automatically assigns a number. The assigned number is the smallest unused number which is not less than either the XmNfirstPageNumber or the previous automatically allocated number. This may exceed XmNlastPageNumber. If XmNfirstPageNumber exceeds XmNlastPageNumber, the behavior of the Notebook is undefined. The default value of the constraint XmNpageNumber is XmUNSPECIFIED_PAGE_NUMBER, which causes the Notebook to automatically assign a number.

Pages may be assigned duplicate XmNpageNumber values, in which case the Notebook displays the child which is most recently managed. There may be gaps in the assigned page numbering scheme, in which case the Notebook displays a blank background, unless the application dynamically provides feedback into the background whilst processing a XmNpageChangedCallback. It is possible to create a tab for an empty slot by assigning an XmNpageNumber constraint to a tab child, for which there is no corresponding XmPAGE child.

The ConstraintSetValues method of the Notebook sorts children into ascending logical page number order. Logical page numbers may be assigned in any order, provided that the programmer does not rely anywhere upon the particular order in which children are added.

Traits

Notebook holds the XmQTscrollFrame, XmQTtraversalControl, and XmQTspecifyUnhighlight traits, which are inherited by any derived classes, and uses the XmQTscrollFrame, XmQTactivatable, XmQTnavigator, XmQTjoinSide, and XmQTaccessTextual traits.

New Resources

Notebook defines the following resources:

Name	Class	Type	Default	Access
XmNbackPageBackground	XmCBackPageBackground	Pixel	dynamic	CSG
XmNbackPageForeground	XmCBackPageForeground	Pixel	dynamic	CSG
XmNbackPageNumber	XmCBackPageNumber	Cardinal	2	CSG
XmNbackPagePlacement	XmCBackPagePlacement	unsigned char	XmBOTTOM_RIGHT	CSG
XmNbackPageSize	XmCBackPageSize	Dimension	8	CSG
XmNbindingPixmap	XmCBindingPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNbindingType	XmCBindingType	unsigned char	XmSPIRAL	CSG
XmNbindingWidth	XmCBindingWidth	Dimension	25	CSG
XmNcurrentPageNumber	XmCCurrentPageNumber	int	XmUNSPECIFIED_PAGE_NUMBER	CSG

Name	Class	Type	Default	Access
XmNfirstPageNumber	XmCFirstPageNumber	int	1	CSG
XmNframeBackground	XmCFrameBackground	Pixel	dynamic	CSG
XmNframeShadowThickness	XmCShadowThickness	Dimension	dynamic	CSG
XmNinnerMarginHeight	XmCInnerMarginHeight	Dimension	0	CSG
XmNinnerMarginWidth	XmCInnerMarginWidth	Dimension	0	CSG
XmNlastPageNumber	XmCLastPageNumber	int	XmUNSPECIFIED_PAGE_NUMBER	CSG
XmNmajorTabSpacing	XmCMajorTabSpacing	Dimension	3	CSG
XmNminorTabSpacing	XmCMinorTabSpacing	Dimension	3	CSG
XmNorientation	XmCOrientation	unsigned char	XmHORIZONTAL	CSG

XmNbackPageBackground
Specifies the background color for drawing the back (overlapped) pages.

XmNbackPageForeground
Specifies the foreground color for drawing the back (overlapped) pages.

XmNbackPageNumber
Specifies the number of lines to draw to represent the back (overlapped) pages. The minimum value is 1, the maximum is half the XmNbackPageSize value.

XmNbackPagePlacement
Specifies where to draw the back (overlapped) pages. The default depends upon the XmNlayoutDirection of the Notebook parent widget, and the Notebook XmNorientation. Possible values:

```
XmBOTTOM_RIGHT    /* lines drawn to bottom and right */
XmBOTTOM_LEFT     /* lines drawn to bottom and left */
XmTOP_RIGHT       /* lines drawn to top and right    */
XmTOP_LEFT        /* lines drawn to top and left     */
```

XmNbackPageSize
Specifies the thickness, in pixels, of the back page rendering.

XmNbindingPixmap
Specifies the pixmap for drawing the binding. The value is only used if the XmNbindingType is XmPIXMAP or XmPIXMAP_OVERLAP_ONLY.

XmNbindingType
Specifies the type of binding. Possible values:

XmNONE	XmSOLID
XmSPIRAL	XmPIXMAP
XmPIXMAP_OVERLAP_ONLY	

A value of XmNONE displays no binding.

The value XmSOLID draws a solid binding using the foreground color of the widget. The binding is contained within the area of a frame containing the pages of the Notebook, and curtailed by the value of XmNbindingWidth.

The value XmSPIRAL draws a spiral binding using the foreground color of the widget. The binding is contained within the area of a frame containing the pages of the Notebook, and curtailed by the value of XmNbindingWidth, and by an area outside the frame of the pages also bounded by the value of XmNbindingWidth.

The value XmPIXMAP draws a binding using the value of XmNbindingPixmap as a tile or stipple, depending upon the depth of the supplied pixmap. A pixmap of depth 1 is used as a stipple, and tiled otherwise. The foreground color of the Notebook is used when stippling. The size of the binding drawn is the larger of XmNbindingWidth and the width of the pixmap.

The value XmPIXMAP_OVERLAP_ONLY is similar to XmPIXMAP, except that the size of the binding is bounded only by the value of XmNbindingWidth, and not the width of the pixmap.

XmNbindingWidth
Specifies the width, in pixels, of the Notebook binding area.

XmNcurrentPageNumber
Specifies the page number of the currently visible page. Initially set to the value of XmNfirstPageNumber, the XmNcurrentPageNumber is constrained to be not be less than the XmNfirstPageNumber and not to exceed the XmNlastPageNumber.

XmNfirstPageNumber
Specifies the page number of the first logical page that may be displayed in the Notebook. Any child with an XmNpageNumber less than this value cannot be displayed until such time as either the XmNfirstPageNumber, or the XmNpageNumber of the given child itself, is suitably altered.

XmNframeBackground
Specifies the background color for the Notebook frame.

XmNframeShadowThickness
Specifies the shadow thickness around the Notebook frame. In Motif 2.0 the default is 2. In Motif 2.1 and later, the default depends upon the XmDisplay XmNenableThinThickness resource: if True, the default is 1, otherwise 2.

XmNinnerMarginHeight
Specifies the margin on the top and bottom sides of page, status area, and page scroller children.

XmNinnerMarginWidth
Specifies the margin on the left and right sides of page, status area, and page scroller children.

XmNlastPageNumber
Specifies the page number of the last logical page that may be displayed in the Notebook. Any child with an XmNpageNumber in excess of this value may not be displayed until such time as either the XmNlastPageNumber, or the XmNpageNumber of the given child itself, is suitably altered. The XmNlastPageNumber is automatically set by the Notebook as pages are added, unless the programmer directly changes the value: once modified, the Notebook no longer maintains the last page reference, and the programmer must continue to set the value as required.

XmNmajorTabSpacing
Specifies the spacing between major tabs. If XmNframeShadowThickness exceeds the value of the spacing, then the shadow thickness is used.

XmNminorTabSpacing
Specifies the spacing between minor tabs. If XmNframeShadowThickness exceeds the value of the spacing, then the shadow thickness is used.

XmNorientation
Specifies the orientation of the Notebook. Possible values:

```
XmHORIZONTAL    /* binding on left or right side */
XmVERTICAL      /* binding on top or bottom side */
```

New Constraint Resources

Notebook defines the following constraint resources for its children:

Name	Class	Type	Default	Access
XmNnotebookChildType	XmCNotebookChildType	unsigned char	XmNONE	CG[a]
XmNpageNumber	XmCPageNumber	int	XmUNSPECIFIED_PAGE_NUMBER	CSG
XmNresizable	XmCResizable	Boolean	True	CSG

a.Erroneously listed as CSG in 2nd edition.

XmNnotebookChildType
Specifies the child type of the Notebook. Possible values:

```
XmPAGE              /* the child is a page         */
XmMAJOR_TAB         /* the child is a major tab    */
XmMINOR_TAB         /* the child is a minor page   */
XmSTATUS_AREA       /* the child is a status area  */
XmPAGE_SCROLLER     /* the child is a page scroller */
```

XmNpageNumber
Specifies a logical page number associated with a child of the Notebook. If unspecified, the Notebook assigns the next unallocated number when the child is managed. The assigned number is calculated to be not less than the Notebook XmNfirstPageNumber.

XmNresizable
Specifies whether any child resize request is processed by the Notebook.

Callback Resources

Notebook defines the following callback resources:

Callback	Reason Constant
XmNpageChangedCallback	XmCR_NONE XmCR_PAGE_SCROLLER_INCREMENT XmCR_PAGE_SCROLLER_DECREMENT XmCR_MAJOR_TAB XmCR_MINOR_TAB

XmNpageChangedCallback
List of callbacks called when the current logical page number is initialized or changed.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int       reason;              /* the reason that the callback was called    */
    XEvent    *event;              /* points to event structure                  */
                                   /* that triggered callback                    */
    int       page_number;         /* current page number                        */
    Widget    page_widget;         /* widget associated with current page number */
    int       prev_page_number;    /* previous current logical page number       */
    Widget    prev_page_widget;    /* widget associated with previous            */
                                   /* current logical page number                */
} XmNotebookCallbackStruct;
```

The structure members page_number, page_widget, prev_page_number, prev_page_widget are valid for any value of reason.

reason specifies the cause of callback invocation. At Notebook initialization, the page changed callback list is called in order to set up the first current page, and the *reason* structure member in this instance will have the value XmCR_NONE.

Thereafter, if a tab child is activated, the *reason* member has the value XmCR_MAJOR_TAB or XmCR_MINOR_TAB, depending upon the XmNnotebookChildType resource of the selected tab. If the page scroller is activated, the *reason* field is either XmCR_PAGE_SCROLLER_INCREMENT or XmCR_PAGE_SCROLLER_DECREMENT, depending upon the scroller action. The *reason* member is also XmCR_NONE if the XmNcurrentPageNumber resource is changed through `XtSetValues`().

page_number specifies the new logical page number.

page_widget specifies the widget which has the new logical page number. This is NULL if no page widget with an XmNpageNumber value equal to *page_number* is found.

prev_page_number specifies the current logical page number. At Notebook initialization, the value is XmUNSPECIFIED_PAGE_NUMBER.

prev_page_widget specifies the currently displayed page child of the Notebook. This is NULL at Notebook initialization, and where there is no page widget with an XmNpageNumber value equal to *prev_page_number*.

Inherited Resources

Notebook inherits the resources shown below. The resources are listed alphabetically, along with the superclass that defines them.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Translations

The translations for Notebook include those of XmManager. In addition, Notebook places the following accelerators upon Tab children:

Event	Action
KBeginLine	TraverseTab(Home)
KEndLine	TraverseTab(End)
KLeft	TraverseTab(Previous)
KRight	TraverseTab(Next)
KUp	TraverseTab(Previous)
KDown	TraverseTab(Next)

Action Routines

Notebook defines the following action routines:

TraverseTab(type)

A generic action to move the focus between major and minor tabs in the Notebook. The action type may be one of Home, End, Previous, or Next.

See Also

`XmNotebookGetPageInfo`(1), `XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `RectObj`(2), `XmManager`(2).

Name

XmOptionMenu – a type of RowColumn widget used as an option menu.

Synopsis

Public Header:
<Xm/RowColumn.h>

Class Name:
XmRowColumn

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmRowColumn

Class Pointer:
xmRowColumnWidgetClass

Instantiation:
widget = XmCreateOptionMenu (parent, name,...)

Functions/Macros:
`XmCreateOptionMenu`(), `XmCreateSimpleOptionMenu`(), `XmVaCreateSimpleOptionMenu`(), `XmIsRowColumn`(), `XmOption-ButtonGadget`(), `XmOptionLabelGadget`()

Description

An OptionMenu is an instance of a RowColumn widget that is used as a menu that allows a user to select one of several choices. An OptionMenu consists of a label, a selection area, and pulldown menu pane. When you create an OptionMenu, you must supply the pulldown menu pane via the XmNsubMenuId resource. The menu pane must already exist and it must be a child of the OptionMenu's parent. The label (a LabelGadget) and the selection area (a CascadeButtonGadget) are created by the OptionMenu. You can specify the label string with the XmNlabelString resource.

OptionMenu is a RowColumn widget whose XmNrowColumnType resource is set to XmMENU_OPTION. The XmNorientation resource defaults to XmHORIZONTAL, which means that the label is displayed to the left of the selection area. If the resource is set to XmVERTICAL, the label is placed above the selection area. The selection area posts the menu pane, as well as displays the label of the current selection. The XmNmenuPost resource is set to BSelect Press. The XmNmenuHistory resource can be used to specify which item in the pulldown menu is the current choice. The XmNmnemonic and XmNmnemonicCharSet resources can be set to specify a mnemonic for the OptionMenu.

An OptionMenu can be created using `XmCreateOptionMenu`(). In this case, the OptionMenu does not automatically create its submenu; it must be added by the application.

An OptionMenu can also be created by XmCreateSimpleOptionMenu(), which automatically creates the OptionMenu and its submenu with the specified children. This routine uses the RowColumn resources associated with the creation of simple menus. For an OptionMenu, the only types allowed in the XmNbuttonType resource are XmCASCADEBUTTON, XmPUSHBUTTON, XmSEPARATOR, and XmDOUBLE_SEPARATOR. The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 to 1 less than the number of buttons in the submenu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 to 1 less than the number of separators in the submenu.

Default Resource Values

An OptionMenu sets the following default values for RowColumn resources:

Name	Default
XmNmenuPost	BSelect Press
XmNorientation	XmHORIZONTAL
XmNrowColumnType	XmMENU_OPTION

Widget Hierarchy

When an OptionMenu is created, the LabelGadget is named OptionLabel and the CascadeButtonGadget is named OptionButton.

See Also

`XmCreateObject`(1), `XmOptionButtonGadget`(1), `XmOptionLabelGadget`(1), `XmVaCreateSimpleOptionMenu`(1), `XmCascadeButtonGadget`(2), `XmLabelGadget`(2), `XmRowColumn`(2).

Name

XmOptionMenu – a type of RowColumn widget used as an option menu.

Synopsis

Public Header:
<Xm/RowColumn.h>

Class Name:
XmRowColumn

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmRowColumn

Class Pointer:
xmRowColumnWidgetClass

Instantiation:
widget = XmCreateOptionMenu (parent, name,...)

Functions/Macros:
`XmCreateOptionMenu`(), `XmCreateSimpleOptionMenu`(), `XmVaCreateSimpleOptionMenu`(), `XmIsRowColumn`(), `XmOption-ButtonGadget`(), `XmOptionLabelGadget`()

Description

An XmOptionMenu is an instance of a RowColumn widget that is used as a menu that allows a user to select one of several choices. An OptionMenu consists of a label, a selection area, and pulldown menu pane. When you create an OptionMenu, you must supply the pulldown menu pane via the XmNsubMenuId resource. The menu pane must already exist and it must be a child of the OptionMenu's parent. The label (a LabelGadget) and the selection area (a CascadeButtonGadget) are created by the OptionMenu. You can specify the label string with the XmNlabelString resource.

OptionMenu is a RowColumn widget whose XmNrowColumnType resource is set to XmMENU_OPTION. The XmNorientation resource defaults to XmHORIZONTAL, which means that the label is displayed to the left of the selection area. If the resource is set to XmVERTICAL, the label is placed above the selection area. The selection area posts the menu pane, as well as displays the label of the current selection. The XmNmenuPost resource is set to BSelect Press. The XmNmenuHistory resource can be used to specify which item in the pulldown menu is the current choice. The XmNmnemonic and XmNmnemonicCharSet resources can be set to specify a mnemonic for the OptionMenu.

An OptionMenu can be created using XmCreateOptionMenu(). In this case, the OptionMenu does not automatically create its submenu; it must be added by the application.

An OptionMenu can also be created by XmCreateSimpleOptionMenu(), which automatically creates the OptionMenu and its submenu with the specified children. This routine uses the RowColumn resources associated with the creation of simple menus. For an OptionMenu, the only types allowed in the XmNbuttonType resource are XmCASCADEBUTTON, XmPUSHBUTTON, XmSEPARATOR, and XmDOUBLE_SEPARATOR. The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 to 1 less than the number of buttons in the submenu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 to 1 less than the number of separators in the submenu.

Default Resource Values

An OptionMenu sets the following default values for RowColumn resources:

Name	Default
XmNmenuPost	BSelect Press
XmNorientation	XmHORIZONTAL
XmNrowColumnType	XmMENU_OPTION

Widget Hierarchy

When an OptionMenu is created, the LabelGadget is named OptionLabel and the CascadeButtonGadget is named OptionButton.

See Also

XmCreateObject(1), XmOptionButtonGadget(1), XmOptionLabelGadget(1), XmVaCreateSimpleOptionMenu(1), XmCascadeButtonGadget(2), XmLabelGadget(2), XmRowColumn(2).

Name

XmPanedWindow widget class – a constraint widget that tiles its children.

Synopsis

Public Header:
<Xm/PanedW.h>

Class Name:
XmPanedWindow

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmPanedWindow

Class Pointer:
xmPanedWindowWidgetClass

Instantiation:
widget = `XmCreatePanedWindow` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmPanedWindowWidgetClass,...)

Functions/Macros:
`XmCreatePanedWindow`(), `XmIsPanedWindow`()

Description

PanedWindow is a constraint widget that tiles its children. In Motif 1.1, the children are laid out vertically from top to bottom, in the order that they are added to the PanedWindow. In Motif 1.2, the position of each child is controlled by the XmNpositionIndex resource. A PanedWindow is as wide as its widest child and all children are made that width. Users can adjust the height of a pane using a sash that appears below the corresponding pane.

In Motif 2.0 and later, the PanedWindow may be oriented either vertically or horizontally. When the XmNorientation resource is set to XmHORIZONTAL, the PanedWindow is as tall as its tallest child, and all children are made that height. The sash in this orientation is used to control the width of the pane.

New Resources

PanedWindow defines the following resources:

Name	Class	Type	Default	Access
XmNmarginHeight	XmCMarginHeight	Dimension	3	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	3	CSG
XmNorientation	XmCOrientation	unsigned char	XmVERTICAL	CSG
XmNrefigureMode	XmCBoolean	Boolean	True	CSG
XmNsashHeight	XmCSashHeight	Dimension	10	CSG

Name	Class	Type	Default	Access
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
The spacing between a PanedWindow widget's top or bottom edge and any child widget.

XmNmarginWidth
The spacing between a PanedWindow widget's right or left edge and any child widget.

XmNorientation
In Motif 2.0 and later, the orientation of the PanedWindow. Possible values:

XmHORIZONTAL
XmVERTICAL

XmNrefigureMode
If True (default), children are reset to their appropriate positions following a change in the PanedWindow widget.

XmNsashHeight

XmNsashWidth
The height and width of the sash.

XmNsashIndent
If the PanedWindow orientation is XmVERTICAL, the resource specifies the horizontal position of the sash along each pane. Positive values specify the indent from the left edge; negative values, from the right edge (assuming the default values of XmNstringDirection and XmNlayoutDirection). Similarly, in an XmHORIZONTAL PanedWindow, it specifies the vertical position of the sash, positive values being calculated from the top edge, negative values from the bottom. If the value is too large, the sash is placed flush with the edge of the PanedWindow.

XmNsashShadowThickness
The thickness of shadows drawn on each sash. In Motif 2.0 and earlier, the default is 2. In Motif 2.1 and later, the default depends upon the XmDisplay XmNenableThinThickness resource: if True, the default is 1, otherwise 2.

XmNseparatorOn
If True, the widget places a Separator or SeparatorGadget between each pane.

XmNspacing
The distance between each child pane.

New Constraint Resources

PanedWindow defines the following constraint resources for its children:

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
: If False (default), the PanedWindow widget always refuses resize requests from its children. If True, the PanedWindow widget tries to grant requests to change a child's height.

XmNpaneMaximum

XmNpaneMinimum
: The values of a pane's maximum and minimum dimensions for resizing. You can prevent a sash from being drawn by setting these values to be equal.

XmNpositionIndex
: In Motif 1.2, the position of the widget in the PanedWindow's list of children, not including sashes. A value of 0 indicates the beginning of the list, while XmLAST_POSITION places the child at the end of the list.

XmNskipAdjust
: If False (default), the PanedWindow widget automatically resizes this pane child. If True, resizing is not automatic, and the PanedWindow may choose to skip the adjustment of this pane.

Inherited Resources

PanedWindow inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. PanedWindow sets the default value of XmNshadowThickness to 2. The default value of XmNborderWidth is reset to 0 by Manager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Widget Hierarchy

The sash children of a PanedWindow are created using a private widget class called XmSash, derived from XmPrimitive. Each instance of a sash has the name Sash. The appearance of the sash can be configured using these names in external resource files.

Translations

The translations for PanedWindow are inherited from Manager. Additional translations are defined for sashes within a PanedWindow widget:

Event	Action
BSelect Press	SashAction(Start)
BSelect Motion	SashAction(Move)
BSelect Release	SashAction(Commit)
BTransfer Press	SashAction(Start)
BTransfer Motion	SashAction(Move)
BTransfer Release	SashAction(Commit)
KHelp	Help()
KUp	SashAction(Key,DefaultIncr,Up)
MCtrl KUp	SashAction(Key,LargeIncr,Up)
KDown	SashAction(Key,DefaultIncr,Down)
MCtrl KDown	SashAction(Key,LargeIncr,Down)
KLeft	SashAction(Key,DefaultIncr,Left)
MCtrl KLeft	SashAction(Key,LargeIncr,Left)
KRight	SashAction(Key,DefaultIncr,Right)
MCtrl KRight	SashAction(Key,LargeIncr,Right)
KNextField	NextTabGroup()
KPrevField	PrevTabGroup()

Action Routines

PanedWindow defines the following action routines:

Help()

Invokes the list of callbacks specified by XmNhelpCallback. If the PanedWindow doesn't have any help callbacks, the Help() routine invokes those associated with the nearest ancestor that has them.

NextTabGroup()

Traverses to the next tab group. Normally a tab group consists of a pane and its sash.

PrevTabGroup()

Traverses to the previous tab group. Normally a tab group consists of a pane and its sash.

SashAction(action)

Controls the interactive placement of the sash using the mouse. action can have one of three values:

StartBegins the placement operation.
MoveCauses the sash to move as the mouse moves.
CommitEnds the placement operation.

SashAction(Key,increment,direction)

Controls the placement of the sash when it is moved using the keyboard. increment is either DefaultIncr, which moves the sash's position by one line or LargeIncr, which moves the sash's position by one viewing region. direction is either Up, Down, Left, or Right.

Additional Behavior

PanedWindow has the following additional behavior:

<FocusIn>

Highlights the sash and gives it keyboard focus.

<FocusOut>

Unhighlights the sash and removes its keyboard focus.

See Also

`XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmManager`(2).

Name

XmParseMapping data type – an opaque type representing an entry in a parse table

Synopsis

Public Header:
<Xm/Xm.h>

Functions/Macros:
`XmParseMappingCreate`(), `XmParseMappingFree`(), `XmParseMappingGetValues`(), `XmParseMappingSetValues`(),

Availability

Motif 2.0 and later.

Description

XmParseMapping is an opaque data type representing an entry in a parse table, which is used for table-driven parsing of strings and compound strings.

A parse mapping consists of a match pattern, and either a substitution pattern or parse procedure, which can be used by string manipulation functions in order to compare against and subsequently transform text.

An XmParseTable is simply an array of parse mappings. `XmParseMappingCreate`() creates a parse mapping, for subsequent use in a parse table, using a resource style parameter list. The parse table can be passed to `XmStringParseText`(), for example, in order to filter or modify an input string. In the parse process, a pointer is initialized to the head of some input text. The parse table is inspected from top to bottom, comparing the match pattern of each parse mapping entry with the value at the input pointer. Where a correspondence is found, the parse mapping is used to supply transformed output at that point in the parsing process. The input pointer is subsequently advanced, and the process is reiterated.

The implementation of XmParseMapping is that of a pseudo widget: although not a real widget, the object has resources and a resource style interface for setting and fetching values of the mapping, principally the match pattern, substitution pattern, and parse procedure. Resources of the object are set and fetches through the procedures `XmParseMappingGetValues`() and `XmParseMappingSetValues`() respectively.

New Resources

ParseMapping defines the following resources:

Name	Type	Default	Access
XmNclientData	XtPointer	NULL	CSG
XmNincludeStatus	XmIncludeStatus	XmINSERT[a]	CSG
XmNinvokeParseProc	XmParseProc	NULL	CSG
XmNpattern	XtPointer	NULL	CSG
XmNpatternType	XmTextType	XmCHARSET_TEXT	CSG
XmNsubstitute	XmString	NULL	CSG

a.Erroneously given as XmInsert in 2nd edition.

XmNclientData
Specifies application data passed to the parse procedure associated with the XmNinvokeParseProc resource.

XmNincludeStatus
Specifies the way in which the result of the mapping is constructed. Possible values:

```
XmINSERT      /* concatenate XmNsubstitute value to output   */
              /*     parsing is continued                    */
XmINVOKE      /* result determined by XmNinvokeParseProc     */
XmTERMINATE   /* concatenate XmNsubstitute value to output   */
              /*     parsing is terminated                   */
```

XmNinvokeParseProc
Specifies a procedure for determining the result of the mapping. The procedure is only used if XmNincludeStatus is XmINVOKE. An XmParseProc routine places the result of the mapping into the address specified by its *str_include* parameter, and returns either XmINSERT or XmTERMINATE, depending upon whether parsing is to continue. A full description of the format of an XmParseProc is given below.

XmNpattern
Specifies a pattern to be matched against the input being parsed. The pattern is a maximum of one character.

XmNpatternType
The type of the value specified as value for the resource XmNpattern. Possible values:

XmMULTIBYTE_TEXT
XmWIDECHAR_TEXT
XmCHARSET_TEXT

XmNsubstitute
Specifies a compound string to be added to the result of the parse process.

Procedures

The XmParseProc has the following format:

```
typedef XmIncludeStatus (*XmParseProc) (    XtPointer *,
                                            XtPointer,
                                            XmTextType,
                                            XmStringTag,
                                            XmParseMapping,
                                            int,
                                            XmString *,
                                            XtPointer)

    XtPointer       *in_out;        /* text being parsed                  */
    XtPointer       text_end;       /* pointer to end of the text         */
    XmTextType      type;           /* type of text                       */
    XmStringTag     locale_tag;     /* type to be used for the result     */
    XmParseMapping  entry;          /* parse mapping triggering callback  */
    int             pattern_length; /* number of bytes in input text      */
    XmString        *str_include;   /* returned result of the parse       */
    XtPointer       call_data;      /* application data                   */
```

in_out initially points to the current location within the text being parsed. The pointer can be changed in order to reset the location from which to continue parsing after the callback finishes.

text_end points to the end of the in_out string. A parse procedure can set the value of the element to indicate where the parse is to continue from after the mapping has been applied to the input.

type is the type of the text *in_out*, and the type of the *locale_tag* to be used in creating the return compound string. *type* is one of XmCHARSET_TEXT, XmMULTIBYTE_TEXT, or XmWIDECHAR_TEXT.

locale_tag specifies the tag to be used in creating the result. If NULL, the tag created depends upon the value of *type*. If *type* is XmCHARSET_TEXT, a charset string tag is created from the value XmSTRING_DEFAULT_CHARSET. Otherwise, a locale string tag is created from the value _MOTIF_DEFAULT_LOCALE.

entry points to the XmParseMapping object which triggered the callback.

pattern_length is the number of bytes in the input text remaining at the address specified by *in_out*.

str_include is an address where the parse procedure supplies a compound string which is to be inserted into the result of the parse process.

call_data is a pointer to application data passed through by the string parsing functions which invoke the callback.

See Also

`XmParseMappingCreate`(1), `XmParseMappingFree`(1), `XmParseMappingGetValues`(1), `XmParseMappingSetValues`(1), `XmParseTableFree`(1), `XmStringParseText`(1), `XmStringUnparse`(1).

Name

XmPopupMenu – a type of RowColumn widget used as a popup menu pane.

Synopsis

Public Header:
<Xm/RowColumn.h>

Instantiation:
widget = XmCreatePopupMenu (parent, name,...)

Functions/Macros:
`XmCreatePopupMenu`(), `XmCreateSimplePopupMenu`(), `XmMenuPosition`(), `XmVaCreateSimplePopupMenu`()

Description

An XmPopupMenu is the first menu pane in a popup menu system. All other menu panes in the menu system are pulldown panes. A PopupMenu can contain Labels, Separators, PushButtons, ToggleButtons, CascadeButtons, or the corresponding Gadget equivalents.

A PopupMenu is a RowColumn widget whose XmNrowColumnType resource is set to XmMENU_POPUP. The XmNmenuAccelerator resource is set to KMenu and XmNmenuPost is set to BMenu Press. The XmNpopupEnabled resource controls whether or not keyboard accelerators and mnemonics are enabled for a PopupMenu. A PopupMenu needs to be the child of a MenuShell widget to function properly. Use `XmMenuPosition`() to place a PopupMenu.

A PopupMenu can be created using `XmCreatePopupMenu`(). In this case, the PopupMenu does not automatically contain any components; they are added by the application. The PopupMenu created by this routine is a compound object consisting of a MenuShell widget and a RowColumn child.

A PopupMenu can also be created by `XmCreateSimplePopupMenu`(), which automatically creates the PopupMenu with the specified children and makes it the child of a MenuShell. This routine uses the RowColumn resources associated with the creation of simple menus. For a PopupMenu, any type is allowed in the XmNbuttonType resource. The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 to 1 less than the number of buttons in the menu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 to 1 less than the number of separators in the menu. The name of each title is label_*n*, where *n* is the number of the title, ranging from 0 to 1 less than the number of titles in the menu.

In Motif 2.0 and later, the support for automatic popup menus is extended. The Manager and Primitive classes contain XmNpopupHandlerCallback resources, and the RowColumn installs event handlers which simplifies the posting and choice of a popup menu to display. These are installed if the XmNpopupEnabled resource is set to XmPOPUP_AUTOMATIC or XmPOPUP_AUTOMATIC_RECURSIVE.

Default Resource Values

A PopupMenu sets the following default values for RowColumn resources:

Name	Default
XmNmenuAccelerator	KMenu
XmNmenuPost	BMenu Press
XmNpopupEnabled	XmPOPUP_KEYBOARD
XmNrowColumnType	XmMENU_POPUP

Widget Hierarchy

When a PopupMenu is created with a specified name, the MenuShell is named popup_*name* and the RowColumn is called *name*.

See Also

`XmCreateObject`(1), `XmMenuPosition`(1), `XmVaCreateSimplePopupMenu`(1), `XmMenuShell`(2), `XmRowColumn`(2).

Name

XmPrintShell widget class – a Shell interfacing onto the Xp printing facilities

Synopsis

Public Headers:
<Xm/Print.h>

Class Name:
XmPrintShell

Class Hierarchy:
Core →Composite →Shell →WMShell →VendorShell →TopLevelShell →
ApplicationShell →XmPrintShell

Class Pointer:
xmPrintShellWidgetClass

Instantiation:
widget = `XmPrintSetup` (...)
or
widget = `XtCreatePopupShell` (name, xmPrintShellWidgetClass,...)

Functions/Macros:
`XmPrintSetup`(), `XmPrintToFile`(), `XmPrintPopupPDM`(), `XmRedisplayWidget`(), `XmIsPrintShell`()

Availability

Motif 2.1 and later.

Description

PrintShell is a Shell widget which interfaces to the X11R6 X Print (Xp) extensions in order to print out a widget hierarchy. The X Printing Architecture reuses the code which renders the contents and visuals of a widget on the video screen in order to print the widget hierarchy. The technique involves creating instances of requisite widgets within the X Print Server by adding the hierarchy below a PrintShell, and configuring the widgets with suitable resources: the expose methods of the widgets perform the printing. The PrintShell itself is created through `XmPrintSetup`(), which returns an XmPrintShell widget after establishing a connection to an X Print Server. The PrintShell provides resources for configuring an XPrint connection and for specifying printer attributes. In addition, callback resources are available for handling any pagination. If no XPrint connection can be established, PrintShell behaves like an ApplicationShell, from which it is derived.

The Print model allows for either synchronous or asynchronous printing. `XmRedisplayWidget`() provides synchronous printing by forcing a widget to print itself directly. Asynchronous printing is performed through widget exposure as

a result of events generated by the X Print Server and dispatched to the PrintShell: the programmer calls XpStartJob() to initialize the printing process; XmNstartJobCallback, XmNpageSetupCallback, and XmNendJobCallback resources of the PrintShell specify callbacks to provide asynchronous notification at critical points in the printing task.

Whether printing synchronously or asynchronously, an application creates a widget hierarchy on the PrintShell suitable for the required output. What is meant by suitable in this context is application and widget specific: typically, a Text widget is created under the PrintShell, and resources are set to contain the data to print. It may be appropriate to turn off highlighting or blinking cursors in the Text, or to set the background white. This depends upon whether the intention is to print out just the content of the widget, or whether the programmer intends a screen shot which involves the widget visuals.

As a related topic, the procedure XmPrintSetupPDM() requests that a print dialog manager (PDM) is started. The status of the connection to the PDM is monitored by specifying a procedure for the XmNpdmNotificationCallback resource of the PrintShell. Lastly, an additional printing interface is available through the function XmPrintToFile(), which fetches the data on the Print Server, and sends this to a file.

New Resources

PrintShell defines the following resources:

Name	Class	Type	Default	Access
XmNdefaultPixmapResolution	XmCDefaultPixmapResolution	unsigned short	100	CSG
XmNmaxX	XmCMaxX	Dimension	dynamic	G
XmNmaxY	XmCMaxY	Dimension	dynamic	G
XmNminX	XmCMinX	Dimension	dynamic	G
XmNminY	XmCMinY	Dimension	dynamic	G

XmNdefaultPixmapResolution

Indicates the resolution in dots per inch used for scaling image files read by descendants of the widget. In general, the resource is not used when reading tile pixmaps (XmNbackgroundPixmap, (XmNbottomShadowPixmap, or similar).

XmNmaxX

XmNmaxY

XmNminX

XmNminY

Specifies the image area of the page in the current print context. The PrintShell maintains the values to reflect change in resolution or other attributes.

Callback Resources

PrintShell defines the following callback resources:

Callback	Reason Constant
XmNendJobCallback	XmCR_END_JOB
XmNpageSetupCallback	XmCR_PAGE_SETUP
XmNpdmNotificationCallback	XmCR_PDM_NONE XmCR_PDM_UP XmCR_PDM_START_ERROR XmCR_PDM_START_VXAUTH XmCR_PDM_START_PXAUTH XmCR_PDM_OK XmCR_PDM_CANCEL XmCR_PDM_EXIT_ERROR
XmNstartJobCallback	XmCR_START_JOB

XmNendJobCallback
Specifies the list of callbacks called to control the end of rendering.

XmNpageSetupCallback
Specifies the list of callbacks called to control page layout.

XmNpdmNotificationCallback
Specifies the list of callbacks called in notification of the Print Dialog Manager (PDM) status.

XmNstartJobCallback
Specifies the list of callbacks called to control the start of rendering.

Callback Structure

Each callback is passed a pointer to the following structure:

```
typedef struct {
    int         reason;     /* reason that the callback is invoked      */
    XEvent      *event;     /* event structure that triggered callback */
    XPContext   context;    /* X Print Context                          */
    Boolean     last_page;  /* whether this is the last page            */
    XtPointer   detail;     /* PDM selection                            */
} XmPrintShellCallbackStruct;
```

reason specifies the cause of the callback invocation. The value is XmCR_START_JOB for any callback on the XmNstartJobCallback list, XmCR_END_JOB for XmNendJobCallback procedures, and XmCR_PAGE_SETUP for XmNpageSetupCallback routines. When a XmNpd-

mNotificationCallback procedure is invoked, *reason* indicates the status of the PDM connection. Possible values:

```
XmCR_PDM_CANCEL          /* PDM exited with CANCEL status        */
XmCR_PDM_EXIT_ERROR      /* PDM exited with ERROR status         */
XmCR_PDM_NONE            /* No PDM available on the display      */
XmCR_PDM_OK              /* PDM exited with OK status            */
XmCR_PDM_START_ERROR     /* PDM cannot start                     */
XmCR_PDM_START_VXAUTH    /* PDM cannot connect to video display */
XmCR_PDM_START_PXAUTH    /* PDM cannot connect to print display */
XmCR_PDM_UP              /* PDM is running                       */
```

context is an opaque handle representing the connection to the Print Server.

last_page is only of relevance within an XmNpageSetupCallback procedure, and it is a flag whereby the application indicates that this is the last page of the printing task. The value is initially set to False by the toolkit, and the application callback should set the value True as required. Thereafter, the XmNpageSetupCallback procedures are invoked once more with the value initially True, by way of return notification, and subsequently the XmNendJobCallback procedures are called.

detail is only used by XmNpdmNotificationCallback procedures, and it contains an Atom representing the selection used in connecting to the PDM. The *reason* element is XmCR_PDM_NONE[1] when detail is set.

Inherited Resources

PrintShell inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of XmNborderWidth is reset to 0 by VendorShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNlayoutDirection	VendorShell
XmNallowShellResize	Shell	XmNmappedWhenManaged	Core
XmNancestorSensitive	Core	XmNmaxAspectX	WMShell
XmNargc	ApplicationShell	XmNmaxAspectY	WMShell
XmNargv	ApplicationShell	XmNmaxHeight	WMShell
XmNaudibleWarning	VendorShell	XmNmaxWIdth	WMShell
XmNbackground	Core	XmNminAspectX	WMShell
XmNbackgroundPixmap	Core	XmNminAspectY	WMShell

1.Erroneously given as XmPDM_NONE in 2nd edition.

Resource	Inherited From	Resource	Inherited From
XmNbaseHeight	WMShell	XmNminHeight	WMShell
XmNbaseWidth	WMShell	XmNminWidth	WMShell
XmNborderColor	Core	XmNmwmDecorations	VendorShell
XmNborderPixmap	Core	XmNmwmFunctions	VendorShell
XmNborderWidth	Core	XmNmwmInputMode	VendorShell
XmNbuttonFontList	VendorShell	XmNmwmMenu	VendorShell
XmNbuttonRenderTable	VendorShell	XmNnumChildren	Composite
XmNchildren	Composite	XmNoverrideRedirect	Shell
XmNcolormap	Core	XmNpopdownCalback	Shell
XmNcreatePopupChildProc	Shell	XmNpopupCallback	Shell
XmNdefaultFontList	VendorShell	XmNpreeditType	VendorShell
XmNdeleteResponse	VendorShell	XmNsaveUnder	Shell
XmNdepth	Core	XmNscreen	Core
XmNdestroyCallback	Core	XmNsensitive	Core
XmNgeometry	Shell	XmNshellUnitType	VendorShell
XmNheight	Core	XmNtextFontList	VendorShell
XmNheightInc	WMShell	XmNtextRenderTable	VendorShell
XmNiconic	TopLevelShell	XmNtitle	WMShell
XmNiconMask	WMShell	XmNtitleEncoding	WMShell
XmNiconName	TopLevelShell	XmNtransient	WMShell
XmNiconNameEncoding	TopLevelShell	XmNtranslations	Core
XmNiconPixmap	WMShell	XmNuseAsyncGeometry	VendorShell
XmNiconWindow	WMShell	XmNunitType	VendorShell
XmNinitialResourcesPersistent	Core	XmNvisual	Shell
XmNinitialState	WMShell	XmNwaitForWm	WMShell
XmNinput	WMShell	XmNwidth	Core
XmNinputMethod	VendorShell	XmNwidthInc	WMShell
XmNinputPolicy	VendorShell	XmNwindowGroup	WMShell
XmNinsertPosition	Composite	XmNwinGravity	WMShell
XmNkeyboardFocusPolicy	VendorShell	XmNwmTimeout	WMShell
XmNlabelFontList	VendorShell	XmNx	Core
XmNlabelRenderTable	VendorShell	XmNy	Core

See Also

XmGetScaledPixmap(1), XmPrintPopupPDM(1), XmPrintToFile(1), XmPrintSetup(1), XmRedisplayWidget(1), ApplicationShell(2), Composite(2), Core(2), Shell(2), TopLevelShell(2), VendorShell(2), WMShell(2).

Name

XmPromptDialog – an unmanaged SelectionBox as a child of a Dialog Shell.

Synopsis

Public Header:
<Xm/SelectioB.h>

Instantiation:
widget = XmCreatePromptDialog(...)

Functions/Macros:
`XmCreatePromptDialog`(), `XmSelectionBoxGetChild`(), `XmIsSelectionBox`()

Description

An XmPromptDialog is a compound object created by a call to XmCreatePromptDialog() that an application can use to prompt the user for textual input. A PromptDialog consists of a DialogShell with an unmanaged SelectionBox widget as its child. The SelectionBox resource XmNdialogType is set to XmDIALOG_PROMPT.

A PromptDialog contains a message, a region for text input, and three managed buttons. A fourth button is created but not managed; you can manage it explicitly if necessary. In Motif 1.2 and later, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Apply**, **Cancel**, and **Help** by default. The **Apply** button is the unmanaged button.

Default Resource Values

A PromptDialog sets the following default values for SelectionBox resources:

Name	Default
XmNdialogType	XmDIALOG_PROMPT
XmNlistLabelString	NULL
XmNlistVisibleItemCount	0

Widget Hierarchy

When a PromptDialog is created with a specified name, the DialogShell is named name*_popup* and the SelectionBox is called *name***.**

See Also

`XmCreateObject`(1), `XmSelectionBoxGetChild`(1), `XmDialogShell`(2), `XmSelectionBox`(2).

Name

XmPulldownMenu – a type of RowColumn used as a pulldown menu pane.

Synopsis

Public Header:
<Xm/RowColumn.h>

Instantiation:
widget = XmCreatePulldownMenu (parent, name,...)

Functions/Macros:
`XmCreatePulldownMenu`(), `XmCreateSimplePulldownMenu`(), `XmVaCreateSimplePulldownMenu`()

Description

An XmPulldownMenu is a menu pane for all types of pulldown menu systems, including menus off of a menu bar, cascading submenus, and the menu associated with an option menu. A PulldownMenu is associated with a CascadeButton. A PulldownMenu can contain Separators, PushButtons, ToggleButtons, and CascadeButtons.

A PulldownMenu is a RowColumn widget whose XmNrowColumnType resource is set to XmMENU_PULLDOWN. A PulldownMenu needs to be the child of a MenuShell widget to function properly.

A PulldownMenu can be created using `XmCreatePulldownMenu`(). In this case, the PulldownMenu does not automatically contain any components; they are added by the application. The PulldownMenu created by this routine is a compound object consisting of a MenuShell widget and a RowColumn child.

A PulldownMenu can also be created by `XmCreateSimplePulldown-Menu`(), which automatically creates the PulldownMenu with the specified children and makes it the child of a MenuShell. This routine uses the RowColumn resources associated with the creation of simple menus. For a PulldownMenu, any type is allowed in the XmNbuttonType resource. The name of each button is button_*n*, where *n* is the number of the button, ranging from 0 to 1 less than the number of buttons in the menu. The name of each separator is separator_*n*, where *n* is the number of the separator, ranging from 0 to 1 less than the number of separators in the menu. The name of each title is label_*n*, where *n* is the number of the title, ranging from 0 to 1 less than the number of titles in the menu.

Default Resource Values

A PulldownMenu sets the following default values for RowColumn resources:

Name	Default
XmNrowColumnType	XmMENU_PULLDOWN[a]

a.Erroneously given as XmMENU_POPUP in 2nd edition.

Widget Hierarchy

When a PulldownMenu is created with a specified name, the MenuShell is named popup_*name* and the RowColumn is called *name*.

See Also

`XmCreateObject`(1), `XmVaCreateSimplePulldownMenu`(1), `XmCascadeButton`(2), `XmMenuShell`(2), `XmRowColumn`(2).

Name

XmPushButton widget class – a widget that starts an operation when it is pressed.

Synopsis

Public Header:
<Xm/PushB.h>

Class Name:
XmPushButton

Class Hierarchy:
Core →XmPrimitive →XmLabel →XmPushButton

Class Pointer:
xmPushButtonWidgetClass

Instantiation:
widget = `XmCreatePushButton` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmPushButtonWidgetClass,...)

Functions/Macros:
`XmCreatePushButton`(), `XmIsPushButton`()

Description

A PushButton is a widget which, when pressed by the user, issues a logical event to the application. A PushButton displays a text or pixmap label. It invokes an application callback when it is clicked on with the mouse. The shading of the PushButton changes to make it appear either pressed in when selected or raised when unselected.

Traits

PushButton holds the XmQTactivatable, XmQTtakesDefault, XmQT-careParentVisual and XmQTmenuSavvy traits, which are inherited by any derived classes, and uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

New Resources

PushButton defines the following resources:

Name	Class	Type	Default	Access
XmNarmColor	XmCArmColor	Pixel	dynamic	CSG
XmNarmPixmap	XmCArmPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNdefaultButtonShadowThickness	XmCDefaultButtonShadowThickness	Dimension	dynamic	CSG
XmNfillOnArm	XmCFillOnArm	Boolean	True	CSG
XmNmultiClick	XmCMultiClick	unsigned char	dynamic	CSG
XmNshowAsDefault	XmCShowAsDefault	Dimension	0	CSG

XmNarmColor
The color with which the armed button is filled. For a color display, the default color is a shade between the bottom shadow color and the background color. For a monochrome display, the default is the foreground color, and label text is switched to the background color. This resource is in effect only when XmNfillOnArm is set to True.

XmNarmPixmap
The pixmap that identifies the button when it is armed (and when its XmNlabelType is XmPIXMAP). For a PushButton in a menu, this resource is disabled.

XmNdefaultButtonShadowThickness
The width of the shadow used to indicate a default PushButton.

XmNfillOnArm
If True (default), the PushButton widget fills the button (when armed) with the color specified by XmNarmColor. If False, the PushButton widget only switches the top and bottom shadow colors. For a PushButton in a menu, this resource is disabled (and assumed to be False).

XmNmultiClick
A flag that determines whether successive button clicks are processed or ignored. Possible values:

```
XmMULTICLICK_DISCARD    /* ignore successive button clicks;  */
                        /* default value in a menu system    */
XmMULTICLICK_KEEP       /* count successive button clicks;   */
                        /* default value when not in a menu */
```

XmNshowAsDefault
Indicates the default PushButton by displaying a shadow. (In a menu, this resource is disabled.) This resource works in different ways:

If the width of the shadow is already specified through the resource XmNdefaultButtonShadowThickness, then XmNshowAsDefault behaves like a Boolean: that is, with a value of 0, no shadow is displayed; with a value greater than 0, a shadow is displayed.

If the width of the shadow has not been specified through the resource XmNdefaultButtonShadowThickness (i.e., it has a value of 0), then XmNshowAsDefault performs double duty: that is, a value greater than 0 says to highlight the PushButton as the default button and to use this value as the thickness of the shadow.

Callback Resources

PushButton defines the following callback resources:

Callback	Reason Constant
XmNactivateCallback	XmCR_ACTIVATE
XmNarmCallback	XmCR_ARM
XmNdisarmCallback	XmCR_DISARM

XmNactivateCallback
List of callbacks that are called when BSelect is pressed and released inside the widget.

XmNarmCallback
List of callbacks that are called when BSelect is pressed while the pointer is inside the widget.

XmNdisarmCallback
List of callbacks that are called when BSelect is released after it has been pressed inside the widget.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* the reason that the callback was called */
    XEvent      *event;         /* event structure that triggered callback  */
    int         click_count;    /* number of multi-clicks                   */
} XmPushButtonCallbackStruct;
```

click_count is meaningful only for XmNactivateCallback. Furthermore, if the XmN-multiClick resource is set to XmMULTICLICK_KEEP, then XmN-activate--Callback is called for each click, and the value of *click_count* is the number of clicks that have occurred in the last sequence of multiple clicks. If the XmNmultiClick resource is set to XmMULTICLICK_DISCARD, then *click_count* always has a value of 1.

Inherited Resources

PushButton inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. PushButton sets the default values of XmNmarginBottom, XmNmarginLeft, XmNmarginRight, and XmNmarginTop dynamically based on the value of XmNshowAsDefault. If XmNarmPixmap is specified but XmNlabelPixmap is not, the default value of XmNlabelPixmap is set to the value of XmNarmPixmap. The default value of XmNborderWidth is reset to 0 by Primitive. In Motif 2.0 and earlier, the default

values of XmNhighlightThickness and XmNshadowThickness are reset to 2. In Motif 2.1, the default values depend upon the XmDisplay XmNenableThin-Thickness resource: if True the default is 1, otherwise 2.

Resource	Inherited From	Resource	Inherited From
XmNaccelerator	XmLabel	XmNlabelType	XmLabel
XmNaccelerators	Core	XmNlayoutDirection	XmPrimitive
XmNacceleratorText	XmLabel	XmNmappedWhenManaged	Core
XmNalignment	XmLabel	XmNmarginBottom	XmLabel
XmNancestorSensitive	Core	XmNmarginHeight	XmLabel
XmNbackground	Core	XmNmarginLeft	XmLabel
XmNbackgroundPixmap	Core	XmNmarginRight	XmLabel
XmNborderColor	Core	XmNmarginTop	XmLabel
XmNborderPixmap	Core	XmNmarginWidth	XmLabel
XmNborderWidth	Core	XmNmnemonicCharSet	XmLabel
XmNbottomShadowColor	XmPrimitive	XmNmnemonic	XmLabel
XmNbottomShadowPixmap	XmPrimitive	XmNnavigationType	XmPrimitive
XmNcolormap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNrecomputeSize	XmLabel
XmNdepth	Core	XmNrenderTable	XmLabel
XmNdestroyCallback	Core	XmNscreen	Core
XmNfontList	XmLabel	XmNsensitive	Core
XmNforeground	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNheight	Core	XmNstringDirection	XmLabel
XmNhelpCallback	XmPrimitive	XmNtopShadowColor	XmPrimitive
XmNhighlightColor	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNhighlightOnEnter	XmPrimitive	XmNtranslations	Core
XmNhighlightPixmap	XmPrimitive	XmNtraversalOn	XmPrimitive
XmNhighlightThickness	XmPrimitive	XmNunitType	XmPrimitive
XmNinitialResourcesPersistent	Core	XmNuserData	XmPrimitive
XmNlabelInsensitivePixmap	XmLabel	XmNwidth	Core
XmNlabelPixmap	XmLabel	XmNx	Core
XmNlabelString	XmLabel	XmNy	Core

Translations

For PushButtons outside a Menu System:

Event	Action
MCtrl BSelect Press	ButtonTakeFocus()
BSelect Press	Arm()
BSelect Click	Activate() Disarm()
BSelect Release	Activate() Disarm()
BSelect Press 2+	MultiArm()
BSelect Release 2+	MultiActivate() Disarm()
BTransfer Press	ProcessDrag()
KSelect	ArmAndActivate()
KHelp	Help()

For PushButtons in a Menu System:

Event	Action
MCtrl BSelect Press	MenuButtonTakeFocus()
BSelect Press	BtnDown()
BMenu Press	BtnDown()
BMenu Release	BtnUp()
KActivate	ArmAndActivate()
KSelect	ArmAndActivate()
MAny KCancel	MenuShellPopdownOne() (1.2)
MAny KCancel	MenuEscape (2.0)

Action Routines

PushButton defines the following action routines:

Activate()

Displays the PushButton as unarmed, and invokes the list of callbacks specified by XmNactivateCallback. The button's appearance may depend on the values of the resources XmNfillOnArm and XmNlabelPixmap.

Arm()

Displays the PushButton as armed, and invokes the list of callbacks specified by XmNarmCallback. The button's appearance may depend on the values of the resources XmNarmColor and XmNarmPixmap.

ArmAndActivate()

When the PushButton is in a menu, this action unposts the menu hierarchy and invokes the callbacks specified by the resources XmNarmCallback, XmNactivateCallback, and finally, XmNdisarmCallback.

When the PushButton is not in a menu, this action displays the PushButton as armed (as determined by the values of the resources XmNarmColor and XmNarmPixmap) and (assuming the button is not yet armed) invokes the list of callbacks specified by XmNarmCallback. After this occurs, the action displays the PushButton as unarmed and invokes the callbacks specified in XmNactivateCallback and XmNdisarmCallback.

BtnDown()

Unposts any menus that were posted by the parent menu of the PushButton, changes from keyboard traversal to mouse traversal, displays the PushButton as armed, and (assuming the button is not yet armed) invokes the callbacks specified by XmNarmCallback.

BtnUp()

Unposts the menu hierarchy, activates the PushButton, and invokes first the callbacks specified by XmNactivateCallback and then those specified by XmNdisarmCallback.

ButtonTakeFocus()

In Motif 2.0 and later, moves the current keyboard focus to the PushButton, without activating the widget.

Disarm()

Invokes the callbacks specified by XmNdisarmCallback.

Help()

Unposts the menu hierarchy, restores the previous keyboard focus, and invokes the callbacks specified by the XmNhelpCallback resource.

MenuButtonTakeFocus()

In Motif 2.0 and later, moves the current keyboard focus to the PushButton, without activating the widget.

MenuShellPopdownOne()
: In Motif 1.2 and earlier, unposts the current menu and (unless the menu is a pulldown submenu) restores keyboard focus to the tab group or widget that previously had it. In a top-level pulldown menu pane attached to a menu bar, this action routine also disarms the cascade button and the menu bar.

MenuEscape()
: In Motif 2.0 and later, invokes the popdownOne method of the MenuShell class, which unposts the current menu, restores keyboard focus, and ungrabs the pointer where necessary. Any containing menu row column is disarmed.

MultiActivate()
: Increments the click_count member of the XmPushButtonCallbackStruct, displays the PushButton as unarmed (as determined by the resources XmNfillOnArm and XmNlabelPixmap), and invokes first the callbacks specified by XmNactivateCallback and then those specified by XmNdisarmCallback. This action routine takes effect only when the XmNmultiClick resource is set to XmMULTICLICK_KEEP.

MultiArm()
: Displays the PushButton as armed (as determined by the resources XmNarmColor and XmNarmPixmap) and invokes the list of callbacks specified by XmNarmCallback. This action routine takes effect only when the XmNmultiClick resource is set to XmMULTICLICK_KEEP.

ProcessDrag()
: In Motif 1.2 and later, initiates a drag and drop operation using the label of the PushButton.

Additional Behavior

PushButton has the following additional behavior:

<EnterWindow>
: Displays the PushButton as armed.

<LeaveWindow>
: Displays the PushButton as unarmed.

See Also

`XmCreateObject`(1), `Core`(2), `XmLabel`(2), `XmPrimitive`(2).

Name

XmPushButtonGadget widget class – a gadget that starts an operation when it is pressed.

Synopsis

Public Header:
<Xm/PushBG.h>

Class Name:
XmPushButtonGadget

Class Hierarchy:
Object →RectObj →XmGadget →XmLabelGadget →XmPushButtonGadget

Class Pointer:
xmPushButtonGadgetClass

Instantiation:
widget = `XmCreatePushButtonGadget` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmPushButtonGadgetClass,...)

Functions/Macros:
`XmCreatePushButtonGadget`(), `XmIsPushButtonGadget`()

Description

PushButtonGadget is the gadget variant of PushButton.

PushButtonGadget's new resources, callback resources, and callback structure are the same as those for PushButton.

Traits

PushButtonGadget holds the XmQTactivatable, XmQTmenuSavvy, and XmQTtakesDefault traits, which are inherited by any derived classes, and uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

Inherited Resources

PushButtonGadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. PushButtonGadget sets the default values of XmNmarginBottom, XmNmarginLeft, XmNmarginRight, and XmNmarginTop dynamically based on the value of XmNshowAsDefault. If XmNarmPixmap is specified but XmNlabelPixmap is not, the default value of XmNlabelPixmap is set to the value of XmNarmPixmap. The default value of XmNborderWidth is reset to 0 by Gadget.

Resource	Inherited From	Resource	Inherited From
XmNancestorSensitive	RectObj	XmNhighlightThickness	XmGadget
XmNbackground	XmGadget	XmNlayoutDirection	XmGadget
XmNbackgroundPixmap	XmGadget	XmNnavigationType	XmGadget
XmNbottomShadowColor	XmGadget	XmNsensitive	RectObj
XmNbottomShadowPixmap	XmGadget	XmNshadowThickness	XmGadget
XmNborderWidth	RectObj	XmNtopShadowColor	XmGadget
XmNdestroyCallback	Object	XmNtopShadowPixmap	XmGadget
XmNforeground	XmGadget	XmNtraversalOn	XmGadget
XmNheight	RectObj	XmNunitType	XmGadget
XmNhelpCallback	XmGadget	XmNuserData	XmGadget
XmNhighlightColor	XmGadget	XmNwidth	RectObj
XmNhighlightOnEnter	XmGadget	XmNx	RectObj
XmNhighlightPixmap	XmGadget	XmNy	RectObj

Behavior

As a gadget subclass, PushButtonGadget has no translations associated with it. However, PushButtonGadget behavior corresponds to the action routines of the PushButton widget. See the PushButton action routines for more information.

For PushButtonGadgets outside of a menu system, the following translations are defined:

Event	Action
MCtrl BSelect Press	ButtonTakeFocus()
BSelect Press	Arm()
BSelect Click	Activate() Disarm()
BSelect Release	Activate() Disarm()
BSelect Press 2+	MultiArm()
BSelect Release 2+	MultiActivate() Disarm()
BTransfer Press	ProcessDrag()
KSelect	ArmAndActivate()
KHelp	Help()

For PushButtonGadgets inside a menu system:

Event	Action
MCtrl BSelect Press	MenuButtonTakeFocus()
BSelect Press	BtnDown()
BMenu Press	BtnDown()
BMenu Release	BtnUp()
KActivate	ArmAndActivate()
KSelect	ArmAndActivate()
MAny KCancel	MenuShellPopdownOne() (1.2)
MAny KCancel	MenuEscape (2.0)

PushButtonGadget has additional behavior associated with <Enter> and <Leave>, which draw the shadow in the armed or unarmed state, respectively.

See Also

`XmCreateObject`(1), `Object`(2), `RectObj`(2), `XmGadget`(2), `XmLabelGadget`(2), `XmPushButton`(2).

Name

XmQuestionDialog – an unmanaged MessageBox as a child of a DialogShell.

Synopsis

Public Header:
<Xm/MessageB.h>

Instantiation:
widget = `XmCreateQuestionDialog` (parent, name,...)

Functions/Macros:
`XmCreateQuestionDialog`(), `XmMessageBoxGetChild`()

Description

An XmQuestionDialog is a compound object created by a call to `XmCreate-QuestionDialog`() that an application can use to ask the user a question. A QuestionDialog consists of a DialogShell with a MessageBox widget as its child. The MessageBox resource XmNdialogType is set to XmDIALOG_QUESTION.

A QuestionDialog includes four components: a symbol, a message, three buttons, and a separator between the message and the buttons. By default, the symbol is a question mark. In Motif 1.2, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Cancel**, and **Help** by default.

Default Resource Values

A QuestionDialog sets the following default values for MessageBox resources:

Name	Default
XmNdialogType	XmDIALOG_QUESTION
XmNsymbolPixmap	xm_question

Widget Hierarchy

When a QuestionDialog is created with a specified name, the DialogShell is named name*_popup* and the MessageBox is called *name*.

See Also

`XmCreateObject`(1), `XmMessageBoxGetChild`(1), `XmDialogShell`(2), `XmMessageBox`(2).

Name

XmRadioBox – a RowColumn that contains ToggleButtons.

Synopsis

Public Header:
<Xm/RowColumn.h>

Class Name:
XmRowColumn

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmRowColumn

Class Pointer:
xmRowColumnWidgetClass

Instantiation:
widget = `XmCreateRadioBox` (parent, name,...)

Functions/Macros:
`XmCreateRadioBox`(), `XmCreateSimpleRadioBox`(), `XmVaCreateSimpleRadioBox`(), `XmIsRowColumn`()

Description

An XmRadioBox is an instance of a RowColumn widget that contains ToggleButtonGadgets, only one of which can be selected at a given time. When a RadioBox is created with `XmCreateRadioBox`(): it does not automatically contain ToggleButtonGadget children; they are added by the application developer. A RadioBox can also be created by a call to `XmCreateSimpleRadioBox`(), which automatically creates the specified ToggleButtonGadget widgets as children.

A RadioBox is a RowColumn widget with its XmNrowColumnType resource set to XmWORK_AREA, XmNpacking set to XmPACK_COLUMN, and XmNradioAlwaysOne set to True, which means that one button is always selected. The XmNradioBehavior resource is set to True. The XmNmenuHistory resource indicates the last ToggleButtonGadget that was selected. The XmNisHomogenous resource is set to True and XmNentryClass is set to ToggleButtonGadget, to ensure that only ToggleButtonGadgets are added as children. RadioBox sets XmNvisibleWhenOff to True and XmNindicatorType to XmONE_OF_MANY for all of its ToggleButtonGadget children.

A RadioBox can be created by making a RowColumn with these resource values. When it is created in this way, a RadioBox does not automatically contain ToggleButtonGadget children; they are added by the application.

A RadioBox can also be created by a call to XmCreateSimpleRadioBox() or XmVaCreateSimpleRadioBox(). These routines automatically create the RadioBox with ToggleButtonGadgets as children. The routines use the RowColumn resources associated with the creation of simple menus. For a RadioBox, the only type allowed in the XmNbuttonType resource is XmRADIOBUTTON. The name of each ToggleButtonGadget is button_*n*, where *n* is the number of the button, ranging from 0 to 1 less than the number of buttons in the RadioBox.

Default Resource Values

A RadioBox sets the following default values for its resources:

Name	Default
XmNentryClass	xmToggleButtonGadgetClass[a]
XmNisHomogenous	True
XmNnavigationType	XmTAB_GROUP
XmNradioAlwaysOne	True
XmNradioBehavior	True
XmNrowColumnType	XmWORK_AREA
XmNtraversalOn	True

a. Erroneously given as xmToggleButtonWidgetClass in 1st and 2nd editions

See Also

`XmCreateObject`(1), `XmVaCreateSimpleRadioBox`(1), `XmRowColumn`(2), `XmToggleButtonGadget`(2).

Name

XmRendition data type – an opaque type representing an entry in a render table

Synopsis

Public Header:
<Xm/Xm.h>

Instantiation:
rendition = XmRenditionCreate (....)

Functions/Macros:
`XmRenditionCreate`(), `XmRenditionFree`(), `XmRenditionRetrieve`(),
`XmRenditionUpdate`().

Availability

Motif 2.0 and later.

Description

XmRendition is an opaque data type used for rendering XmStrings.

A rendition consists of two parts: an XmStringTag, which is matched against tag elements within a compound string to be rendered, and rendering data such as color, font, line style.

The implementation of XmRendition is through a pseudo widget: although not a true widget, the object has resources and a resource style interface for setting and fetching values of the rendition. The object is created by `XmRenditionCreate`(), resources are fetched using `XmRenditionRetrieve`(), and set through `XmRenditionUpdate`(), and finally deallocated by `XmRenditionFree`(). Typically, a rendition forms an entry within a render table, and a rendition is merged into an existing render table, or used as the basis for a new table, through the function `XmRenderTableAddRenditions`(). Compound strings are rendered by successively matching tags within the compound string with tags associated with entries in a render table. Where there is an equivalence, that rendition is used to display the corresponding component of the compound string.

Resources within a rendition may have the value XmAS_IS, either explicitly assigned or as an implicit default. Where the value is XmAS_IS, the value is taken from the previous rendition used to render the compound string. If the previous rendition also has the value XmAS_IS, then the rendition before the previous one is inspected, and so on. A default value is provided if no previous renditions supply a resource value.

Renditions, and the render tables to which they belong, are sharply across widgets, and are reference counted. In the general case, widgets inherit a render table from the nearest ancestor which holds the XmQTspecifyRenderTable trait if their own render table is unspecified. It is important to deallocate a rendition through `XmRenditionFree`(), rather than directly invoking `XtFree`(), in order to maintain the reference count.

New Resources

XmRendition defines the following resources:

Name	Class	Type	Default	Access
XmNfont	XmCFont	XtPointer	XmAS_IS	CSG
XmNfontName	XmCFontName	String	XmAS_IS	CSG
XmNfontType	XmCFontType	XmFontType	XmAS_IS	CSG
XmNloadModel	XmCLoadModel	unsigned char	XmAS_IS	CSG
XmNrenditionBackground	XmCRenditionBackground	Pixel	XmUNSPECIFIED_PIXEL	CSG
XmNrenditionForeground	XmCRenditionForeground	Pixel	XmUNSPECIFIED_PIXEL	CSG
XmNstrikethruType	XmCStrikethruType	unsigned char	XmAS_IS	CSG
XmNtabList	XmCTabList	XmTabList	XmAS_IS	CSG
XmNtag	XmCTag	XmStringTag	""	C
XmNunderlineType	XmCUnderlineType	unsigned char	XmAS_IS	CSG

XmNfont
: Specifies the font for the rendition. If specified, XmNloadModel is forced to XmLOAD_IMMEDIATE. Otherwise, the rendition automatically sets the value of the resource when the font associated with XmNfontName is loaded, either because the rendition is used to render a compound string, or as a side effect of a XmRenditionUpdate() call.

XmNfontName
: Specifies either the name of a font, or a comma-separated list of font names (a font set). Each name is assumed to be in standard X Logical Font Description (XLFD) format. If both XmNfontName and XmNfont are specified for a rendition, the XmNfont resource takes precedence.

XmNfontType
: Specifies whether the font associated with the rendition is a font or a fontset. Possible values:

XmFONT_IS_FONT XmFONT_IS_FONTSET

XmNloadModel
Specifies whether the font or fontset indicated by the XmNfontName resource is loaded when the rendition is created, or when first required. Possible values:

```
XmLOAD_IMMEDIATE        /* load on rendition create   */
XmLOAD_DEFERRED         /* load when font is required */
```

XmNrenditionBackground
Specifies the background color for the rendition.

XmNrenditionForeground
Specifies the foreground color for the rendition.

XmNstrikethruType
Specifies a line type for striking through a text segment. Possible values:

```
XmDOUBLE_DASHED_LINE    XmDOUBLE_LINE
XmSINGLE_DASHED_LINE    XmSINGLE_LINE
XmNO_LINE
```

XmNtabList
Specifies the tab list which specifies how compound strings containing tab components are laid out in columns.

XmNtag
Specifies a tag which is used in matching against XmStringTag components within compound strings, or in matching against other renditions. The value is taken from the tag parameter of the XmRenditionCreate() procedure which creates the rendition object. The value NULL is not valid, although the empty string "" is.

XmNunderlineType
Specifies a line type for underlining a text segment. Possible values:

```
XmDOUBLE_DASHED_LINE    XmDOUBLE_LINE
XmNO_LINE               XmSINGLE_DASHED_LINE
XmSINGLE_LINE
```

See Also

`XmRenditionCreate`(1), `XmRenditionFree`(1), `XmRenditionRetrieve`(1), `XmRenditionUpdate`(1), `XmRenderTableAddRenditions`(1), `XmRenderTableCopy`(1), `XmRenderTableFree`(1), `XmRenderTableGetRendition`(1), `XmRenderTableGetRenditions`(1), `XmRenderTableGetTags`(1), `XmRenderTableRemoveRenditions`(1).

Name

XmRowColumn widget class – a manager widget that arranges its children in rows and columns.

Synopsis

Public Header:
<Xm/RowColumn.h>

Class Name:
XmRowColumn

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmRowColumn

Class Pointer:
xmRowColumnWidgetClass

Instantiation:
widget = `XmCreateRowColumn` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmRowColumnWidgetClass,...)

Functions/Macros:
`XmCreateMenuBar`(), `XmCreateOptionMenu`(), `XmCreatePopupMenu`(), `XmCreatePulldownMenu`(), `XmCreateRadioBox`(), `XmCreateRowColumn`(), `XmCreateSimpleCheckBox`(), `XmCreateSimpleMenuBar`(), `XmCreateSimpleOptionMenu`(), `XmCreateSimplePopupMenu`(), `XmCreateSimplePulldownMenu`(), `XmCreateSimpleRadioBox`(), `XmCreateWorkArea`(), `XmIsRowColumn`(), `XmVaCreateSimpleCheckBox`(), `XmVaCreateSimpleMenuBar`(), `XmVaCreateSimpleOptionMenu`(), `XmVaCreateSimplePopupMenu`(), `XmVaCreateSimplePulldownMenu`(), `XmVaCreateSimpleRadioBox`()

Description

RowColumn provides an area in which children belonging to any widget type are displayed in rows and columns. RowColumn is a general-purpose manager widget class that can be configured into many layouts, such as a MenuBar, PopupMenu, PulldownMenu, OptionMenu, CheckBox, or RadioBox. Many of RowColumn's resources pertain only to a specific layout type.

In Motif 1.2 and later, a RowColumn that is configured as a PopupMenu or a PulldownMenu supports tear off menus. When a menu is torn off, it remains on the screen after a selection is made so that additional selections can be made. A

menu pane that can be torn off contains a tear-off button at the top of the menu. A tear-off button is a button that has a Separator-like appearance. The name of a tear-off button in a menu pane is TearOffControl. An application can set the following resources for a tear-off button: XmNbackground, XmNbackgroundPixmap, XmNbottomShadowColor, XmNforeground, XmNheight, XmNmargin, XmNseparatorType, XmNshadowThickness, and XmNtopShadowColor.

In Motif 2.0 and later, the mechanisms whereby pulldown menus are selected and posted have been rationalized. The Manager and Primitive classes support XmNpopupHandlerCallback resources which can be used to choose a popup menu to display in a given context. In addition, the RowColumn provides automatic posting of popups through extensions to the XmNpopupEnabled resource.

Traits

RowColumn holds the XmQTmenuSystem trait, which is inherited by any derived class, and uses the XmQTmenuSystem and XmQTmenuSavvy traits.

New Resources

RowColumn defines the following resources:

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
XmNentryClass	XmCEntryClass	WidgetClass	dynamic	CSG
XmNentryVerticalAlignment	XmCVerticalAlignment	unsigned char	XmALIGNMENT_CENTER	CSG
XmNisAligned	XmCIsAligned	Boolean	True	CSG
XmNisHomogeneous	XmCIsHomogenous	Boolean	dynamic	CSG
XmNlabelString	XmCXmString	XmString	NULL	C
XmNmarginHeight	XmCMarginHeight	Dimension	dynamic	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	dynamic	CSG
XmNmenuAccelerator	XmCAccelerators	String	dynamic	CSG
XmNmenuHelpWidget	XmCMenuWidget	Widget	NULL	CSG
XmNmenuHistory	XmCMenuWidget	Widget	NULL	CSG
XmNmenuPost	XmCMenuPost	String	NULL	CSG
XmNmnemonic	XmCMnemonic	KeySym	NULL	CSG
XmNmnemonicCharSet	XmCMnemonicCharSet	String	XmFONTLIST_DEFAULT_TAG	CSG
XmNnumColumns	XmCNumColumns	short	1	CSG

Name	Class	Type	Default	Access
XmNorientation	XmCOrientation	unsigned char	dynamic	CSG
XmNpacking	XmCPacking	unsigned char	dynamic	CSG
XmNpopupEnabled	XmCPopupEnabled	XtEnum	XmPOPUP_KEYBOARD	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
XmNtearOffModel	XmCTearOffModel	unsigned char	XmTEAR_OFF_DISABLED	CSG
XmNtearOffTitle	XmCTearOffTitle	XmString	NULL	CSG
XmNwhichButton	XmCWhichButton	unsigned int	dynamic	CSG

XmNadjustLast
: If True (default), the last row (or column) in the RowColumn widget is expanded so as to be flush with the edge.

XmNadjustMargin
: If True (default), text in each row (or column) will align with other text in its row (or column). This is done by forcing the margin resources (defined by the Label widget) to have the same value. For example, in a horizontally-oriented RowColumn widget, all items will have the same value for XmNmarginTop and XmNmarginBottom; in a vertically-oriented RowColumn widget, all items will have the same value for XmNmarginLeft and XmNmarginRight.

XmNentryAlignment
: When XmNisAligned is True, this resource tells RowColumn children how to align. The children must be subclasses of XmLabel or XmLabelGadget. If XmNrowColumnType is XmMENU_OPTION, the resource is forced to XmALIGNMENT_CENTER and cannot be changed. Possible values:

    XmALIGNMENT_BEGINNING
    XmALIGNMENT_CENTER
    XmALIGNMENT_END

XmNentryBorder
: The border width of a RowColumn widget's children.

XmNentryClass
The widget (or gadget) class to which children must belong when being added to a RowColumn widget. This resource is used only when the XmNisHomogeneous resource is set to True. XmNentryClass ensures that a MenuBar will have only cascade button children and that a RadioBox will have only toggle button children (or gadget variants of each class). XmNentryClass can have one of two default values. For a MenuBar, the default value is xmCascadeButtonWidgetClass. For a RadioBox, the default value is xmToggleButtonGadgetClass. Possible values:

```
xmToggleButtonGadgetClass     /* XmWORK_AREA with                */
                              /*      XmNradioBehavior True      */
xmCascadeButtonWidgetClass    /* XmMENU_BAR                      */
```

XmNentryVerticalAlignment
In Motif 1.2 and later, specifies how children that are subclasses of Label, Text, and TextField are aligned vertically. The resource has no effect if XmNorientation is XmVERTICAL or XmNpacking is XmPACK_TIGHT. Possible values:

XmALIGNMENT_BASELINE_BOTTOM
XmALIGNMENT_BASELINE_TOP
XmALIGNMENT_CONTENTS_BOTTOM
XmALIGNMENT_CENTER
XmALIGNMENT_CONTENTS_TOP

XmNisAligned
If True, enable the alignment specified in the XmNentryAlignment resource. Alignment is ignored in a label whose parent is a popup or pulldown MenuPane (for example, in an OptionMenu).

XmNisHomogeneous
If True, enforce the condition that all RowColumn children belong to the same class (the class specified by the XmNentryClass resource). When creating a RadioBox or a MenuBar, the default value of this resource is True; otherwise, it's False.

XmNlabelString
A label used only in option menus. A text string displays next to the selection area. By default, there is no label.

XmNmarginHeight

XmNmarginWidth
The spacing between an edge of the RowColumn widget and its nearest child. In popup and pulldown menus, the default is 0; in other types of RowColumn widgets, the default is 3 pixels.

XmNmenuAccelerator
: A pointer to a string that specifies an accelerator (keyboard shortcut) for use only in RowColumn widgets of type XmMENU_POPUP or XmMENU_BAR. In a popup menu, typing the accelerator posts the menu; in a menu bar, typing the accelerator highlights the first item and enables traversal in the menu bar. The string's format is like that of a translation but allows only a single key press event to be specified. The default value of this resource is KMenu (for popup menus) and KMenuBar (for menu bars).

XmNmenuHelpWidget
: The widget ID of the CascadeButton widget that serves as the Help button. This resource is meaningful only in RowColumn widgets of type XmMENU_BAR.

XmNmenuHistory
: The widget ID of the most recently activated menu entry. Since the most recently activated menu entry is also the choice that displays in an OptionMenu, this resource is useful for indicating the current selection in a RowColumn widget of type XmMENU_OPTION. In a RowColumn widget whose XmNradioBehavior resource is set to True, the XmNmenuHistory resource indicates the last toggle button to change from unselected to selected.

XmNmenuPost
: The string that describes the event for posting a menu. The value specifies an X event using translation table syntax. The default value depends on the type of RowColumn widget: for XmMENU_POPUP, the default is <Btn3Down>; for XmMENU_OPTION, XmMENU_BAR, and XmWORK_AREA, the default is <Btn1Down>; for XmMENU_PULLDOWN, this resource isn't meaningful.

XmNmnemonic
: The keysym of the key to press (in combination with the MAlt modifier) in order to post the pulldown menu associated with an option menu. This resource is meaningful only in option menus. In the label string, the first character matching this keysym will be underlined.

XmNmnemonicCharSet
: The character set for the option menu's mnemonic. The default value depends on the current language environment.

XmNnumColumns
: The number of columns (in a vertically-oriented RowColumn widget) or the number of rows (in a horizontally-oriented RowColumn widget). This resource is meaningful only when the XmNpacking resource is set to XmPACK_COLUMN.

XmNorientation
The direction for laying out the rows and columns of children of a RowColumn widget. For all RowColumn widgets except a MenuBar, the default value is XmVERTICAL. Possible values:

```
XmVERTICAL      /* top-to-bottom creation      */
XmHORIZONTAL    /* left-to-right creation      */
```

XmNpacking
The method of spacing the items placed within a RowColumn widget. The default value is XmPACK_COLUMN for a RadioBox, and XmPACK_TIGHT for other types of RowColumn widget. Possible values:

```
XmPACK_TIGHT    /* give each box minimum sizing    */
XmPACK_COLUMN   /* pad boxes to align if needed    */
XmPACK_NONE     /* widget accommodates placement   */
```

XmNpopupEnabled
If True (default), keyboard shortcuts are in effect for popup menus. Set this resource to False if you want to disable accelerators and mnemonics in popup menus.

In Motif 2.0 and later, the resource changes type from Boolean to XtEnum in order to support the range of values required for the enhanced automatic popup mechanisms. The value XmPOPUP_DISABLED is equivalent to the behaviour described above for False, and XmPOPUP_KEYBOARD is equivalent to True. In addition, XmPOPUP_AUTOMATIC adds event handlers for automatic menu popup, and enables the keyboard for the menu. XmPOPUP_AUTOMATIC_RECURSIVE is similar, except that the search for a popup menu in a given context is not restricted to immediate children, and the most specific popup to display may be found in the parent of a target widget.

The new enumeration erroneously has the generic representation type XmREnum rather than a putative XmRPopupEnabled, and an enumeration for the values has not been installed within the standard representation types. Hence as of Motif 2.0, the type cannot be properly specified in a resource file. The problem persists in Motif 2.1.30.

Only True (XmPOPUP_KEYBOARD) and False (XmPOPUP_DISABLED) work in the resource file.

XmNradioAlwaysOne
: This resource is effective only when the XmNradioBehavior resource is True. XmNradioAlwaysOne, when set to True (default), ensures that one of the toggle buttons is always selected. Once this button is selected, clicking on it will not deselect it; it can be deselected only by selecting another toggle button. If XmNradioAlwaysOne is False, a selected toggle button can be deselected by clicking on it or by selecting another button.

XmNradioBehavior
: If True, the RowColumn widget acts like a RadioBox by setting two of the resources for its toggle button children. Namely, the XmNindicatorType resource defaults to XmONE_OF_MANY, and the XmNvisibleWhenOff resource defaults to True. The default value of the XmNradioBehavior resource is False, unless the RowColumn widget was created with the `XmCreateRadioBox`() routine.

XmNresizeHeight

XmNresizeWidth
: If True (default), the widget requests a new height or width when necessary. If False, no resize requests are made.

XmNrowColumnType
: The type of RowColumn widget to create. You can't change this resource after it's set. Convenience routines create a RowColumn widget of the appropriate type. Possible values:

XmWORK_AREA	XmMENU_PULLDOWN
XmMENU_BAR	XmMENU_OPTION
XmMENU_POPUP	

XmNspacing
: The horizontal and vertical spacing between children in the RowColumn widget. For RowColumn widgets of type XmOPTION_MENU or XmWORK_AREA, the default value is 3 pixels; for other RowColumn types, the default is 0.

XmNsubMenuId
: The widget ID for the pulldown menu pane to be associated with an OptionMenu. This resource is meaningful only in RowColumn widgets of type XmMENU_OPTION.

XmNtearOffModel
: In Motif 1.2 and later, specifies whether tear-off behavior is enabled for a RowColumn with XmN-row-ColumnType set to XmMENU_PULLDOWN or XmMENU_POPUP. In Motif 1.2, this resource cannot be set from a resource file unless a converter is installed by calling the function `XmRepTypeInstallTearOffModelConverter`(). Possible values:

XmTEAR_OFF_DISABLED	XmTEAR_OFF_ENABLED

In Motif 2.0 and later, the converter is automatically installed, and `XmRepTypeInstallTearOffModelConverter`() is obsolete.

XmNtearOffTitle
In Motif 2.0 and later, specifies the title of the TearOff shell.

XmNwhichButton
This resource has been superseded by the XmNmenuPost resource but is retained for compatibility with older releases of Motif.

New Constraint Resources

RowColumn defines the following constraint resources for its children:

Name	Class	Type	Default	Access
XmNpositionIndex	XmCPositionIndex	short	XmLAST_POSITION	CSG

XmNpositionIndex
In Motif 1.2 and later, specifies the position of the widget in the RowColumn's list of children. A value of 0 indicates the beginning of the list, while XmLAST_POSITION places the child at the end of the list.

Simple Menu Creation Resources

The following resources are used with the simple menu creation routines. All resources have access C (create-only):

Name	Class	Type	Default
XmNbuttonAccelerators	XmCButtonAccelerators	StringTable	NULL
XmNbuttonAcceleratorText	XmCButtonAcceleratorText	XmStringTable	NULL
XmNbuttonCount	XmCButtonCount	int	0
XmNbuttonMnemonicCharSets	XmCButtonMnemonicCharSets	XmStringCharSetTable	NULL
XmNbuttonMnemonics	XmCButtonMnemonics	XmKeySymTable	NULL
XmNbuttons	XmCButtons	XmStringTable	NULL
XmNbuttonSet	XmCButtonSet	int	1
XmNbuttonType	XmCButtonType	XmButtonTypeTable	NULL
XmNoptionLabel	XmCOptionLabel	XmString	NULL
XmNoptionMnemonic	XmCOptionMnemonic	KeySym	NULL
XmNpostFromButton	XmCPostFromButton	int	-1
XmNsimpleCallback	XmCCallback	XtCallbackProc	NULL

XmNbuttonAccelerators
A list of accelerators, containing one item for each created title, separator, and button.

XmNbuttonAcceleratorText
A list of compound strings that represent the accelerators for the created buttons. The list contains one item for each created title, separator, and button.

XmNbuttonCount
The number of titles, separators, and menu buttons to create.

XmNbuttonMnemonicCharSets
A list of character sets to use for displaying button mnemonics. The list contains an item for each created title, separator, and button.

XmNbuttonMnemonics
A list of mnemonics associated with the buttons created. The list contains one item for each created title, separator, and button.

XmNbuttons
A list of compound strings that will serve as labels for the created buttons. The list contains one item for each created title, separator, and button.

XmNbuttonSet
The numeric position of the button to be initially set within a RadioBox or within an OptionMenu's pulldown submenu. The first button is specified as 0.

XmNbuttonType
A list of button types for the created buttons. The list contains one item for each created title, separator, and button. If this resource is not set, the buttons created will be CascadeButtonGadgets in a MenuBar and PushButtonGadgets in other types of RowColumn widget. The XmNbuttonType resource is an enumerated type whose possible values are:

XmPUSHBUTTON	XmCASCADEBUTTON
XmDOUBLE_SEPARATOR	XmCHECKBUTTON
XmRADIOBUTTON	XmSEPARATOR
XmTITLE	

XmNoptionLabel
A compound string with which to label the left side of an option menu.

XmNoptionMnemonic
The keysym of the key to press (in combination with the MAlt modifier) in order to post the pulldown menu associated with an option menu.

XmNpostFromButton
The numeric position of the cascade button (in the parent) from which the pull-down submenu is attached and subsequently posted. The first button is specified as 0.

XmNsimpleCallback
List of callbacks that are called when a button is pressed or when its value changes. For PushButtons and CascadeButtons, the callbacks are added to the XmNactivateCallback and for ToggleButtons they are added to the XmNvalueChangedCallback.

Callback Resources

RowColumn defines the following callback resources:

Callback	Reason Constant
XmNentryCallback	XmCR_ACTIVATE
XmNmapCallback	XmCR_MAP
XmNtearOffMenuActivateCallback	XmCR_TEAR_OFF_ACTIVATE
XmNtearOffMenuDeactivateCallback	XmCR_TEAR_OFF_DEACTIVATE
XmNunmapCallback	XmCR_UNMAP

XmNentryCallback
List of callbacks that are called when any button is pressed or when its value changes. When this resource is specified, the XmNactivateCallback and XmNvalueChangedCallback callbacks for all PushButtons, ToggleButtons, DrawnButtons, and CascadeButtons are disabled and instead call this callback. This resource must be specified when the RowColumn is created.

XmNmapCallback
List of callbacks that are called when the window associated with a RowColumn is going to be mapped.

XmNtearOffMenuActivateCallback
List of callbacks that are called when a tear-off menu pane is going to be torn off.

XmNtearOffMenuDeactivateCallback
List of callbacks that are called when a torn-off menu pane is going to be deactivated.

XmNunmapCallback
List of callbacks that are called when the window associated with a RowColumn is going to be unmapped.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;          /* the reason that the callback was called */
    XEvent      *event;          /* event structure that triggered callback */
    Widget      widget;          /* ID of activated RowColumn item          */
    char        *data;           /* value of application's client data      */
    char        *callbackstruct; /* created when item is activated          */
} XmRowColumnCallbackStruct;
```

The structure members *widget*, *data*, and *callbackstruct* are meaningful only when the callback *reason* is XmCR_ACTIVATE; otherwise, these structure members are set to NULL.

callbackstruct points to a structure that is created by the activation callback of the RowColumn item.

Inherited Resources

RowColumn inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. In Motif 2.0 and earlier, RowColumn sets the default value of XmNshadowThickness to 2 if XmN-rowColumnType is XmMENU_BAR, XmMENU_POPUP, or XmMENU_PULLDOWN; the resource is undefined when XmNrowColumnType is XmMENU_OPTION or XmWORK_AREA. In Motif 2.1 and later, the default value depends upon the XmDisplay XmNenableThinThickness resource: if True the default is 1, otherwise 2. The default value of XmNnavigationType is set to XmTAB_GROUP for a work area and XmNONE for an option menu; the resource is undefined for the other row column types. The default value of XmNtraversalOn is set to True for a work area or an option menu; the resource is undefined for the other row column types. The default value of XmNborderWidth is reset to 0 by XmManager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core

Resource	Inherited From	Resource	Inherited From
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Translations

The value of the XmN-rowColumnType resource determines the available translations. When XmNrowColumnType is XmWORK_AREA, RowColumn's translations are inherited from XmManager. When XmNrowColumnType is XmMENU_OPTION, RowColumn's translations are the traversal, KActivate, and KCancel translations inherited from XmManager, as well as the following:

Event	Action
BSelect Press	MenuBtnDown()
BSelect Release	MenuBtnUp()
KSelect	ManagerGadgetSelect()
KHelp	Help()

When XmNrowColumnType is XmMENU_BAR, XmMENU_PULLDOWN, or XmMENU_POPUP, RowColumn has the following translations (in PopupMenu systems, BMenu performs the BSelect actions as well):

Event	Action
BSelect Press	MenuBtnDown()
BSelect Release	MenuBtnUp()
KActivate	ManagerGadgetSelect()
KSelect	ManagerGadgetSelect()
Many KCancel	MenuGadgetEscape()

Event	Action
KHelp	Help()
KLeft	MenuGadgetTraverseLeft()
KRight	MenuGadgetTraverseRight()
KUp	MenuGadgetTraverseUp()
KDown	MenuGadgetTraverseDown()

Action Routines

RowColumn defines the following action routines:

Help()

Invokes any callbacks specified by the XmNhelpCallback resource.

ManagerGadgetSelect()

Arms and activates the gadget child (in a menu) that has focus. For a CascadeButtonGadget, its submenu is posted; for other gadget children, the menu hierarchy is unposted.

MenuBtnDown()

In a gadget child (in a menu), unposts any menus that were posted by the gadget's parent menu, turns mouse traversal on, and arms the gadget. If the child is a CascadeButtonGadget, its submenu is posted.

MenuBtnUp()

In a gadget child (in a menu), unposts the menu hierarchy and activates the gadget. If the child is a CascadeButtonGadget, this action posts the submenu and turns on keyboard traversal in the submenu.

MenuGadgetEscape()

Unposts the current menu and (unless the menu is a pulldown submenu) restores keyboard focus to the tab group or widget that previously had it (assuming an explicit focus policy). In a top-level pulldown menu pane attached to a menu bar, this action routine also disarms the cascade button and the menu bar.

MenuGadgetTraverseDown()

When the current menu item has a submenu and is in a MenuBar, disarms the current menu item, posts the submenu, and arms the first item in it. When the current menu item is in a menu pane, disarms the current menu item and arms the item below it, wrapping around to the top if necessary.

MenuGadgetTraverseLeft()

If the current menu item is in a MenuBar, disarms the current item and arms the MenuBar item to the left, wrapping around to the right if necessary. When the current item is in a menu pane, if the item is not at the left edge of the pane, disarms the current menu item and arms the item to its left. If the item is at the left edge of a submenu attached to the MenuBar, unposts the submenu, traverses to the MenuBar item to the left, and posts its submenu, wrapping if necessary.

MenuGadgetTraverseRight()

If the current menu item is in a MenuBar, disarms the current item and arms the MenuBar item to the right, wrapping around to the left if necessary. When the current item is in a menu pane, if the item is a CascadeButton, posts the associated submenu. If the current item is not at the right edge of the pane, disarms the current item and arms the item to the right, wrapping if necessary. Otherwise, unposts all submenus, traverses to the MenuBar item to the right, and posts its submenu, wrapping if necessary.

MenuGadgetTraverseUp()

Disarms the current menu item and arms the item above it, wrapping around to the bottom if necessary.

Additional Behavior

RowColumn has additional menu behavior:

KMenuBar

In a menu bar or in any menu pane cascaded from it, unposts the menu tree and (under an explicit focus policy) returns keyboard focus to the tab group that had it before entering the menu tree. In other non-popup menu panes, turns on keyboard traversal and sets the focus to the first menu bar item.

KMenu

Pops up the menu associated with the component with the keyboard focus and turns on keyboard traversal. In a popup menu system, unposts the menu tree and (under an explicit focus policy) returns keyboard focus to the tab group that had it before entering the menu tree.

See Also

XmCreateObject(1), XmGetMenuCursor(1), XmGetPostedFromWidget(1), XmGetTearOffControl(1), XmMenuPosition(1), XmOptionButtonGadget(1), XmOptionLabelGadget(1), XmRepTypeInstallTearOffModelConverter(1), XmSetMenuCursor(1), XmVaCreateSimpleCheckBox(1), XmVaCreateSimpleMenuBar(1), XmVaCreateSimpleOptionMenu(1), XmVaCreateSimplePopupMenu(1), XmVaCreateSimplePulldownMenu(1), XmVaCreateSimpleRadioBox(1), Composite(2), Constraint(2), Core(2), XmCascadeButton(2), XmCheckBox(2), XmManager(2), XmMenuBar(2), XmOptionMenu(2), XmPopupMenu(2), XmPulldownMenu(2), XmRadioBox(2).

Name

XmScale widget class – a manager widget that allows selection from a range of values.

Synopsis

Public Header:
<Xm/Scale.h>

Class Name:
XmScale

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmScale

Class Pointer:
xmScaleWidgetClass

Instantiation:
widget = `XmCreateScale` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmScaleWidgetClass,...)

Functions/Macros:
`XmCreateScale`(), `XmIsScale`(), `XmScaleGetValue`(), `XmScaleSetValue`(),
`XmScaleSetTicks`()

Description

A Scale displays a value from a range of values and allows a user to adjust the value. A Scale consists of a narrow, rectangular trough that contains a slider. The slider's position marks the current value within the range of values. Scale is a manager widget that orients its children along its axis. These children, typically labels, can be used as tick marks.

If the Scale is an input-output type, a user can change the value by moving the slider. An output-only Scale displays a value but does not allow the user to modify it. In Motif 2.0 and later, the XmNeditable resource controls the input-output type of the Scale; in Motif 1.2 and earlier, the programmer calls XtSetSensitive() or changes the inherited XmNsensitive resource to set the editable state.

In Motif 2.0 and later, the Scale supports tick marks directly through the XmScaleSetTicks() routine.

Traits

Scale holds the XmQTtransfer trait, which is inherited by any derived classes, and uses the XmQTspecifyRenderTable trait.

New Resources

Scale defines the following resources:

Name	Class	Type	Default	Access
XmNdecimalPoints	XmCDecimalPoints	short	0	CSG
XmNeditable	XmCEditable	Boolean	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNhighlightOnEnter	XmCHighlightOnEnter	Boolean	False	CSG
XmNhighlightThickness	XmCHighlightThickness	Dimension	dynamic	CSG
XmNmaximum	XmCMaximum	int	100	CSG
XmNminimum	XmCMinimum	int	0	CSG
XmNorientation	XmCOrientation	unsigned char	XmVERTICAL	CSG
XmNprocessingDirection	XmCProcessingDirection	unsigned char	dynamic	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNscaleHeight	XmCScaleHeight	Dimension	0	CSG
XmNscaleMultiple	XmCScaleMultiple	int	dynamic	CSG
XmNscaleWidth	XmCScaleWidth	Dimension	0	CSG
XmNshowArrows	XmCShowArrows	XtEnum	XmNONE	CSG
XmNshowValue	XmCShowValue	XtEnum	XmNONE	CSG
XmNsliderMark	XmCSliderMark	XtEnum	dynamic	CSG
XmNsliderSize	XmCSliderSize	int	dynamic	CSG
XmNsliderVisual	XmCSliderVisual	XtEnum	dynamic	CSG
XmNslidingMode	XmCSlidingMode	XtEnum	XmSLIDER	CSG
XmNtitleString	XmCTitleString	XmString	NULL	CSG
XmNvalue	XmCValue	int	dynamic	CSG

XmNdecimalPoints

A positive integer that determines how the slider's value will be displayed. The decimal point in the slider's value gets shifted to the right, and this resource specifies the number of decimal places to shift. For example, if the slider's value is 5678, then setting the XmNdecimalPoints[1] resource to 2 causes the widget to display the value as 56.78.

1.Erroneously given as XmdecimalPoints in 1st and 2nd editions.

XmNeditable
: In Motif 2.0 and later, specifies whether the Scale responds to user input. The default depends upon the value of the XmNslidingMode resource. If the value is XmSLIDER, the default is True, and for the value XmTHERMOMETER the default is False.

XmNfontList
: The font list used by the widget for the title. In Motif 2.0 and later, the XmFontList is obsolete, and is replaced by the XmRenderTable. Maintained for backwards compatibility, any specified render table takes precedence over the font list.

XmNhighlightOnEnter
: Determines whether to draw the widget's highlighting rectangle whenever the cursor moves into the widget. This resource applies only when the shell has a focus policy of XmPOINTER. If the XmNhighlightOnEnter resource is True, highlighting is drawn; if False (default), highlighting is not drawn.

XmNhighlightThickness
: The thickness of the highlighting rectangle. In Motif 2.0 and earlier, the default is 2. In Motif 2.1 and later, the default depends upon the XmDisplay XmNenableThinThickness resource: if True, the default is 1, otherwise 2.

XmNmaximum

XmNminimum
: The maximum/minimum value of the slider.

XmNorientation
: The direction in which the scale is displayed. Possible values:

```
XmVERTICAL                  /* top-to-bottom creation      */
XmHORIZONTAL                /* left-to-right creation      */
```

XmNprocessingDirection
: Determines the position at which to display the slider's maximum and minimum values, with respect to the slider. Possible values:

```
XmMAX_ON_TOP                /* scale increases toward top      */
XmMAX_ON_BOTTOM             /* scale increases toward bottom */
XmMAX_ON_LEFT               /* scale increases toward left     */
XmMAX_ON_RIGHT              /* scale increases toward right    */
```

  For vertically-oriented Scale widgets, the default value is XmMAX_ON_TOP. For horizontally-oriented Scale widgets, the default value is usually XmMAX_ON_RIGHT (depending on the value of the XmNstringDirection resource).

In Motif 2.0 and later, the XmNstringDirection resource is obsolete, and the default depends upon the XmNlayoutDirection value.

XmNrenderTable
In Motif 2.0 and later, specifies the XmRenderTable to use for both the title text string and the label displaying the current Scale value. If NULL, the value is found from the nearest ancestor holding the XmQTspecifyRenderTable trait, using the XmLABEL_RENDER_TABLE value of the ancestor.

XmNscaleHeight

XmNscaleWidth
The height or width of the slider area.

XmNscaleMultiple
The distance to move the slider when the user moves it by a multiple increment. The default value is calculated as (XmNmaximum – XmNminimum) / 10.

XmNshowArrows
In Motif 2.0 and later, specifies whether and how arrows are displayed on each end of the Scale. Possible values:

```
XmEACH_SIDE     /* arrow at both ends          */
XmMAX_SIDE      /* arrows at maximum end */
XmMIN_SIDE      /* arrows at minimum end  */
XmNONE          /* arrows at neither end      */
```

XmNshowValue
In Motif 1.2 and earlier, a Boolean value which specifies whether the current scale value is displayed on an adjacent label. If True, the Scale displays the value beside the slider. If False, the value label isn't displayed. In Motif 2.0 and later, the type of the resource changes to an enumeration. XmNONE is equivalent to False, and does not display a value. XmNEAR_BORDER places the value adjacent to the border of the Scale, and XmNEAR_SLIDER places the value at the slider.

XmNsliderMark
In Motif 2.0 and later, specifies the appearance of the slider. The default depends upon the value of the XmNslidingMode resource. If the sliding mode is XmSLIDER, the default is XmETCHED_LINE. With a mode of XmTHERMOMETER, the default is XmNONE if the scale is editable, otherwise XmROUND_MARK. Possible values:

```
XmETCHED_LINE   /* drawn as an etched line                          */
XmNONE          /* drawn as a foreground rectangle                  */
XmROUND_MARK    /* drawn as a shadowed circle                       */
XmTHUMB_MARK    /* three etched lines in foregrounded rectangle */
```

XmNsliderSize
In Motif 2.0 and later, an undocumented resource which represents the size of the slider in pixels.

XmNsliderVisual
In Motif 2.0 and later, specifies the color of the slider visual. The default is XmTROUGH_COLOR when the sliding model is XmTHERMOMETER, otherwise XmSHADOWED_BACKGROUND. Possible values:

```
XmBACKGROUND_COLOR      /* visual in background color          */
XmFOREGROUND_COLOR      /* visual in foreground color          */
XmSHADOWED_BACKGROUND   /* visual in background, with shadow */
XmTROUGH_COLOR          /* visual in trough color              */
```

XmNslidingMode
In Motif 2.0 and later, specifies the way in which the slider moves. Possible values:

```
XmSLIDER        /* slider moves freely between each end   */
XmTHERMOMETER   /* slider anchored to one end             */
```

XmNtitleString
The text string that appears as the title in the Scale widget.

XmNvalue
The current position of the slider along the scale. This resource must have a value between the values of XmNminimum and XmNmaximum.

Callback Resources

Scale defines the following callback resources:

Callback	Reason Constant
XmNconvertCallback	XmCR_OK
XmNdragCallback	XmCR_DRAG
XmNvalueChangedCallback	XmCR_VALUE_CHANGED

XmNconvertCallback
In Motif 2.0 and later, specifies a list of callbacks called when the slider is requested to convert a selection as part of a data transfer operation.

XmNdragCallback
List of callbacks that are called when the slider is being dragged.

XmNvalueChangedCallback
List of callbacks that are called when the position of the slider has changed.

Callback Structure

Convert callbacks are fully described within the sections covering the Uniform Transfer Model. See XmTransfer(1) for more details. For quick reference, a pointer to the following structure is passed to callbacks on the XmNconvertCallback list:

```
typedef struct {
int             reason;         /* the reason that the callback is invoked     */
XEvent          *event;         /* points to event that triggered callback     */
Atom            selection;      /* selection for which conversion is requested */
Atom            target;         /* the conversion target                       */
XtPointer       source_data;      /* selection source information              */
XtPointer       location_data;    /* information about data to be transferred  */
int             flags;          /* input status of the conversion              */
XtPointer       parm;           /* parameter data for the target               */
int             parm_format;    /* format of parameter data                    */
unsigned long   parm_length;    /* number of elements in parameter data      */
Atom            parm_type;    /* the type of the parameter data              */
int             status;         /* output status of the conversion             */
XtPointer       value;          /* returned conversion data                    */
Atom            type;           /* type of conversion data returned            */
int             format;         /* format of the conversion data               */
unsigned long   length;       /* number of elements in the conversion data   */
} XmConvertCallbackStruct;
```

Each drag and value changed callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* the reason that the callback was called */
    XEvent      *event;         /* event structure that triggered callback */
    int         value;          /* new value of the slider                 */
} XmScaleCallbackStruct;
```

Inherited Resources

Scale inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. In Motif 2.0 and earlier, the default value of XmNshadowThickness is reset to 2. In Motif 2.1, the default value depends upon the XmDisplay XmNenableThinThickness resource: if True the default is 1, otherwise 2. The default value of XmNborderWidth is reset to 0 by XmManager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Translations

Scale does not define any new translations.

Behavior

Scale has the following behavior:

BSelect Press or BTransfer Press
: In the trough between the slider and an end of the Scale, moves the slider by one multiple increment in the direction of the end of the Scale. Calls the XmNvalueChangedCallback callbacks. Whether the value of the Scale is incremented or decremented depends on the value of XmNprocessingDirection. In the slider, starts interactive dragging of the slider.

BSelect Motion or BTransfer Motion
: If the button press occurred within the slider, the slider tracks the pointer and calls the XmNdragCallback callbacks.

BSelect Release or BTransfer Release
: If the button press occurs within the slider and the position of the slider has changed, the XmNvalueChangedCallback callbacks are invoked.

MCtrl BSelect Press
: In the trough between the slider and an end of the Scale, moves the slider to that end of the Scale and calls the XmNvalueChangedCallback callbacks. Whether the value of the Scale is incremented or decremented depends on the value of the XmNprocessingDirection resource.

KUp
KDown
: In a vertical Scale, moves the slider up or down one increment and calls the XmN-value-Changed-Callback callbacks. Whether the value of the Scale is incremented or decremented depends on the value of the XmN-processing-Direction resource.

KLeft
KRight
: In a horizontal Scale, moves the slider left or right one increment and calls the XmNvalue-Changed-Callback callbacks. Whether the value of the Scale is incremented or decremented depends on the value of the XmN-processing-Direction resource.

MCtrl KUp or KPageUp
MCtrl KDown or KPageDown
: In a vertical Scale, moves the slider up or down one multiple increment and calls the XmNvalueChangedCallback callbacks.

Whether the value of the Scale is incremented or decremented depends on the value of the XmNprocessingDirection resource.

MCtrl KLeft or KPageLeft
MCtrl KRight or KPageRight
: In a horizontal Scale, moves the slider left or right one multiple increment and calls the XmNvalueChangedCallback callbacks. Whether the value of the Scale is incremented or decremented depends on the value of the XmNprocessingDirection resource.

KBeginLine or KBeginData
: Moves the slider to the Scale's minimum value and calls the XmNvalueChangedCallback callbacks.

KEndLine or KEndData
: Moves the slider to the Scale's maximum value and calls the XmNvalueChangedCallback callbacks.

KNextField
KPrevField
: Moves the keyboard focus to the first item in the next or previous tab group, wrapping if necessary.

KHelp
: Invokes the list of callbacks specified by XmNhelpCallback. If the Scale does not have any help callbacks, invokes those associated with the nearest ancestor that has them.

See Also

`XmCreateObject`(1), `XmScaleGetValue`(1), `XmScaleSetValue`(1), `XmScaleSetTicks`(1), `XmTransfer`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmManager`(2).

Name

XmScreen widget class – an object used to store screen-specific information.

Synopsis

Public Header:
<Xm/Screen.h>

Class Name:
XmScreen

Class Pointer:
xmScreenClass

Class Hierarchy:
Core →XmScreen

Instantiation:
widget = `XtAppInitialize`(...)

Functions/Macros:
`XmGetXmScreen`(), `XmIsScreen`()

Availability

Motif 1.2 and later.

Description

The Screen object stores screen-specific information for use by the toolkit. An application has a Screen object for each screen that it accesses. When an application creates its first shell on a screen, typically by calling `XtAppInitialize`() or `XtAppCreateShell`(), a Screen object is created automatically. There is no way to create a Screen independently. The function `XmGetXmScreen`() can be used to get the widget ID of the Screen object.

New Resources

Screen defines the following resources:

Name	Class	Type	Default	Access
XmNbitmapConversionModel	XmCBitmapConversionModel	XtEnum	XmMATCH_DEPTH	CSG
XmNcolorAllocationProc	XmCColorAllocationProc	XmAllocColorProc	NULL	CSG
XmNcolorCalculationProc	XmCColorCalculationProc	XmScreenColorProc	NULL	CSG
XmNdarkThreshold	XmCDarkThreshold	int	dynamic	C
XmNdefaultCopyCursorIcon	XmCDefaultCopyCursorIcon	Widget	NULL	CSG
XmNdefaultInvalidCursorIcon	XmCDefaultInvalidCursorIcon	Widget	NULL	CSG
XmNdefaultLinkCursorIcon	XmCDefaultLinkCursorIcon	Widget	NULL	CSG
XmNdefaultMoveCursorIcon	XmCDefaultMoveCursorIcon	Widget	NULL	CSG

Name	Class	Type	Default	Access
XmNdefaultNoneCursorIcon	XmCDefaultNoneCursorIcon	Widget	NULL	CSG
XmNdefaultSourceCursorIcon	XmCDefaultSourceCursorIcon	Widget	NULL	CSG
XmNdefaultValidCursorIcon	XmCDefaultValidCursorIcon	Widget	NULL	CSG
XmNfont	XmCFont	XFontStruct *	NULL	CSG
XmNforegroundThreshold	XmCForegroundThreshold	int	dynamic	C
XmNhorizontalFontUnit	XmCHorizontalFontUnit	int	dynamic	CSG
XmNinsensitiveStippleBitmap	XmCInsensitiveStippleBitmap	Pixmap	50_foreground	C
XmNlightThreshold	XmCLightThreshold	int	dynamic	C
XmNmenuCursor	XmCCursor	Cursor[a]	arrow	C
XmNmoveOpaque	XmCMoveOpaque	Boolean	False	CSG
XmNunpostBehavior	XmCUnpostBehavior	unsigned char	XmUNPOST_AND_REPLAY	CSG
XmNuseColorObject	XmCUseColorObject	Boolean	False	C
XmNuserData	XmCUserData	XtPointer	NULL	CSG
XmNverticalFontUnit	XmCVerticalFontUnit	int	dynamic	CSG

a.Erroneously given as String in 1st and 2nd editions. The default is the arrow Cursor (XC_arrow).

XmNbitmapConversionModel
: In Motif 2.0 and later, specifies the way in which Xpm and Xbm files are converted to a pixmap. If the value is XmMATCH_DEPTH, the pixmap has the same depth as the widget to which it is associated. If the value is XmMATCH_DYNAMIC, Xbm files are converted to a pixmap of depth 1.

XmNcolorAllocationProc
: In Motif 2.0 and later, specifies an XmAllocColorProc procedure used for allocating color on the particular screen associated with the XmScreen object. If this is NULL, the default procedure `XAllocColor()` is used.

XmNcolorCalculationProc
: In Motif 2.0 and later, specifies an XmScreenColorProc procedure for calculating the default foreground, background, top shadow, bottom shadow, select colors on the particular screen associated with the XmScreen object. If this is NULL, color is calculated using a default screen-independent procedure. The default procedure can be changed by `XmSetColorCalculation`().

XmNdarkThreshold
: The level of perceived brightness (between 0 and 100) that is treated as a "dark" background color when computing default shadow and select colors.

XmNdefaultCopyCursorIcon
: The DragIcon used during a copy operation. When the value is NULL, a default system icon is used.

XmNdefaultInvalidCursorIcon
: The DragIcon used when the pointer is over an invalid drop site. When the value is NULL, a default system icon is used.

XmNdefaultLinkCursorIcon
: The DragIcon used during a link operation. When the value is NULL, a default system icon is used.

XmNdefaultMoveCursorIcon
: The DragIcon used during a move operation. When the value is NULL, a default system icon is used.

XmNdefaultNoneCursorIcon
: The DragIcon used when the pointer is not over a drop site. When the value is NULL, a default system icon is used.

XmNdefaultSourceCursorIcon
: The bitmap used as a cursor when an XmNsourceCursorIcon is not provided by the DragContext. When the value is NULL, a default system icon is used.

XmNdefaultValidCursorIcon
: The DragIcon used when the pointer is over a valid drop site. When the value is NULL, a default system icon is used.

XmNfont
: The font used in computing values for XmNhorizontalFontUnit and XmN-verticalFontUnit.

XmNforegroundThreshold
: The level of perceived brightness (between 0 and 100) that distinguishes between a "dark" and "light" background when computing the default foreground and highlight colors.

XmNhorizontalFontUnit
: The horizontal component of the font units that are used to convert geometry values when XmNshellUnitType or XmNunitType is set to Xm100TH_FONT_UNITS. If a value is not specified, the default is computed from the XmNfont resource.

XmNinsensitiveStippleBitmap
: In Motif 2.0 and later, specifies a default stipple for drawing widgets in insensitive state. Mostly used within the graphics contexts of Gadgets.

XmNlightThreshold
: The level of perceived brightness (between 0 and 100) that is treated as a "light" background color when computing default shadow and select colors.

XmNmenuCursor
: The cursor that is used when the application posts a menu. Possible values include all of the cursors in the X cursor font.

XmNmoveOpaque
If False (default), an operation that moves a window displays an outline of the window during the operation. If True, a move operation displays a representation of the window.

XmNunpostBehavior
The behavior of a posted menu when the pointer button is pressed outside of the menu. Possible values:

```
XmUNPOST_AND_REPLAY     /* unposts the menu hierarchy and
replays event */
XmUNPOST                /* unposts the menu hierar-
chy            */
```

XmNuseColorObject
In Motif 2.0 and later, specifies whether colors are shareable between widgets, and whether an alteration to a color dynamically changes all widgets which reference the color.

XmNuserData
In Motif 2.0 and later, specifies a pointer to data that the application can attach to the structure representing the screen. The resource is unused internally.

XmNverticalFontUnit
The vertical component of the font units that are used to convert geometry values when XmNshellUnitType or XmNunitType is set to Xm100TH_FONT_UNITS or XmFONT_UNITS[1]. If a value is not specified, the default is computed from the XmNfont resource.

Procedures

The XmScreenColorProc has the following syntax:

typedef void (*XmScreenColorProc) (Screen *, XColor *, XColor *, XColor *, XColor *, XColor *)

```
Screen      *screen;        /* the screen                  */
XColor      *bg_color;      /* specifies the background    color */
XColor      *fg_color;      /* returns   the foreground    color */
XColor      *sel_color;     /* returns   the select        color */
XColor      *ts_color;      /* returns   the top    shadow color */
XColor      *bs_color;      /* returns   the bottom shadow color */
```

An XmScreenColorProc takes six arguments. The first argument is a pointer to the screen. The second argument, bg_color, is a pointer to an XColor structure that specifies the background color. The red, green, blue, and pixel fields in the

1.Erroneously given as Xm_FONT_UNITS in 2nd edition.

structure contain valid values. The rest of the arguments are pointers to XColor structures for the colors that are to be calculated. The procedure fills in the red, green, and blue fields in these structures.

The XmAllocColorProc has the following syntax:

typedef void (*XmAllocColorProc) (Display *, Colormap, XColor *)

```
Display     *display;       /* connection to X server              */
Colormap    colormap;       /* a colormap in which to allocate color */
XColor      *bs_color;      /* specifies and returns allocated color */
```

An XmAllocColorProc takes three arguments. The first argument is a pointer to the Display connection. The second argument is the Colormap where the color is to be allocated. The third argument is a pointer to an XColor structure for the color that is to be allocated. The programmer fills in the red, green, and blue fields in the structure to the required values, and the procedure returns the actually allocated values into the same fields.

Inherited Resources

None of the resources inherited by Screen are applicable.

See Also

`XmGetXmScreen`(1), `XmSetColorCalculation`(1), `Core`(2), `XmDisplay`(2).

Name

XmScrollBar widget class – a widget to control the scrolling of the viewing area in another widget.

Synopsis

Public Header:
<Xm/ScrollBar.h>

Class Name:
XmScrollBar

Class Hierarchy:
Core →XmPrimitive →XmScrollBar

Class Pointer:
xmScrollBarWidgetClass

Instantiation:
widget = `XmCreateScrollBar` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmScrollBarWidgetClass,...)

Functions/Macros:
`XmCreateScrollBar`(), `XmIsScrollBar`(), `XmScrollBarGetValues`(),
`XmScrollBarSetValues`()

Description

A ScrollBar allows users to reposition data that is too large to fit in the viewing window. Although a ScrollBar can be used as a standalone widget, it is normally used in a ScrolledWindow. A ScrollBar consists of a rectangular strip, called the scroll region or trough, and two arrows placed on either end of the scroll region. Within the scroll region is a smaller, movable rectangle called the slider. To scroll the data, users can click on one of the arrows, click in the scroll region, or drag the slider. The application typically sets the XmNsliderSize resource such that the size of the slider relative to the size of the scroll region corresponds to the percentage of total data that is currently displayed.

Traits

ScrollBar holds the XmQTnavigator trait, which is inherited by any derived classes.

New Resources

ScrollBar defines the following resources:

Name	Class	Type	Default	Access
XmNeditable	XmCEditable	Boolean	dynamic	CSG
XmNincrement	XmCIncrement	int	1	CSG
XmNinitialDelay	XmCInitialDelay	int	250	CSG
XmNmaximum	XmCMaximum	int	dynamic	CSG
XmNminimum	XmCMinimum	int	0	CSG
XmNorientation	XmCOrientation	unsigned char	XmVERTICAL	CSG
XmNpageIncrement	XmCPageIncrement	int	10	C
XmNprocessingDirection	XmCProcessingDirection	unsigned char	dynamic	CSG
XmNrepeatDelay	XmCRepeatDelay	int	50	CSG
XmNshowArrows	XmCShowArrows	XtEnum	XmEACH_SIDE	CSG
XmNsliderMark	XmCSliderMark	XtEnum	dynamic	CSG
XmNsliderSize	XmCSliderSize	int	dynamic	CSG
XmNsliderVisual	XmCSliderVisual	XtEnum	dynamic[a]	CSG
XmNslidingMode	XmCSlidingMode	XtEnum	XmSLIDER	CSG
XmNsnapBackMultiple	XmCSnapBackMultiple	unsigned short	65535	CSG
XmNtroughColor	XmCTroughColor	Pixel	dynamic	CSG
XmNvalue	XmCValue	int	dynamic	CSG

a.Erroneously given as XmSHADOWED_BACKGROUND in 2nd edition.

XmNeditable
: In Motif 2.0 and later, specifies whether the ScrollBar responds to user input. The default depends upon the value of the XmNslidingMode resource. If the value is XmSLIDER, the default is True, and for the value XmTHERMOMETER the default is False.

XmNincrement
: The amount the value changes due to the user's moving the slider one increment.

XmNinitialDelay
: The number of milliseconds a button must remain pressed before triggering continuous slider movement.

XmNmaximum

XmNminimum
: The maximum/minimum value of the slider.

XmNorientation
The direction in which the scale is displayed. Possible values:

```
XmVERTICAL          /* top-to-bottom creation */
XmHORIZONTAL        /* left-to-right creation */
```

XmNpageIncrement
The amount the value changes due to the user's moving the slider one page increment.

XmNprocessingDirection
Determines the position at which to display the slider's maximum and minimum values, with respect to the slider. Possible values:

```
XmMAX_ON_TOP        /* scale increases toward top    */
XmMAX_ON_BOTTOM     /* scale increases toward bottom */
XmMAX_ON_LEFT       /* scale increases toward left   */
XmMAX_ON_RIGHT      /* scale increases toward right */
```

For vertically oriented ScrollBar widgets, the default value is XmMAX_ON_TOP. For horizontally oriented ScrollBar widgets, the default value is usually XmMAX_ON_RIGHT (depending on the value of the XmNstringDirection resource).

XmNrepeatDelay
The number of milliseconds a button must remain pressed before continuing further slider motions, once the XmNinitialDelay time has been triggered.

XmNshowArrows
In Motif 1.2 and earlier, a Boolean value which indicates whether arrows are displayed. If True, arrows are displayed; if False, they are not.

In Motif 2.0 and later, the resource is represented by an enumerated type: if XmEACH_SIDE, arrows are displayed at each end of the ScrollBar, XmMAX_SIDE displays both[1] arrows at the end where the maximum value is displayed, XmMIN_SIDE displays both[2] arrows at the minimum value end, and XmNONE does not display any arrows.

XmNsliderMark
In Motif 2.0 and later, specifies the appearance of the slider. The default depends upon the value of the XmNslidingMode resource. If the sliding mode is XmSLIDER, the default is XmETCHED_LINE. With a mode of XmTHERMOMETER, the default is XmNONE if the scale is editable, otherwise XmROUND_MARK. Possible values:

1.Erroneously given as *one* arrow in 2nd edition.

2.Erroneously given as *one* arrow in 2nd edition

```
XmETCHED_LINE       /* drawn as an etched line                          */
XmNONE              /* drawn as a foreground rectangle                  */
XmROUND_MARK        /* drawn as a shadowed circle                       */
XmTHUMB_MARK        /* three etched lines in foregrounded rectangle */
```

XmNsliderSize
The slider's length. The length ranges from 1 to the value of XmNmaximum – XmNminimum. By default, the value is computed to be:

(XmNmaximum – XmNminimum) / 10.

XmNsliderVisual
In Motif 2.0 and later, specifies the color of the slider visual. The default is XmTROUGH_COLOR when the sliding model is XmTHERMOMETER, otherwise XmSHADOWED_BACKGROUND. Possible values:

```
XmBACKGROUND_COLOR      /* visual in background color          */
XmFOREGROUND_COLOR      /* visual in foreground color          */
XmSHADOWED_BACKGROUND   /* visual in background, with shadow   */
XmTROUGH_COLOR          /* visual in trough color              */
```

XmNslidingMode
In Motif 2.0 and later, specifies the way in which the slider moves. Possible values:

```
XmSLIDER        /* slider moves freely between each end    */
XmTHERMOMETER   /* slider anchored to one end              */
```

XmNsnapBackMultiple
In Motif 2.0 and later, specifies a distance, which if exceeded, causes the ScrollBar to snap back to its original settings. The resource comes into effect when the user drags the mouse outside the bounds of the ScrollBar. The resource is measured in terms of multiples of the ScrollBar width. For example, the value 0 (zero) causes the slider to snap back as soon as the pointer moves outside the ScrollBar, the value 1 snaps back at one ScrollBar width, etc. The default is very large, in order to disable snap back even if the size of the screen is abnormal.

XmNtroughColor
The color of the slider's trough.

XmNvalue
The slider's position. The position ranges from the value of XmNminimum to the value of (XmNmaximum – XmNsliderSize).

Callback Resources

ScrollBar defines the following callback resources:

Callback	Reason Constant
XmNdecrementCallback	XmCR_DECREMENT
XmNdragCallback	XmCR_DRAG
XmNincrementCallback	XmCR_INCREMENT
XmNpageDecrementCallback	XmCR_PAGE_DECREMENT
XmNpageIncrementCallback	XmCR_PAGE_INCREMENT
XmNtoBottomCallback	XmCR_TO_BOTTOM
XmNtoTopCallback	XmCR_TO_TOP
XmNvalueChangedCallback	XmCR_VALUE_CHANGED

XmNdecrementCallback
: List of callbacks that are called when the value of the ScrollBar decreases by one increment.

XmNdragCallback
: List of callbacks that are called for each change in position when the slider is being dragged.

XmNincrementCallback
: List of callbacks that are called when the value of the ScrollBar increases by one increment.

XmNpageDecrementCallback
: List of callbacks that are called when the value of the ScrollBar decreases by one page increment.

XmNpageIncrementCallback
: List of callbacks that are called when the value of the ScrollBar increases by one page increment.

XmNtoBottomCallback
: List of callbacks that are called when the slider is moved to the maximum value of the ScrollBar.

XmNtoTopCallback
: List of callbacks that are called when the slider is moved to the minimum value of the ScrollBar.

XmNvalueChangedCallback
: List of callbacks that are called at the end of a slider drag operation. These callbacks are also called in place of each of the other ScrollBar callbacks that reports a value change when the callback resource is NULL.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;     /* the reason that the callback was called */
    XEvent      *event;     /* event structure that triggered callback */
    int         value;      /* value of the slider's new location      */
    int         pixel;      /* coordinate where selection occurred     */
} XmScrollBarCallbackStruct;
```

pixel is meaningful only when the callback *reason* is XmCR_TO_TOP or XmCR_TO_BOTTOM. The *pixel* member specifies the location at which the mouse button selection occurred, giving the x-coordinate in the case of a horizontal ScrollBar and the y-coordinate in the case of a vertical ScrollBar.

Inherited Resources

ScrollBar inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. XmNnavigationType to XmSTICKY_TAB_GROUP, and XmN-traversalOn to False. The default value of XmNborderWidth is reset to 0 by Manager.

In versions of Motif prior to 2.0, the default value of XmNhighlightThickness is reset to zero by the ScrollBar. In Motif 2.0, the default is reset to 2 if the ScrollBar's parent is a ScrolledWindow, zero otherwise. In Motif 2.1, if the XmNenableThinThickness resource of XmDisplay is True, the default is 1 if the ScrollBar's parent is a ScrolledWindow, zero otherwise.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNhighlightThickness	XmPrimitive
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNlayoutDirection	XmPrimitive
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnavigationType	XmPrimitive
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmPrimitive	XmNsensitive	Core
XmNbottomShadowPixmap	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNcolormap	Core	XmNtopShadowColor	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNdepth	Core	XmNtranslations	Core
XmNdestroyCallback	Core	XmNtraversalOn	XmPrimitive

Resource	Inherited From	Resource	Inherited From
XmNforeground	XmPrimitive	XmNunitType	XmPrimitive
XmNheight	Core	XmNuserData	XmPrimitive
XmNhelpCallback	XmPrimitive	XmNwidth	Core
XmNhighlightColor	XmPrimitive	XmNx	Core
XmNhighlightOnEnter	XmPrimitive	XmNy	Core
XmNhighlightPixmap	XmPrimitive		

Translations

The translations for ScrollBar include those from Primitive, plus the following:

Event	Action
BSelect Press	Select()
BSelect Release	Release()
BSelect Press Moved	Moved()
BTransfer Press	Select()
BTransfer Release	Release()
BTransfer Press Moved	Moved()
MCtrl BSelect Press	TopOrBottom()
MCtrl BSelect Release	Release()
KUp	IncrementUpOrLeft(0)
MCtrl KUp	PageUpOrLeft(0)
KDown	IncrementDownOrRight(0)
MCtrl KDown	PageDownOrRight(0)
KLeft	IncrementUpOrLeft(1)
MCtrl KLeft	PageUpOrLeft(1)
KRight	IncrementDownOrRight(1)
MCtrl KRight	PageDownOrRight(1)
KPageUp	PageUpOrLeft(0)
KPageDown	PageDownOrRight(0)
KPageLeft	PageUpOrLeft(1)
KPageRight	PageDownOrRight(1)
KBeingLine	TopOrBottom()
KEndLine	TopOrBottom()
KBeginData	TopOrBottom()
KEndData	TopOrBottom()

Event	Action
KNextField	PrimitiveNextTabGroup()
KPrevField	PrimitivePrevTabGroup()
KActivate	PrimitiveParentActivate()
KCancel	CancelDrag()
KHelp	PrimitiveHelp()

Action Routines

ScrollBar defines the following action routines:

CancelDrag()

In Motif 1.2 and later, cancels the scrolling operation and returns the slider to its previous location if the event happened during a drag. Otherwise, passes the event to the parent if it is a manager.

IncrementDownOrRight(flag):

Moves the slider by one increment--downward if flag is 0; to the right if flag is 1. Depending on the value of the XmNprocessingDirection resource, the slider's movement invokes the list of callbacks specified by either the XmNincrementCallback or the XmNdecrementCallback resource (or XmNvalueChangedCallback if the appropriate callback resource is NULL).

IncrementUpOrLeft(flag):

Same as IncrementDownOrRight except that the slider moves upward if flag is 0 and to the left if flag is 1.

Moved()

This action applies when the mouse button is pressed in the slider. When this is done, moving the pointer moves the slider along with it and also invokes the callbacks specified by XmNdragCallback.

PageDownOrRight(flag):

Moves the slider by one page increment--downward if flag is 0; to the right if flag is 1. Depending on the value of the XmNprocessingDirection resource, the slider's movement invokes the callbacks listed in either XmNpageIncrementCallback or XmNpageDecrementCallback (or XmNvalueChangedCallback if the appropriate callback resource is NULL).

PageUpOrLeft(flag):

Same as IncrementDownOrRight except that the slider moves upward if flag is 0 and to the left if flag is 1.

PrimitiveHelp()

Invokes the list of callbacks specified by XmNhelpCallback. If the ScrollBar doesn't have any help callbacks, the Help() routine invokes those associated with the nearest ancestor that has them.

PrimitiveNextTabGroup()
PrimitivePrevTabGroup()

Traverses to the first item in the next/previous tab group, wrapping if necessary.

PrimitiveParentActivate()

In Motif 1.2 and later, passes the event to the parent if it is a manager.

Release()

If the Moved() action changes the slider's position, then the Release() action invokes the callbacks specified by XmNvalueChangedCallback.

Select()

The results of this action depend on the location in which its applied: Within an arrow, this action is the same as IncrementDownOrRight() or IncrementUpOrLeft()--incrementing or decrementing according to the value of the XmN-processingDirection resource, and invoking the appropriate increment or decrement callback. Within the scrolling area that lies between an arrow and the slider, this action works like the page increment action routines--moving by one page increment according to the value of the XmNprocessingDirection resource, and invoking the appropriate page increment or page decrement callback. Within either of these locations, keeping the button pressed repeats the incremental movement of the slider. This behavior is triggered when the duration of the button press exceeds the value of the XmNinitialDelay resource; the slider movement then repeats with a time interval specified by the XmNrepeatDelay resource. Within the slider, this action begins slider dragging, which is subsequently affected by the actions Moved() and Release().

TopOrBottom()

Moves the slider to its minimum value and invokes the callbacks specified by XmNtoTopCallback, or moves the slider to its maximum value and invokes the callbacks specified by XmNtoBottomCallback. The direction of the slider's movement depends on the value of the XmNprocessingDirection resource. This action can be applied using either keyboard or mouse events.

See Also

`XmCreateObject`(1), `XmScrollBarGetValues`(1), `Core`(2), `XmPrimitive`(2).

Name

XmScrolledList – a List as a child of a ScrolledWindow.

Synopsis

Public Header:
<Xm/List.h>

Instantiation:
widget = `XmCreateScrolledList` (parent, name,...)

Functions/Macros:
`XmCreateScrolledList`(), `XmCreateScrolledWindow`()

Description

An XmScrolledList is a compound object created by a call to XmCreateScrolledList() that provides scroll bars for a list that is not visible all at once. A ScrolledList consists of a ScrolledWindow widget with a List widget as its child.

A ScrolledList automatically creates the necessary scroll bars. The ScrolledWindow resource XmNscrollingPolicy is set to XmAPPLICATION_DEFINED and XmNvisualPolicy is set to XmVARIABLE. The ScrolledWindow resource XmNscrollBarDisplayPolicy is set to XmSTATIC, but no initial value is set for the List XmNscrollBarDisplayPolicy resource.

Default Resource Values

A ScrolledList sets the following default values for ScrolledWindow resources:

Name	Default
XmNscrollBarDisplayPolicy	XmSTATIC
XmNscrollingPolicy	XmAPPLICATION_DEFINED
XmNvisualPolicy	XmVARIABLE

Widget Hierarchy

When a ScrolledList is created with a specified name, the ScrolledWindow is named name*SW* and the List is called *name*. The horizontal and vertical scroll bars are named HorScrollBar and VertScrollBar, respectively.

See Also

`XmCreateObject`(1), `XmList`(2), `XmScrolledWindow`(2).

Name

XmScrolledText – a Text widget as a child of a ScrolledWindow.

Synopsis

Public Header:
<Xm/Text.h>

Instantiation:
widget = `XmCreateScrolledText` (parent, name,...)

Functions/Macros:
`XmCreateScrolledText`(), `XmCreateScrolledWindow`()

Description

An XmScrolledText is a compound object created by a call to XmCreate-ScrolledText() that provides scroll bars for text that is not visible all at once. A ScrolledText object consists of a ScrolledWindow widget with a multi-line Text widget as its child.

ScrolledText automatically creates the necessary scroll bars. The ScrolledWindow resource XmNscrollingPolicy is set to XmAPPLICATION_DEFINED, XmNvisualPolicy is set to XmVARIABLE and XmNscrollBarDisplayPolicy is set to XmSTATIC.

Default Resource Values

ScrolledText sets the following default values for ScrolledWindow resources:

Name	Default
XmNscrollBarDisplayPolicy	XmSTATIC
XmNscrollingPolicy	XmAPPLICATION_DEFINED
XmNvisualPolicy	XmVARIABLE

Widget Hierarchy

When a ScrolledText object is created with a specified name, the ScrolledWindow is named name*SW* and the Text widget is called *name*. The horizontal and vertical scroll bars are named HorScrollBar and VertScrollBar respectively.

See Also

`XmCreateObject`(1), `XmScrolledWindow`(2), `XmText`(2).

Name

XmScrolledWindow widget class – a manager widget that provides scroll bars for the data display.

Synopsis

Public Header:
<Xm/ScrolledW.h>

Class Name:
XmScrolledWindow

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmScrolledWindow

Class Pointer:
xmScrolledWindowWidgetClass

Instantiation:
widget = XmCreateScrolledWindow (parent, name,...)
or
widget = XtCreateWidget (name, xmScrolledWindowWidgetClass,...)

Functions/Macros:
`XmCreateScrolledList`(), `XmCreateScrolledText`(), `XmCreateScrolledWindow`(), `XmIsScrolledWindow`(), `XmScrollVisible`(), `XmScrolledWindowSetAreas`()

Description

ScrolledWindow provides a scrollable view of data that may not be visible all at once. ScrollBars allow a user to scroll the visible part of the window through the larger display. A ScrolledWindow widget can be created so that it scrolls automatically without application intervention or so that an application provides support for all scrolling operations. When scrolling is handled automatically, ScrolledWindow creates the scroll bars, which are named HorScrollBar and VertScrollBar.

Each of the ScrolledWindow regions is associated with a ScrolledWindow resource; `XmScrolledWindowSetAreas`() sets the associated resources. If an application does not call `XmScrolledWindowSetAreas`(), the widget may still set some of the standard regions. If ScrollBars are added as children, the XmNhorizontalScrollBar and XmNverticalScrollBar resources may be set if they have not already been specified. Any child that is not a ScrollBar is used for the XmNworkWindow. If you want to be certain about which widgets are used for the different regions, it is wise to call `XmScrolledWindowSetAreas`() explicitly.

Traits

ScrolledWindow holds the XmQTscrollFrame trait, which is inherited by any derived classes, and uses the XmQTnavigator trait.

New Resources

ScrolledWindow defines the following resources:

Name	Class	Type	Default	Access
XmNautoDragModel	XmCAutoDragModel	XtEnum	XmAUTO_DRAG_ENABLED	G
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	CSG
XmNscrolledWindowMarginWidth	XmCScrolledWindowMarginWidth	Dimension	0	CSG
XmNscrollingPolicy	XmCScrollingPolicy	unsigned char	XmAPPLICATION_DEFINED	CG
XmNspacing	XmCSpacing	Dimension	4	CSG
XmNverticalScrollBar	XmCVerticalScrollBar	Widget	dynamic	CSG
XmNvisualPolicy	XmCVisualPolicy	unsigned char	dynamic	C
XmNworkWindow	XmCWorkWindow	Widget	NULL	CSG

XmNautoDragModel
: In Motif 2.0 and later, specified whether automatic drag is enabled. Possible values:

```
XmAUTO_DRAG_ENABLED      XmAUTO_DRAG_DISABLED
```

XmNclipWindow
: The widget ID of the clipping area. The clipping window exists only when the XmNvisualPolicy resource is set to XmCONSTANT. The XmNclipWindow resource cannot be set to a new value.

XmNhorizontalScrollBar
: The widget ID of the horizontal ScrollBar.

XmNscrollBarDisplayPolicy
: Controls the placement of ScrollBars, depending on the value of the XmNscrollingPolicy resource. Possible values:

```
XmSTATIC          /* vertical ScrollBar always displays      */
XmAS_NEEDED       /* add ScrollBar when view is clipped      */
```

If XmNscrollingPolicy is set to XmAUTOMATIC, then XmNscrollBarDisplay-Policy defaults to a value of XmAS_NEEDED, and ScrollBars are displayed only when the workspace cannot fit within the clip area. If XmNscrollingPolicy is set to XmAPPLICATION_DEFINED, then XmNscrollBarDisplayPolicy defaults to (and must remain with) a value of XmSTATIC. This means that ScrollBars will always be displayed.

XmNscrollBarPlacement
The positions of the ScrollBars relative to the work window. The default value of this resource depends on the value of the XmNstringDirection resource. Possible values:

```
XmTOP_LEFT          /* vertical ScrollBar on left; horizontal on top       */
XmBOTTOM_LEFT       /* vertical ScrollBar on left; horizontal on bottom    */
XmTOP_RIGHT         /* vertical ScrollBar on right; horizontal on top      */
XmBOTTOM_RIGHT      /* vertical ScrollBar on right; horizontal on bottom   */
```

XmNscrolledWindowMarginHeight
The spacing at the top and bottom of the ScrolledWindow.

XmNscrolledWindowMarginWidth
The spacing at the right and left sides of the ScrolledWindow.

XmNscrollingPolicy
Determines how automatic scrolling occurs. Possible values:

```
XmAUTOMATIC             /* ScrolledWindow handles scrolling    */
XmAPPLICATION_DEFINED   /* application handles scrolling       */
```

XmNspacing
The distance between each ScrollBar and the work window.

XmNverticalScrollBar
The widget ID of the vertical ScrollBar.

XmNvisualPolicy
The visual layout policy of the ScrolledWindow. In Motif 2.0 and later, the resource is obsolete, and the internal widget initialization functions ensure that the policy is consistent. Possible values:

```
XmCONSTANT      /* viewing area is clipped if needed;                  */
                /* default when XmNscrollingPolicy is XmAUTOMATIC      */
XmVARIABLE      /* layout grows or shrinks; default otherwise          */
```

XmNworkWindow
The widget ID of the viewing area.

Callback Resources

ScrolledWindow defines the following callback resources:

Callback	Reason Constant
XmNtraverseObscuredCallback	XmCR_OBSCURED_TRAVERSAL

XmNtraverseObscuredCallback
List of callbacks that are called when the keyboard focus is moved to a widget or gadget that is obscured from view.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int                 reason;              /* reason that callback was called */
    XEvent              *event;              /* event that triggered callback   */
    Widget          traversal_destination;   /* widget or gadget to traverse to */
    XmTraversalDirection  direction;         /* direction of traversal          */
} XmTraverseObscuredCallbackStruct;
```

New Constraint Resources

In Motif 2.0 and later, ScrolledWindow defines the following constraint resources for its children:

Name	Class	Type	Default	Access
XmNscrolledWindowChildType	XmCScrolledWindowChildType	unsigned char	dynamic	CG

XmNscrolledWindowChildType
Specifies the logical type of child of the ScrolledWindow. Possible values:

```
XmHOR_SCROLLBAR     /* horizontal ScrollBar                          */
XmVERT_SCROLLBAR    /* vertical   ScrollBar                          */
XmSCROLL_HOR        /* horizontal ScrollBar - horizontal scrolling only */
XmSCROLL_VERT       /* vertical   ScrollBar - vertical   scrolling only */
XmWORK_AREA         /* work area child                               */
XmCLIP_WINDOW       /* XmClipWindow                                  */
XmNO_SCROLL         /* no child scrolling                            */
```

The values XmSCROLL_HOR, XmSCROLL_VERT, and XmNO_SCROLL are only valid if the scrolling policy is XmAUTOMATIC.

Inherited Resources

ScrolledWindow inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. ScrolledWindow sets the default value of XmNshadowThickness dynamically. The default value of XmNborderWidth is reset to 0 by XmManager.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Widget Hierarchy

When the ScrolledWindow has an XmNvisualPolicy of XmCONSTANT (the XmNscrollingPolicy is XmAUTOMATIC) the ScrolledWindow creates an additional clip widget which is used as the parent of any added work window: the additional widget acts as the logical viewport, and the XmNclipWindow resource is set to the ID of this.

Before Motif 2.0, the clip widget has the name ScrolledWindowClipWindow. In Motif 2.0 and later, the name is changed to ClipWindow: any resources, and XtNameToWidget() or similar code which relies upon the name should be changed accordingly.

Translations

The translations for ScrolledWindow include those from Manager.

Additional Behavior

ScrolledWindow has the following additional behavior when the XmNscrolling-Policy resource is XmAUTOMATIC:

See Also

XmCreateObject(1), XmScrollVisible(1), XmScrolledWindowSetAreas(1), Composite(2), Constraint(2), Core(2), XmManager(2), XmScrollBar(2), XmScrolledList(2), XmScrolledText(2).

Name

XmSelectionBox widget class – a widget for selecting one of a list of alternatives.

Synopsis

Public Header:
<Xm/SelectioB.h>

Class Name:
XmSelectionBox

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmBulletinBoard →
XmSelectionBox

Class Pointer:
xmSelectionBoxWidgetClass[1]

Instantiation:
widget = `XmCreateSelectionBox` (parent, name,...)
or
widget = `XtCreateWidget` (name, XmSelectionBoxWidgetClass,...)

Functions/Macros:
`XmCreateSelectionBox`(), `XmCreateSelectionDialog`(), `XmCreatePromptDialog`(),
`XmIsSelectionBox`(), `XmSelectionBoxGetChild`()

Description

SelectionBox is a composite widget that displays a scrollable list of alternatives from which the user can choose items. A SelectionBox contains a text field in which the user can enter a selection, the scrollable list of selections, labels for the text field and the scrollable list, a separator, and a group of three or four buttons. The names of these components in the SelectionBox are Items, ItemsList, Selection, Text, and Separator, respectively.

In Motif 1.2 and later, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Apply**, **Cancel**, and **Help** by default. The **Apply** button is created but not always managed. If the parent of the SelectionBox is a DialogShell the button is managed, otherwise it is not.

You can customize a SelectionBox by removing existing children or adding new children. Use `XmSelectionBoxGetChild`() to retrieve the widget ID of an existing child and then unmanage the child. With Motif 1.2 and later, multiple widgets can be added as children of a SelectionBox. The first child is considered

1.Erroneously given as XmSelectionBoxWidgetClass in 1st and 2nd editions.

a work area and is placed based on the value of the XmNchildPlacement resource. If a menu bar is added, it is placed at the top of the window. Any buttons are placed after the **OK** button. Any additional children are placed below the message. In Motif 1.1, only a single widget can be added as a child of a SelectionBox. This child is placed below the selection text and acts as a work area.

Traits

SelectionBox uses the XmQTactivatable and XmQTaccessTextual traits.

New Resources

SelectionBox defines the following resources:

Name	Class	Type	Default	Access
XmNapplyLabelString	XmCApplyLabelString	XmString	dynamic	CSG
XmNcancelLabelString	XmCCancelLabelString	XmString	dynamic	CSG
XmNchildPlacement	XmCChildPlacement	unsigned char	XmPLACE_ABOVE_SELECTION	CSG
XmNdialogType	XmCDialogType	unsigned char	dynamic	CG
XmNhelpLabelString	XmCHelpLabelString	XmString	dynamic	CSG
XmNlistItems	XmCItems	XmStringTable	NULL	CSG
XmNlistItemCount	XmCItemCount	int	0	CSG
XmNlistLabelString	XmCListLabelString	XmString	dynamic	CSG
XmNlistVisibleItemCount	XmCListVisibleItemCount	int	dynamic	CSG
XmNminimizeButtons	XmCMinimizeButtons	Boolean	False	CSG
XmNmustMatch	XmCMustMatch	Boolean	False	CSG
XmNokLabelString	XmCOkLabelString	XmString	dynamic	CSG
XmNselectionLabelString	XmCSelectionLabelString	XmString	dynamic	C
XmNtextAccelerators	XmCTextAccelerators	XtAccelerators	default	CSG
XmNtextColumns	XmCColumns	short	dynamic	CSG
XmNtextString	XmCTextString	XmString	dynamic	CSG

XmNapplyLabelString
: The string that labels the **Apply** button. In Motif 1.2 and later, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Apply".

XmNcancelLabelString
: The string that labels the **Cancel** button. In Motif 1.2 and later, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Cancel".

XmNchildPlacement
In Motif 1.2 and later, determines the placement of the work area child. Possible values:

```
XmPLACE_ABOVE_SELECTION     /* above the text area */
XmPLACE_BELOW_SELECTION     /* below the text area */
XmPLACE_TOP                 /* above the list area */
```

XmNdialogType
Determines which children of the SelectionBox widget will be initially created and managed. Possible values:

```
XmDIALOG_WORK_AREA      /* default, when parent isn't a DialogShell */
XmDIALOG_PROMPT         /* all children except list and label       */
XmDIALOG_SELECTION      /* default, when parent is a DialogShell    */
XmDIALOG_COMMAND        /* only list, selection label and text field */
XmDIALOG_FILE_SELECTION /* all standard children                     */
```

Note that in Release 1.1, Command and FileSelectionBox are separate widget classes, and they can no longer be created by setting XmNdialogType.

XmNhelpLabelString
The string that labels the **Help** button. In Motif 1.2 and later, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Help".

XmNlistItems
The items in the SelectionBox list. A call to XtGetValues() returns the actual list items (not a copy), so don't have your application free these items.

XmNlistItemCount
The number of items in the SelectionBox list.

XmNlistLabelString
The string that labels the SelectionBox list. The default string is NULL when the XmN-dialogType resource is set to XmDIALOG_PROMPT; otherwise, in Motif 1.2 and later, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Items".

XmNlistVisibleItemCount
The number of items that appear in the SelectionBox list. The default value depends on the height of the list. This resource has a value of 0 when the XmNdialogType resource is set to XmDIALOG_PROMPT, and otherwise has a default of 8.

XmNminimizeButtons
If False (default), all buttons are standardized to be as wide as the widest button and as high as the highest button. If True, buttons will keep their preferred size.

XmNmustMatch
: If True, the selection that a user types in the text edit field must match an existing entry in the SelectionBox list. If False (default), the typed selection doesn't need to match a list entry. (When the user activates the Ok button, the widget calls one of two lists of callbacks: if this resource is True but the selections don't match, then the SelectionBox widget calls the callbacks specified by the XmNnoMatch-Callback resource; if this resource is False or if the selections do match, then the widget calls the callbacks specified by the XmNokCallback resource.)

XmNokLabelString
: The string that labels the **Ok** button. In Motif 1.2 and later, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "OK"**.**

XmNselectionLabelString
: The string that labels the text edit field. In Motif 1.2 and later, the default value is locale-dependent. In the C locale, and in Motif 1.1, the default value is "Selection".

XmNtextAccelerators
: The translations to add to the SelectionBox's Text widget child. The default bindings allow the up and down keys to be used in selecting list items. This resource is meaningful only when the SelectionBox widget is using the default values in the XmN-accelerators resource.

XmNtextColumns
: The number of columns in the Text widget.

XmNtextString
: The text string that appears in the text edit selection field.

Callback Resources

SelectionBox defines the following callback resources:

Callback	Reason Constant
XmNapplyCallback	XmCR_APPLY
XmNcancelCallback	XmCR_CANCEL
XmNnoMatchCallback	XmCR_NO_MATCH
XmNokCallback	XmCR_OK

XmNapplyCallback
: List of callbacks that are called when the **Apply** button is activated.

XmNcancelCallback
: List of callbacks that are called when the Cancel button is activated.

XmNnoMatchCallback
List of callbacks that are called when the user types a selection in the text area that does not match an item in the list.

XmNokCallback
List of callbacks that are called when the **OK** button is activated. If XmNmustMatch is True and the selection text does not match an item in the list, the XmNnoMatchCallback callbacks are called instead.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;     /* the reason that the callback was called */
    XEvent      *event;     /* event structure that triggered callback */
    XmString    value;      /* selection string that was either chosen */
                            /* from the SelectionBox list or typed in */
    int         length;     /* number of bytes of value            */
} XmSelectionBoxCallbackStruct;
```

Inherited Resources

SelectionBox inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. The default value of XmNborderWidth is reset to 0 by XmManager. BulletinBoard sets the values of XmNinitialFocus to the text entry area, XmN-defaultButton to the Cancel button, and resets the default XmNshadowThickness from 0 to 1 if the SelectionBox is a child of a DialogShell.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNlabelFontList	XmBulletinBoard
XmNallowOverlap	XmBulletinBoard	XmNlabelRenderTable	XmBulletinBoard
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNautoUnmanage	XmBulletinBoard	XmNmapCallback	XmBulletinBoard
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNmarginHeight	XmBulletinBoard
XmNborderColor	Core	XmNmarginWidth	XmBulletinBoard
XmNborderPixmap	Core	XmNnavigationType	XmManager
XmNborderWidth	Core	XmNnoResize	XmBulletinBoard
XmNbottomShadowColor	XmManager	XmNnumChildren	Composite
XmNbottomShadowPixmap	XmManager	XmNpopupHandlerCallback	XmManager
XmNbuttonFontList	XmBulletinBoard	XmNresizePolicy	XmBulletinBoard

Resource	Inherited From	Resource	Inherited From
XmNbuttonRenderTable	XmBulletinBoard	XmNscreen	Core
XmNcancelButton	XmBulletinBoard	XmNsensitive	Core
XmNchildren	Composite	XmNshadowThickness	XmManager
XmNcolormap	Core	XmNshadowType	XmBulletinBoard
XmNdefaultButton	XmBulletinBoard	XmNstringDirection	XmManager
XmNdefaultPosition	XmBulletinBoard	XmNtextFontList	XmBulletinBoard
XmNdepth	Core	XmNtextRenderTable	XmBulletinBoard
XmNdestroyCallback	Core	XmNtextTranslations	XmBulletinBoard
XmNdialogStyle	XmBulletinBoard	XmNtopShadowColor	XmManager
XmNdialogTitle	XmBulletinBoard	XmNtopShadowPixmap	XmManager
XmNfocusCallback	XmBulletinBoard	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNunmapCallback	XmBulletinBoard
XmNhighlightColor	XmManager	XmNuserData	XmManager
XmNhighlightPixmap	XmManager	XmNwidth	Core
XmNinitialFocus	XmManager	XmNx	Core
XmNinitialResourcesPersistent	Core	XmNy	Core
XmNinsertPosition	Composite		

Translations

The translations for SelectionBox are inherited from BulletinBoard.

Action Routines

SelectionBox defines the following action routines:

SelectionBoxUpOrDown(flag):

This action applies when the location cursor is within the item list. This action selects a list item from one of four possible positions and uses this item to replace the selection text. A flag value of 0, 1, 2, or 3 selects the previous, next, first, or last item, respectively. These four action routines are respectively bound to KUp, KDown, KBeginData, and KEndData, which represent four of the default accelerators in the XmNtextAccelerators resource.

SelectionBoxRestore()

Like SelectionBoxUpOrDown except that this action replaces the selection text with the current list item. This action clears the selection text if no list item is currently selected. This action routine is bound to KRestore, a default accelerator for XmNtextAccelerators.

Additional Behavior

SelectionBox has the following additional behavior:

MAny KCancel
: For a sensitive **Cancel** button, invokes the XmNactivateCallback callbacks.

KActivate
: For the button that has keyboard focus (or else the default button), invokes the callbacks in XmNactivateCallback. In a List or Text widget, this event calls the associated List or Text action before the associated SelectionBox action.

<Ok Button Activated>
: Invokes the XmNokCallback callback or the XmNnoMatchCallback if XmN-mustMatch is True and the text does not match an item in the list.

<Apply Button Activated>
: Invokes the XmNapplyCallback callbacks.

<Cancel Button Activated>
: Invokes the XmNcancelCallback callbacks.

<Help Button Activated>
: Invokes the XmNhelpCallback callbacks.

<MapWindow>
: Invokes the callbacks for XmNmapCallback if the parent is a DialogShell.

<UnmapWindow>
: Invokes the callbacks for XmNunmapCallback if the parent is a DialogShell.

See Also

`XmCreateObject`(1), `XmSelectionBoxGetChild`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmBulletinBoard`(2), `XmManager`(2), `XmPromptDialog`(2), `XmSelectionDialog`(2).

Name

XmSelectionDialog – an unmanaged SelectionBox as a child of a Dialog Shell.

Synopsis

Public Header:
<Xm/SelectioB.h>

Instantiation:
widget = `XmCreateSelectionDialog` (parent, name,...)

Functions/Macros:
`XmCreateSelectionBox`(), `XmCreateSelectionDialog`(), `XmSelectionBoxGetChild`()

Description

An XmSelectionDialog is a compound object created by a call to XmCreateSelectionDialog() that an application can use to allow a user to make a selection from a dialog box. A SelectionDialog consists of a DialogShell with an unmanaged SelectionBox widget as its child. The SelectionBox resource XmNdialogType is set to XmDIALOG_SELECTION.

A SelectionDialog displays a scrollable list of alternatives from which the user can choose items. A SelectionDialog also contains a text field in which the user can edit a selection, labels for the text field and for the scrollable list, and four buttons. In Motif 1.2 and later, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Apply**, **Cancel**, and **Help** by default.

Default Resource Values

A SelectionDialog sets the following default values for SelectionBox resources:

Name	Default
XmNdialogType	XmDIALOG_SELECTION

Widget Hierarchy

When a SelectionDialog is created with a specified name, the DialogShell is named name*_popup* and the SelectionBox is called *name***.**

See Also

`XmCreateObject`(1), `XmSelectionBoxGetChild`(1), `XmDialogShell`(2), `XmSelectionBox`(2).

Name

XmSeparator widget class – a widget that draws a line to separate other widgets visually.

Synopsis

Public Header:
<Xm/Separator.h>

Class Name:
XmSeparator

Class Hierarchy:
Core →XmPrimitive →XmSeparator

Class Pointer:
xmSeparatorWidgetClass

Instantiation:
widget = `XmCreateSeparator` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmSeparatorWidgetClass,...)

Functions/Macros:
`XmCreateSeparator`(), `XmIsSeparator`()

Description

A Separator is a widget that draws a horizontal or vertical line between components in an application. Several line styles are available for the Separator. A pixmap separator can also be made by specifying a pixmap for the Core resource XmNbackgroundPixmap and then setting XmNseparatorType to XmNO_LINE.

Traits

Separator holds the XmQTmenuSavvy trait, which is inherited by any derived classes.

New Resources

Separator defines the following resources:

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
The spacing on either end of the Separator. This would be the left and right margins for a horizontally drawn Separator and the top and bottom margins for a vertically drawn Separator.

XmNorientation
The direction in which to display the Separator. Possible values:

```
XmVERTICAL      /* top-to-bottom creation  */
XmHORIZONTAL    /* left-to-right creation  */
```

XmNseparatorType
The line style in which to draw the Separator. Possible values:

```
XmNO_LINE
XmSINGLE_DASHED_LINE
XmSHADOW_ETCHED_IN
XmSINGLE_LINE
XmDOUBLE_DASHED_LINE
XmSHADOW_ETCHED_OUT
XmDOUBLE_LINE
```

Inherited Resources

Separator inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. Separator sets the default values of XmNhighlightThickness to 0 XmNtraversalOn to False. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNhighlightThickness	XmPrimitive
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNlayoutDirection	XmPrimitive
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnavigationType	XmPrimitive
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmPrimitive	XmNsensitive	Core
XmNbottomShadowPixmap	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNcolormap	Core	XmNtopShadowColor	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNdepth	Core	XmNtranslations	Core
XmNdestroyCallback	Core	XmNtraversalOn	XmPrimitive
XmNforeground	XmPrimitive	XmNunitType	XmPrimitive
XmNheight	Core	XmNuserData	XmPrimitive
XmNhelpCallback	XmPrimitive	XmNwidth	Core
XmNhighlightColor	XmPrimitive	XmNx	Core
XmNhighlightOnEnter	XmPrimitive	XmNy	Core
XmNhighlightPixmap	XmPrimitive		

See Also

`XmCreateObject`(1), `Core`(2), `XmPrimitive`(2).

Name

XmSeparatorGadget widget class – a gadget that draws a line to separate other widgets visually.

Synopsis

Public Header:
<Xm/SeparatoG.h>

Class Name:
XmSeparatorGadget

Class Hierarchy:
Object →RectObj →XmGadget →XmSeparatorGadget

Class Pointer:
xmSeparatorGadgetClass

Instantiation:
widget = `XmCreateSeparatorGadget` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmSeparatorGadgetClass,...)

Functions/Macros:
`XmCreateSeparatorGadget`(), `XmIsSeparatorGadget`()

Description

SeparatorGadget is the gadget variant of Separator. SeparatorGadget's new resources are the same as those for Separator.

Traits

SeparatorGadget holds the XmQTcareParentVisual, XmQTaccessColors, and XmQTmenuSavvy traits, which are inherited by any derived class.

Inherited Resources

SeparatorGadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. SeparatorGadget sets the default values of XmNhighlightThickness to 0 and XmNtraversalOn to False. The default value of XmNborderWidth is reset to 0 by Gadget.

Resource	Inherited From	Resource	Inherited From
XmNancestorSensitive	RectObj	XmNhighlightThickness	XmGadget
XmNbackground	XmGadget	XmNlayoutDirection	XmGadget
XmNbackgroundPixmap	XmGadget	XmNnavigationType	XmGadget
XmNbottomShadowColor	XmGadget	XmNsensitive	RectObj
XmNbottomShadowPixmap	XmGadget	XmNshadowThickness	XmGadget
XmNborderWidth	RectObj	XmNtopShadowColor	XmGadget

Resource	Inherited From	Resource	Inherited From
XmNdestroyCallback	Object	XmNtopShadowPixmap	XmGadget
XmNforeground	XmGadget	XmNtraversalOn	XmGadget
XmNheight	RectObj	XmNunitType	XmGadget
XmNhelpCallback	XmGadget	XmNuserData	XmGadget
XmNhighlightColor	XmGadget	XmNwidth	RectObj
XmNhighlightOnEnter	XmGadget	XmNx	RectObj
XmNhighlightPixmap	XmGadget	XmNy	RectObj

See Also

`XmCreateObject`(1), `Object`(2), `RectObj`(2), `XmGadget`(2), `XmSeparator`(2).

Name

XmSimpleSpinBox widget class – a widget for cycling through a set of choices

Synopsis

Public Header:
<Xm/SSpinB.h>

Class Name:
XmSimpleSpinBox

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmSpinBox →XmSimpleSpinBox

Class Pointer:
xmSimpleSpinBoxWidgetClass

Instantiation:
widget = `XmCreateSimpleSpinBox` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmSimpleSpinBoxWidgetClass,...)

Functions/Macros:
`XmSimpleSpinBoxAddItem`(), `XmSimpleSpinBoxDeletePos`(), `XmSimpleSpinBoxSetItem`(), `XmCreateSimpleSpinBox`()

Availability

Motif 2.1 and later.

Description

A subclass of SpinBox which allows the user to increment or decrement the value in a TextField, or to cycle through a set of values. The TextField is created internally by the widget.

The SimpleSpinBox may not be subclassed.

New Resources

SimpleSpinBox defines the following resources:

Name	Class	Type	Default	Access
XmNarrowSensitivity	XmCArrowSensitivity	unsigned char	XmARROWS_SENSITIVE	CSG
XmNcolumns	XmCColumns	short	20	CSG
XmNdecimalPoints	XmCDecimalPoints	short	0	CSG
XmNeditable	XmCEditable	Boolean	True	CSG
XmNincrementValue	XmCIncrementValue	int	1	CSG
XmNmaximumValue	XmCMaximumValue	int	10	CSG

Name	Class	Type	Default	Access
XmNminimumValue	XmCMinimumValue	int	0	CSG
XmNnumValues	XmCNumValues	int	0	CSG
XmNposition	XmCPosition	int	0	CSG
XmNpositionType	XmCPositionType	unsigned char	XmPOSITION_VALUE	CG
XmNspinBoxChildType	XmCSpinBoxChildType	unsigned char	XmString	CSG
XmNtextField	XmCTextField	Widget	dynamic	G
XmNvalues	XmCValues	XmStringTable	NULL	CSG
XmNwrap	XmCWrap	Boolean	True	CSG

XmNarrowSensitivity
Specifies the sensitivity of the arrows within the SimpleSpinBox. Possible values:

```
XmARROWS_SENSITIVE              /* both arrows sensitive          */
XmARROWS_DECREMENT_SENSITIVE
                                /* increment arrowbutton insensitive */
XmARROWS_INCREMENT_SENSITIVE
                                /* decrement arrowbutton insensitive */
XmARROWS_INSENSITIVE            /* both arrows insensitive        */
```

XmNcolumns
Specifies the number of columns in the TextField.

XmNdecimalPoints
A positive integer that determines how the TextField's value will be displayed. The decimal point in the TextField's value gets shifted to the right, and this resource specifies the number of decimal places to shift. For example, if the TextField's value is 5678, then setting the XmNdecimalPoints[1] resource to 2 causes the widget to display the value as 56.78. The resource has no effect if XmNspinBoxChildType is not XmNUMERIC.

XmNeditable
Specifies whether the TextField accepts user input.

XmNincrementValue
Specifies the amount to increment the XmNposition resource. The resource has no effect if XmNspinBoxChildType is not XmNUMERIC.

1.Erroneously given as XmdecimalPoints in 2nd edition.

XmNmaximumValue
Specifies the largest value. The resource has no effect if XmNspinBoxChildType is not XmNUMERIC.

XmNminimumValue
Specifies the smallest value. The resource has no effect if XmNspinBox-ChildType is not XmNUMERIC.

XmNnumValues
Specifies the number of items in the list determined by the XmNvalues resource. The resource has no effect if XmNspinBoxChildType is not XmSTRING.

XmNposition
Depends upon the value of the XmNpositionType and XmNspinBoxChildType resources, and is used to calculate the current value of the SimpleSpinBox.

If XmNspinBoxChildType is XmSTRING, the position resource is used simply as an index into the XmNvalues array.

If XmNspinBoxChildType is XmNUMERIC and XmNpositionType is XmPOSITION_VALUE, the position resource is used directly for the actual value to display. The position value is bounded by XmNminimumValue and XmNmaximumValue. When XmNpositionType is XmPOSITION_INDEX, the position is interpreted as an index into a set of values, bounded by the XmNminimumValue and XmNmaximumValue resource. A number is in the set depending upon the XmNincrementValue resource: position zero corresponds to the value XmNminimumValue, position n is the value given by:

XmNminimumValue + (n * XmNincrementValue)

XmNpositionType
Specifies how the XmNposition resource is to be interpreted. Possible values:

```
XmPOSITION_INDEX    /* position is an index into an array */
XmPOSITION_VALUE    /* position is a direct value        */
```

XmNspinBoxChildType
Specifies the type of data to be displayed. Possible values:

```
XmNUMERIC       /* value is defined by maximum, minimum,    */
                /* increment resources */
XmSTRING        /* value is defined by the values array                */
```

XmNtextField
Specifies the text field created by the SimpleSpinBox to display the current value.

XmNvalues
Specifies the array of compound strings forming the validset of items for the SimpleSpinBox. Only has effect if XmNspinBoxChildType is XmSTRING.

XmNwrap
Specifies whether the SpinBox wraps around the set of values.

Inherited Resources

SimpleSpinBox inherits the resources shown below. The resources are listed alphabetically, along with the superclass that defines them. SimpleSpinBox resets the default value of XmNshadowThickness to 1.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNarrowLayout	XmSpinBox	XmNmappedWhenManaged	Core
XmNarrowOrientation	XmSpinBox	XmNmarginHeight	XmSpinBox
XmNarrowSize	XmSpinBox	XmNmarginWidth	XmSpinBox
XmNbackground	Core	XmNmodifyVerifyCallback	XmSpinBox
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNrepeatDelat	XmSpinBox
XmNbottomShadowColor	XmManager	XmNscreen	Core
XmNbottomShadowPixmap	XmManager	XmNsensitive	Core
XmNchildren	Composite	XmNshadowThickness	XmManager
XmNcolormap	Core	XmNspacing	XmSpinBox
XmNdefaultArrowSensitivity	XmSpinBox	XmNstringDirection	XmManager
XmNdepth	Core	XmNtopShadowColor	XmManager
XmNdestroyCallback	Core	XmNtopShadowPixmap	XmManager
XmNdetailShadowThickness	XmSpinBox	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNvalueChangedCallback	XmSpinBox
XmNhighlightPixmap	XmManager	XmNwidth	Core
XmNinitialDelay	XmSpinBox	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Widget Hierarchy

When a SimpleSpinBox is created with a specified name, the automatically created TextField child is named name*_TF*.

Translations

The translations for SimpleSpinBox are those of SpinBox.

See Also

`XmSimpleSpinBoxAddItem`(1), `XmSimpleSpinBoxDeletePos`(1), `XmSimpleSpinBoxSetItem`(1), `XmCreateSimpleSpinBox`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmManager`(2), `XmSpinBox`(2).

Name

XmSpinBox widget class – a composite widget which controls cycling through a set of choices

Synopsis

Public Header:
<Xm/SpinB.h>

Class Name:
XmSpinBox

Class Hierarchy:
Core →Composite →Constraint →XmManager →XmSpinBox

Class Pointer:
xmSpinBoxWidgetClass

Instantiation:
widget = `XmCreateSpinBox` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmSpinBoxWidgetClass,...)

Functions/Macros:
`XmSpinBoxValidatePosition`(), `XmCreateSpinBox`()

Availability

Motif 2.0 and later.

Description

SpinBox is a manager which allows the user to cycle through sets of choices. At the minimum, the widget contains a single Text or TextField, which is associated with a group of values. A pair of ArrowButtons are provided which, when pressed, insert the next or previous group item into the Text.

SpinBox can control multiple traversable children, each possessing their own set of values. The ArrowButtons cycle the values of the child with the current focus.

The values associated with any textual child of the SpinBox are specified through constraint resources. Logically, a textual child is considered to be either numeric or string based, as specified by the XmNspinBoxChildType resource. If XmNUMERIC, a set of constraints control the current value, and place an upper and lower bound upon the range through which the value may rotate. If XmSTRING, an array of compound strings is specified for the child, and the ArrowButtons cycle through the set of values by incrementing or decrementing an index into the array.

The location of the ArrowButtons relative to the textual children is controlled through the XmNarrowLayout resource, although this is affected by any specified XmNlayoutDirection. The arrows automatically created by the SpinBox are drawn, and not real widgets.

Traits

SpinBox holds the XmQTnavigator trait, which is inherited by any derived classes, and uses the XmQTaccessTextual trait.

New Resources

SpinBox defines the following resources:

Name	Class	Type	Default	Access
XmNarrowLayout	XmCArrowLayout	unsigned char	XmARROWS_END	CSG
XmNarrowOrientation	XmCArrowOrientation	unsigned char	XmARROWS_VERTICAL	CSG
XmNarrowSize	XmCArrowSize	Dimension	16	CSG
XmNdefaultArrowSensitivity	XmCDefaultArrowSensitivity	unsigned char	XmARROWS_SENSITIVE	CSG
XmNdetailShadowThickness	XmCDetailShadowThickness	Dimension	dynamic	CSG
XmNinitialDelay	XmCInitialDelay	unsigned int	250	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	2	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	2	CSG
XmNrepeatDelat	XmCRepeatDelay	unsigned int	200	CSG
XmNspacing	XmCSpacing	Dimension	2	CSG

XmNarrowLayout

Specifies the location of the drawn arrows relative to the textual children. Possible values:

```
XmARROWS_BEGINNING          /* both arrows placed vertically   before text  */
XmARROWS_END                /* both arrows placed vertically   after text   */
XmARROWS_FLAT_BEGINNING   /* both arrows placed horizontally before text*/
XmARROWS_FLAT_END           /* both arrows placed horizontally after text */
XmARROWS_SPLIT              /* one arrow placed at each end                */
```

The interpretation of beginning and end depends upon the XmNlayoutDirection resource inherited from XmManager. This also affects whether it is the increment or decrement arrow which is to the left or right. If the layout direction is XmLEFT_TO_RIGHT, the decrement arrow is to the left, and both XmARROWS_BEGINNING and XmARROWS_FLAT_BEGINNING place both arrows on the left of the Text. If the direction is XmRIGHT_TO_LEFT, it is the increment arrow which is to the left, XmARROWS_END and XmARROWS_FLAT_END which place both arrows on the left of the Text.

XmNarrowOrientation
: In Motif 2.1, specifies whether arrows point vertically or horizontally. If the orientation is XmARROWS_VERTICAL, the decrement arrow points downwards, and the increment arrow points upwards. If orientation is XmARROWS_HORIZONTAL, the decrement arrow points to the left, and the increment arrow points to the right. This is reversed if the XmNlayoutDirection is XmRIGHT_TO_LEFT.

  Note that this is not the same as the XmNarrowLayout resource, which positions the components of the SpinBox relative to each other, whereas XmNarrowOrientation rotates an arrow within its given position.

XmNarrowSize
: Specifies the width (and height) of the drawn arrows, measured in pixels.

XmNdefaultArrowSensitivity
: Specifies the default sensitivity of the drawn arrows to user input. The resource is overridden by the XmNarrowSensitivity constraint resource of the textual child which has the focus. Possible values:

```
XmARROWS_DECREMENT_SENSITIVE
                        /* only the decrement arrow accepts input */
XmARROWS_INCREMENT_SENSITIVE
                        /* only the increment arrow accepts input */
XmARROWS_INSENSITIVE    /* both arrows are insensitive            */
XmARROWS_SENSITIVE      /* both arrows are sensitive              */
```

XmNdetailShadowThickness
: Specifies the thickness of the shadow used for drawing the arrow shapes. Values of 0, 1, or 2 are implemented. In Motif 2.0, the default value is 2. In Motif 2.1, the default value depends upon the XmDisplay XmNenableThinThickness resource: if True the default is 1, otherwise 2.

XmNinitialDelay
: Specifies the time interval in milliseconds (after pressing the mouse) which is to elapse before the SpinBox triggers automatic spinning. If the value is zero, the value defaults to that specified by the XmNrepeatDelay resource.

XmNmarginWidth
: Specifies the space between the left edge of the SpinBox and the leftmost child, and the space between the right edge of the SpinBox and the rightmost child.

XmNmarginHeight
: Specifies the space between the top edge of the SpinBox and the topmost child, and the space between the bottom edge of the SpinBox and the bottom child.

XmNrepeatDelay
Specifies the time interval in milliseconds (when holding down the mouse) which is to elapse before the SpinBox triggers automatic spinning: with the mouse held down, the SpinBox repeatedly spins until such time as the mouse is released. If the value is zero, automatic spinning is disabled.

XmNspacing
Specifies the horizontal and vertical spacing between items in the SpinBox.

New Constraint Resources

SpinBox defines the following constraint resources for its children:

Name	Class	Type	Default	Access
XmNarrowSensitivity	XmCArrowSensitivity	unsigned char	XmARROWS_DEFAULT_SENSITIVITY	CSG
XmNdecimalPoints	XmCDecimalPoints	short	0	CSG
XmNincrementValue	XmCIncrementValue	int	1	CSG
XmNmaximumValue	XmCMaximumValue	int	10	CSG
XmNminimumValue	XmCMinimumValue	int	0	CSG
XmNnumValues	XmCNumValues	int	0	CSG
XmNposition	XmCPosition	int	0	CSG
XmNpositionType	XmCPositionType	unsigned char	XmPOSITION_VALUE	CSG
XmNspinBoxChildType	XmCSpinBoxChildType	unsigned char	XmString	CSG
XmNvalues	XmCValues	XmStringTable	NULL	CSG
XmNwrap	XmCWrap	Boolean	True	CSG

XmNarrowSensitivity
Specifies the sensitivity of the arrowbuttons. Possible values:

```
XmARROWS_DEFAULT_SENSITIVITY
                        /* inherit XmNdefaultArrowSensitivity   */
XmARROWS_DECREMENT_SENSITIVE
                        /* only decrement arrow accepts input   */
XmARROWS_INCREMENT_SENSITIVE
                        /* only increment arrow accepts input   */
XmARROWS_INSENSITIVE    /* both arrows are insensitive          */
XmARROWS_SENSITIVE      /* both arrows are sensitive            */
```

XmNdecimalPoints
Specifies the number of decimal places used when displaying numeric values. The value is zero padded where necessary.

XmNincrementValue
Specifies the amount to increment or decrement the numeric value in the SpinBox text when the increment or decrement arrow is pressed. The resource is only used when the SpinBox type is XmNUMERIC.

XmNmaximumValue
Specifies the greatest value allowed in an XmNUMERIC SpinBox.

XmNminimumValue
Specifies the smallest value allowed in an XmNUMERIC SpinBox.

XmNnumValues
Specifies the number of values in the XmNvalues array. The resource is only used when the SpinBox type is XmSTRING.

XmNposition
The interpretation of this resource depends upon the value of the XmNpositionType and XmNspinBoxChildType resources.

When XmNpositionType is XmPOSITION_INDEX, the position is interpreted as an index into an array of values. If XmNspinBoxChildType is XmSTRING, these are defined by the XmNvalues resource. If the child type is XmNUMERIC, then the array of values is a set of numbers bounded by the XmNminimumValue and XmNmaximumValue resource. A number is in the set depending upon the XmNincrementValue resource: position zero corresponds to the value XmNminimumValue, position n is the value given by:

XmNminimumValue + (n * XmNincrementValue)

When the XmNpositionType is XmPOSITION_VALUE and the XmNspinBoxChildType is XmNUMERIC, the position is the actual value to display, and is bounded by XmNminimumValue and XmNmaximumValue. If XmNspinBoxChildType is XmSTRING, position remains an index into the XmNvalues resource array.

XmNpositionType
Specifies how the XmNposition resource is to be interpreted. Possible values:

```
XmPOSITION_INDEX   /* position is an index into an array      */
XmPOSITION_VALUE   /* position is a direct value              */
```

XmNspinBoxChildType
Specifies the type of data to be displayed. Possible values:

```
XmNUMERIC          /* choices defined by maximum,             */
                   /* minimum, increment values               */
XmSTRING           /* choices defined by the values array     */
```

XmNvalues
Specifies the array of compound strings forming the set of items for the SimpleSpinBox. The resource only has effect if XmNspinBoxChildType is XmSTRING.

XmNwrap
Specifies whether the SpinBox wraps around the set of values. If the current position is at the maximum value, the increment arrow is pressed, and XmNwrap is True, the requested position becomes the minimum value. Similarly if the current position is at the minimum, and the decrement arrow is pressed with wrap enabled, the requested position is at the maximum. If wrap is False in the given circumstances, the bell is rung and the selection is unchanged.

Callback Resources

SpinBox defines the following callback resources:

Callback	Reason Constant
XmNmodifyVerifyCallback	XmCR_SPIN_FIRST XmCR_SPIN_LAST XmCR_SPIN_NEXT XmCR_SPIN_PRIOR
XmNvalueChangedCallback	XmCR_OK XmCR_SPIN_FIRST XmCR_SPIN_LAST XmCR_SPIN_NEXT XmCR_SPIN_PRIOR

XmNmodifyVerifyCallback
List of callbacks called before the SpinBox position is changed.

XmNvalueChangedCallback
List of callbacks called after the SpinBox position is changed.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;           /* the reason that the callback was called */
    XEvent      *event;           /* points to event that triggered callback */
    Widget      widget;           /* the textual child affected by callback  */
    Boolean     doit;             /* whether to perform the changes          */
    int         position;         /* specifies the index of the next value   */
    XmString    value;            /* specifies the next value                */
    Boolean     crossed_boundary; /* whether the SpinBox has wrapped         */
} XmSpinBoxCallbackStruct;
```

reason indicates why the callback is invoked. If *reason* is XmCR_SPIN_FIRST, the SpinBox position is either XmNminimum, or the index of the first item in the XmNvalues array, depending upon whether the SpinBox type is XmNUMERIC or XmSTRING respectively. If *reason* is XmCR_SPIN_LAST, the position is either XmNmaximum, or the index of the last item in the XmNvalues array. If *reason* is XmCR_SPIN_NEXT, the increment arrow is armed. If *reason* is XmCR_SPIN_PRIOR, the decrement arrow is armed. If *reason* is XmCR_OK, an arrow is disarmed.

widget is the ID of the textual component affected by the callback. This is the text which has the current focus.

doit is a flag indicating whether the action is to be performed. The default value is True, although a programmer may set the value False for whatever reason to prevent the item associated with position and value being displayed at the current instant. *doit* is only relevant to XmNmodifyVerifyCallback callbacks.

position is equivalent to the XmNposition resource, and specifies the next position to display. An XmNmodifyVerifyCallback procedure may alter the value in order to force the SpinBox to display a particular item.

value specifies the new item to be displayed in widget. *value* is a temporary compound string which is freed after callback procedures are finished. The programmer should copy *value* if this is required outside of the callback procedures.

crossed_boundary[1] specifies whether the SpinBox has crossed the upper or lower bound as specified by XmNminimum and XmNmaximum, or the first and last compound string in the XmNvalues array.

Inherited Resources

SpinBox inherits the resources shown below. The resources are listed alphabetically, along with the superclass that defines them.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNinsertPosition	Composite
XmNancestorSensitive	Core	XmNlayoutDirection	XmManager
XmNbackground	Core	XmNmappedWhenManaged	Core
XmNbackgroundPixmap	Core	XmNnavigationType	XmManager
XmNborderColor	Core	XmNnumChildren	Composite
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmManager
XmNborderWidth	Core	XmNscreen	Core

1.Erroneously given as crossing_boundary in 2nd edition.

Resource	Inherited From	Resource	Inherited From
XmNbottomShadowColor	XmManager	XmNsensitive	Core
XmNbottomShadowPixmap	XmManager	XmNshadowThickness	XmManager
XmNchildren	Composite	XmNstringDirection	XmManager
XmNcolormap	Core	XmNtopShadowColor	XmManager
XmNdepth	Core	XmNtopShadowPixmap	XmManager
XmNdestroyCallback	Core	XmNtranslations	Core
XmNforeground	XmManager	XmNtraversalOn	XmManager
XmNheight	Core	XmNunitType	XmManager
XmNhelpCallback	XmManager	XmNuserData	XmManager
XmNhighlightColor	XmManager	XmNwidth	Core
XmNhighlightPixmap	XmManager	XmNx	Core
XmNinitialFocus	XmManager	XmNy	Core
XmNinitialResourcesPersistent	Core		

Translations

The translations for SpinBox include those of XmManager.

Event	Action
BSelect Press	SpinBArm()
BSelect Release	SpinBDisarm()
<EnterWindow>	SpinBEnter()
<LeaveWindow)	SpinBLeave()
KUp	SpinBPrior()
KDown	SpinBNext()
KLeft	SpinBLeft()
KRight	SpinBRight()
KBeginData	SpinBFirst()
KEndData	SpinBLast()

Action Routines

SpinBox defines the following action routines:

SpinBArm()

Draws the arrow shape under the pointer in armed form. If the XmNinitialDelay resource is specified, a timer is initialized to the value of the initial delay period. If the mouse is not released before the timer expires, the SpinBox starts automatically selecting successive items in the SpinBox.XmNmodifyVerifyCallback procedures are called with the position element adjusted to the selected value, depending upon the type of arrow, and subsequently XmNvalueChangedCallback procedures are called if the doit element of the XmSpinBoxCallbackStruct is still True after the modify verify callbacks are finished. The position element of the structure is used to determine the value which is inserted into the textual child of the SpinBox which has the focus.

SpinBDisarm()

Draws the arrow shape in disarmed form. If any period specified by the XmNrepeatDelay or XmNinitialDelay periods have not expired, XmNmodifyVerifyCallback procedures are called with the position element adjusted to the required value, and subsequently XmNvalueChangedCallback procedures are called if the doit element of the XmSpinBoxCallbackStruct is still True. The item as reflected by the final value of the position element of the structure is inserted into the current traversable text.

SpinBPrior()

Draws the decrement arrow in armed form, and invokes any XmNmodifyVerifyCallback procedures, with the position element of the XmSpinBoxCallbackStruct suitably decremented, taking into account possible wrapping. Thereafter, if the doit element is still True, any XmNvalueChangedCallback procedures are invoked. The item as reflected by the final value of the position element of the structure is inserted into the current traversable text.

SpinBNext()

Draws the increment arrow in armed form, and invokes any XmNmodifyVerifyCallback procedures, with the position element of the XmSpinBoxCallbackStruct suitably incremented, taking into account possible wrapping. Thereafter, if the doit element is still True, any XmNvalueChangedCallback procedures are invoked. The item as reflected by the final value of the position element of the structure is inserted into the current traversable text.

SpinBLeft()
: If the XmNlayoutDirection is XmLEFT_TO_RIGHT, invokes the SpinBPrior() action, otherwise invokes SpinBNext().

SpinBRight()
: If the XmNlayoutDirection is XmLEFT_TO_RIGHT, invokes the SpinBNext() action, otherwise invokes SpinBPrior().

SpinBFirst()
: XmNmodifyVerifyCallback procedures are invoked with the position element of the XmSpinBoxCallbackStruct set to zero. Thereafter, if the doit element is still True, any XmNvalueChangedCallback procedures are invoked. The item as reflected by the final value of the position element of the structure is inserted into the current traversable text.

SpinBLast()
: XmNmodifyVerifyCallback procedures are invoked with the position element of the XmSpinBoxCallbackStruct set to the last position. Thereafter, if the doit element is still True, any XmNvalueChangedCallback procedures are invoked. The item as reflected by the final value of the position element of the structure is inserted into the current traversable text.

SpinBEnter()
: If the containing shell has a focus policy of XmPOINTER, draws the highlight border around the child traversable text widget which has the focus.

SpinBLeave()
: If the containing shell has a focus policy of XmPOINTER, erases the highlight border around the child traversable text widget which has the focus.

See Also

`XmSpinBoxValidatePosition`(1), `XmCreateObject`(1), `Composite`(2), `Constraint`(2), `Core`(2), `XmManager`(2).

Name

XmTemplateDialog – an unmanaged MessageBox as a child of DialogShell.

Synopsis

Public Header:
<Xm/MessageB.h>

Instantiation:
widget = `XmCreateTemplateDialog` (parent, name,...)

Functions/Macros:
`XmCreateTemplateDialog`(), `XmMessageBoxGetChild`()

Description

An XmTemplateDialog is a compound object created by a call to `XmCreateTemplateDialog`() that an application can use to present a customized message to the user. A TemplateDialog consists of a DialogShell with an unmanaged MessageBox widget as its child. The MessageBox resource XmNdialogType is set to XmDIALOG_TEMPLATE. By default, a TemplateDialog includes only a separator. An application can create a customized dialog by adding children to the TemplateDialog. To create the standard components associated with a MessageBox, an application needs only specify the label string and callback resources for the desired buttons, and the TemplateDialog automatically creates the relevant button. Setting either the XmNmessageString or XmNsymbolPixmap resource creates a message or a symbol Label.

Default Resource Values

A TemplateDialog sets the following default values for MessageBox resources:

Name	Default
XmNdialogType	XmDIALOG_TEMPLATE

Widget Hierarchy

When a TemplateDialog is created with a specified name, the DialogShell is named name*_popup* and the MessageBox is called *name*.

See Also

`XmCreateObject`(1), `XmMessageBoxGetChild`(1), `XmDialogShell`(2), `XmMessageBox`(2).

Name

XmText widget class – text-editing widget.

Synopsis

Public Header:
<Xm/Text.h>

Class Name:
XmText

Class Hierarchy:
Core →XmPrimitive →XmText

Class Pointer:
xmTextWidgetClass

Instantiation:
widget = `XmCreateText` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmTextWidgetClass,...)

Functions/Macros:
`XmCreateScrolledText`(), `XmCreateText`(), `XmIsText`(), XmText... routines

Description

A Text widget provides a text editor that allows text to be inserted, modified, deleted, and selected. Text provides both single-line and multi-line text editing capabilities.

Traits

Text holds the XmQTaccessTextual and XmQTtransfer traits, which are inherited in any derived classes, and uses the XmQTaccessTextual, XmQTspecifyRenderTable, XmQTnavigator and XmQTscrollFrame traits.

New Resources

Text defines the following resources:

Name	Class	Type	Default	Access
XmNautoShowCursorPosition	XmCAutoShowCursorPosition	Boolean	True	CSG
XmNcursorPosition	XmCCursorPosition	XmTextPosition	0	CSG
XmNeditable	XmCEditable	Boolean	True	CSG
XmNeditMode	XmCEditMode	int	see below	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	5	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	5	CSG

Name	Class	Type	Default	Access
XmNmaxLength	XmCMaxLength	int	largest integer	CSG
XmNsource	XmCSource	XmTextSource	default source	CSG
XmNtotalLines	XmCTotalLines	int	1	CG
XmNtopCharacter	XmNtopCharacter	XmTextPosition	0	CSG
XmNvalue	XmCValue	String	""	CSG
XmNvalueWcs	XmCValueWcs	wchar_t	(Wchar_t *) ""	CSG
XmNverifyBell	XmCVerifyBell	Boolean	dynamic	CSG

XmNautoShowCursorPosition
: If True (default), the visible portion of the Text widget will always contain the insert cursor. The Text widget will scroll its contents, if necessary, to ensure that the cursor remains visible.

XmNcursorPosition
: The location at which to place the current insert cursor. Values for this resource are relative to the beginning of the text; the first character position is defined as 0.

XmNeditable
: If True (default), the user is allowed to edit the text string; if False, the user is not allowed to do so.

XmNeditMode
: Determines which group of keyboard bindings to use. Possible values:

```
XmMULTI_LINE_EDIT     /* key bindings for multi-line edits          */
XmSINGLE_LINE_EDIT    /* key bindings for single line edits (default) */
```

XmNmarginHeight

XmNmarginWidth
: The spacing between the edges of the widget and the text. (Top and bottom edges for height; left and right for width.)

XmNmaxLength
: The maximum length of the text string that a user can enter from the keyboard. This resource does not affect strings that are entered via the XmNvalue resource or the `XmTextSetString`() routine.

XmNsource
: A source that the Text widget uses for displaying text, thereby allowing Text widgets to share the same text source.

XmNtotalLines
: In Motif 2.1, specifies the number of lines within the Text widget buffer. The value takes into account word wrapping.

XmNtopCharacter
: The location of the text to display at the top of the window. Values for this resource are relative to the beginning of the text, with the first character position defined as 0.

XmNvalue
: The string value to display in the Text widget, expressed as a char *. If XmNvalue and XmNvalueWcs are both defined, XmNvalueWcs takes precedence. Use `XtSetValues`() to copy string values to the internal buffer and use `XtGetValues`() to return the value of the internal buffer.

XmNvalueWcs
: In Motif 1.2, the string value to display in the Text widget, expressed as a wchar_t *. If XmNvalue and XmNvalueWcs are both defined, XmNvalueWcs takes precedence. Use `XtSetValues`() to copy string values to the internal buffer and use `XtGetValues`() to return the value of the internal buffer. This resource cannot be set in a resource file.

XmNverifyBell
: If True, a bell will sound when a verification produces no action. The default value depends upon the XmNaudibleWarning resource of any VendorShell ancestor.

Text Input Resources

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
: If True (default), the Text widget's pending delete mode is on, meaning that selected text will be deleted as soon as the next text insertion occurs.

XmNselectionArray
: The array of possible actions caused by multiple mouse clicks. UIL does not define these values for the Text widget. Possible values:

```
XmSELECT_POSITION   /* single-click; reset position of insert cursor */
XmSELECT_WORD       /* double-click; select a word                   */
XmSELECT_LINE       /* triple-click; select a line                   */
XmSELECT_ALL        /* quadruple-click; select all text              */
```

XmNselectionArrayCount
: The number of items in the array specified by XmNselectionArray.

XmNselectThreshold
The number of pixels the insertion cursor must be dragged during selection in order to select the next character.

Text Output Resources

Name	Class	Type	Default	Access
XmNblinkRate	XmCBlinkRate	int	500	CSG
XmNcolumns	XmCColumns	short	dynamic	CSG
XmNcursorPositionVisible	XmCCursorPositionVisible	Boolean	dynamic	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	CSG
XmNresizeHeight	XmCResizeHeight	Boolean	False	CSG
XmNresizeWidth	XmCResizeWidth	Boolean	False	CSG
XmNrows	XmCRows	short	dynamic	CSG
XmNwordWrap	XmCWordWrap	Boolean	False	CSG

XmNblinkRate
The time in milliseconds that the cursor spends either being visible or invisible. A value of 0 prevents the cursor from blinking.

XmNcolumns
The number of character spaces that should fit horizontally in the text window. The XmNwidth resource determines the default value of XmNcolumns, but if no width has been set, the default is 20. See also XmNrows.

XmNcursorPositionVisible
If True, the text cursor will be visible. In Motif 2.1, if the text widget has an XmPrintShell as ancestor, the default is False. Otherwise the default is True.

XmNfontList
The font list used for the widget's text. In Motif 2.0 and later, the XmFontList is obsolete, and is subsumed into the XmRenderTable. If both a render table and font list are specified, the render table takes precedence.

XmNrenderTable
In Motif 2.0 and later, specifies the render table for the Text. If unspecified, the value of the resource is inherited from the nearest ancestor which holds the XmQTspecifyRenderTable trait, using the XmTEXT_RENDER_TABLE value of the ancestor so found.

XmNresizeHeight
If False (default), the Text widget will not expand vertically to fit all of the text (in other words, the widget will need to have scrollbars so that the rest of the text

can be scrolled into view). If True, the Text widget always begins its display with the text at the beginning of the source. This resource has no effect in a ScrolledText widget whose XmNscrollVertical resource is set to True.

XmNresizeWidth
If False (default), the Text widget will not expand horizontally to fit its text. If True, the widget tries to change its width. This resource has no effect when the XmNwordWrap resource is set to True.

XmNrows
The number of character spaces that should fit vertically in the text window. The XmNheight resource determines the default value of XmNrows, but if no height has been set, the default is 1. This resource is meaningful only when XmNeditMode is XmMULTI_LINE_EDIT. See also XmNcolumns.

XmNwordWrap
If False (default), does not break lines automatically between words (in which case text can disappear beyond the window's edge). If True, breaks lines at spaces, tabs, or newlines. This resource is meaningful only when XmNeditMode is XmMULTI_LINE_EDIT.

Scrolled Text Resources

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
If True, the Text widget adds a horizontal ScrollBar. The default is True; however, the value changes to False if the widget is in a ScrolledWindow whose XmNscrollingPolicy resource is set to XmAUTOMATIC. This resource is meaningful only when XmNeditMode is XmMULTI_LINE_EDIT.

XmNscrollLeftSide
If True, the vertical ScrollBar is placed to the left of the scrolled text window. The default value depends on how the XmNstringDirection resource is set. This resource is meaningful only when XmNeditMode is XmMULTI_LINE_EDIT and when XmNscrollVertical is True.

XmNscrollTopSide
If True, the horizontal ScrollBar is placed above the scrolled text window, rather than below by default.

XmNscrollVertical
If True, the Text widget adds a vertical ScrollBar. The default is True; however, the value changes to False if the widget is in a ScrolledWindow whose XmNscrollingPolicy resource is set to XmAUTOMATIC.

Callback Resources

Text defines the following callback resources:

Callback	Reason Constant
XmNactivateCallback	XmCR_ACTIVATE
XmNdestinationCallback	XmCR_OK
XmNfocusCallback	XmCR_FOCUS
XmNgainPrimaryCallback	XmCR_GAIN_PRIMARY
XmNlosePrimaryCallback	XmCR_LOSE_PRIMARY
XmNlosingFocusCallback	XmCR_LOSING_FOCUS
XmNmodifyVerifyCallback	XmCR_MODIFYING_TEXT_VALUE
XmNmodifyVerifyCallbackWcs	XmCR_MODIFYING_TEXT_VALUE
XmNmotionVerifyCallback	XmCR_MOVING_INSERT_CURSOR
XmNvalueChangedCallback	XmCR_VALUE_CHANGED

XmNactivateCallback
List of callbacks that are called when the user causes the Text widget to be activated.

XmNdestinationCallback
List of callbacks that are called when the Text is the destination of a transfer operation.

XmNfocusCallback
List of callbacks that are called when the Text widget receives the input focus.

XmNgainPrimaryCallback
List of callbacks that are called when the Text widget gains ownership of the primary selection.

XmNlosePrimaryCallback
List of callbacks that are called when the Text widget loses ownership of the primary selection.

XmNlosingFocusCallback
List of callbacks that are called when the Text widget loses the input focus.

XmNmodifyVerifyCallback
List of callbacks that are called before the value of the Text widget is changed. If there are callbacks for both XmNmodifyVerifyCallback and XmNmodifyVerifyCallbackWcs, the XmNmodifyVerifyCallback callbacks are called first.

XmNmodifyVerifyCallbackWcs
List of callbacks that are called before the value of the Text widget is changed. If there are callbacks for both XmNmodifyVerifyCallback and XmNmodifyVerifyCallbackWcs, the XmNmodifyVerifyCallback callbacks are called first.

XmNmotionVerifyCallback
List of callbacks that are called before the insertion cursor is moved in the Text widget.

XmNvalueChangedCallback
List of callbacks that are called after the value of the Text widget is changed.

Callback Structure

Destination callbacks are fully described within the sections covering the Uniform Transfer Model. See `XmTransfer`(1) for more details. For quick reference, a pointer to the following structure is passed to callbacks on the XmNdestinationCallback list:

```
typedef struct {
    int         reason;          /* the reason that the callback is invoked */
    XEvent      *event;          /* points to event that triggered callback */
    Atom        selection;       /* the requested selection type, as an Atom */
    XtEnum      operation;       /* the type of transfer requested          */
    int         flags;       /* whether destination and source are same */
    XtPointer   transfer_id;     /* unique identifier for the request       */
    XtPointer   destination_data;     /* information about the destination  */
    XtPointer   location_data;       /* information about the data         */
    Time        time;            /* time when the transfer operation started */
} XmDestinationCallbackStruct;
```

With the exception of the above destination callback, each callback function is passed the following structure:

```
typedef struct {
    int         reason;      /* the reason that the callback was called */
    XEvent      *event;      /* event structure that triggered callback */
} XmAnyCallbackStruct;
```

In addition, the callback resources XmNlosingFocusCallback, XmNmodifyVerifyCallback, and XmNmotionVerifyCallback reference the following structure:

```
typedef struct {
    int         reason;          /* the reason that the callback was called */
    XEvent      *event;          /* points to event that triggered callback */
    Boolean     doit;            /* do the action (True) or undo it (False) */
    long        currInsert;      /* the insert cursor's current position    */
```

```
    long        newInsert;      /* desired new position of insert cursor  */
    long        startPos;       /* start of text to change                */
    long        endPos;         /* end of text to change                  */
    XmTextBlocktext;            /* describes the text to insert           */
} XmTextVerifyCallbackStruct, *XmTextVerifyPtr;
```

start_pos specifies the location at which to start modifying text. *start_pos* is unused if the callback resource is XmNmotionVerifyCallback, and is the same as the *current_insert* member if the callback resource is XmNlosingFocusCallback.

end_pos specifies the location at which to stop modifying text (however, if no text was modified, *end_pos* has the same value as *start_pos*). *end_pos* is unused if the callback resource is XmNmotionVerifyCallback, and is the same as the *current_insert* member if the callback resource is XmNlosingFocusCallback.

text points to the structure below, which specifies information about the text to be inserted.

```
typedef struct {
    char        *ptr;           /* pointer to the text to insert          */
    int         length;         /* length of this text                    */
    XmTextFormat   format;      /* text format (e.g., FMT8BIT, FMT16BIT) */
} XmTextBlockRec, *XmTextBlock;
```

The callback resource XmNmodifyVerifyCallbackWcs references the following structure:

```
typedef struct {
    int         reason;         /* the reason that the callback was called */
    XEvent      *event;         /* structure that triggered callback      */
    Boolean     doit;           /* do the action (True) or undo it (False) */
    long        current_insert; /* the insert cursor's current position   */
    long        new_insert;     /* desired new position of insert cursor  */
    long        start_pos;      /* start of text to change                */
    long        end_pos;        /* end of text to change                  */
    XmTextBlockWcstext;         /* describes the text to insert           */
} XmTextVerifyCallbackStructWcs, *XmTextVerifyPtrWcs;
```

All of the fields in this structure are the same as the fields in the XmTextVerify-CallbackStruct except text, which points to the structure below and specifies information about the text to be inserted.

```
typedef struct {
    wchar_t     *wcsptr;    /* pointer to the text to insert              */
    int         length;     /* length of this text                        */
} XmTextBlockRecWcs, *XmTextBlockWcs;
```

Inherited Resources

Text inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. Text sets the default value of XmNnavigationType to XmTAB_GROUP. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNhighlightThickness	XmPrimitive
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNlayoutDirection	XmPrimitive
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnavigationType	XmPrimitive
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmPrimitive	XmNsensitive	Core
XmNbottomShadowPixmap	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNcolormap	Core	XmNtopShadowColor	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNdepth	Core	XmNtranslations	Core
XmNdestroyCallback	Core	XmNtraversalOn	XmPrimitive
XmNforeground	XmPrimitive	XmNunitType	XmPrimitive
XmNheight	Core	XmNuserData	XmPrimitive
XmNhelpCallback	XmPrimitive	XmNwidth	Core
XmNhighlightColor	XmPrimitive	XmNx	Core
XmNhighlightOnEnter	XmPrimitive	XmNy	Core
XmNhighlightPixmap	XmPrimitive		

Translations

The translations for Text include those from Primitive, as well as the following. (Note that some of the associated actions will be reversed for a language environment in which text is not read from left to right.)

Event	Action	Event	Action
BSelect Press	grab-focus()	MShift KPageDown	next-page(extend)
BSelect Motion	extend-adjust()	KPageLeft	page-left()
BSelect Release	extend-end()	KPageRight	page-right()
BExtend Press	extend-start()	KBeginLine	beginning-of-line()

Event	Action	Event	Action
BExtend Mition	extend-adjust()	MShift KBeginLine	beginning-of-line(extend)[a]
BExtend Release	extend-end()	KEndLine	end-of-line
BToggle Press	move-destination()	MShift KEndLine	end-of-line(extend)
BTransfer Press	process-bdrag() (1.2) secondary-start (1.1)	KBeginData	beginning-of-file()
BTransfer Motion	secondary-adjust()	MShift KBeginData	beginning-of-file(extend)
BTransfer Release	copy-to()	KEndData	end-of-file()
MCtl BTransfer Press	process-bdrag() (1.2) secondary-start (1.1)	MShift KEndData	end-of-file(extend)
MCtrl BTransfer Motion	secondary-adjust()	KTab	process-tab()
MCtrl BTransfer Release	copy-to()	KNextField	next-tab-group()
MAlt BTransfer Press	process-bdrag() (1.2) secondary-start (1.1)	KPrevField	prev-tab-group()
MAlt BTransfer Motion	secondary-adjust()	KEnter	process-return()
MAlt BTransfer Release	copy-to()	KActivate	activate()
MShift BTransfer Press	process-bdrag()	KDelete	delete-next-character()
MShift BTransfer Motion	secondary-adjust()	KBackSpace	delete-previous-character()
MShift BTransfer Release	move-to()	KAddMode	toggle-add-mode()
MAlt MCtrl BTransfer Release	copy-to()	KSpace	self-insert()
MAlt MShift BTransfer Release	move-to()	MShift KSpace	insert
KUp	process-up()	KSelect	set-anchor()
MShift KUp	process-shift-up()	KExtend	key-select()
MCtrl KUp	backward-paragraph()	MAny KCancel	process-cancel()
MShift MCtrl KUp	backward-paragraph(extend)	KClear	clear-selection()
KDown	process-down()	KSelectAll	select-all()
MShift KDown	process-shift-down()	KDeselectAll	deselect-all()
MCtrl KDown	forward-paragraph()	KCut	cut-clipboard()
MShift MCtrl KDown	forward-paragraph(extend)	KCopy	copy-clipboard()
KLeft	backward-character()	KPaste	paste-clipboard()
MShift KLeft	key-select(left)	KPrimaryCut	cut-primary()
MCtrl KLeft	backward-word()	KPrimaryCopy	copy-primary()
MShift MCtrl KLeft	backward-word(extend)	KPrimaryPaste	copy-primary()
KRight	forward-character()	KQuickCut	quick-cut-set() (1.1)
MShift KRight	key-select(right)	KQuickCopy	quick-copy-set() (1.1)
MCtrl KRight	forward-word()	KQuickPaste	quick-copy-set() (1.1)

Event	Action	Event	Action
MShift MCtrl KRight	formward-word(extend)	KQuickExtend	do-quick-action() (1.1)
KPageUp	previous-page()	KHelp	Help()
MShift KPageUp	previous-page(extend)	KAny	self-insert()
KPageDown	next-page()		

a.Erroneously given as beginning-of-file(extend) in 1st and 2nd editions.

Action Routines

Text defines the action routines below. For actions that involve movement such as next, previous, start, end, back, forward, etc., the actual cursor movement depends on whether the layout direction is left-to-right or right-to-left. In addition, some actions accept an optional argument, extend. When applied with no argument, these actions move the cursor; when applied with the extend argument, these actions move the cursor but also extend the text selection. In all descriptions, the term cursor refers to the insertion cursor.

activate()
: Invokes the callbacks specified by XmNactivateCallback.

backward-character()
: Moves the cursor back one character.

backward-paragraph(extend)
: Moves the cursor back to the first non-blank character that follows a blank line (or back to the start of the text if there is no previous blank line). If the cursor is already located at a non-blank character (i.e., if it's already at the beginning of the paragraph), the cursor moves to the start of the previous paragraph. (Multi-line edit mode only.)

backward-word(extend)
: Moves the cursor back to the first non-blank character that follows a blank character (or back to the start of the line if there is no previous blank character). If the cursor is already located at a non-blank character (i.e., if it's already at the beginning of a word), the cursor moves to the start of the previous word.

beep()
: Makes the terminal beep.

beginning-of-file(extend)
: Moves the cursor to the start of the text.

beginning-of-line(extend)
: Moves the cursor to the start of the line.

clear-selection()
: Replaces each character (except a newline) with a space, effectively clearing the current selection.

copy-clipboard()
: Copies the current text selection into the clipboard.

copy-primary()
: Inserts a copy of the primary selection at the cursor location.

copy-to()
: Inserts a copy of the secondary selection at the cursor location, or, if there is no secondary selection, inserts a copy of the primary selection at the pointer location.

cut-clipboard()
: Deletes the current selection and moves it to the clipboard.

cut-primary()
: Deletes the primary selection and inserts it at the cursor.

delete-next-character()
delete-previous-character()
: If the cursor is inside the selection and XmNpendingDelete is True, deletes the selection. Otherwise, deletes the character following/preceding the cursor.

delete-next-word()
delete-previous-word()
: If the cursor is inside the selection and XmNpendingDelete is True, deletes the selection. Otherwise, deletes from the character following/preceding the cursor to the next/previous space, tab, or end of line.

delete-selection()
: Deletes the current selection.

delete-to-end-of-line()
: Deletes forward from the character after the cursor up to and including the end of the line.

delete-to-start-of-line()
: Deletes back from the character before the cursor up to and including the beginning of the line.

deselect-all()
: Deselects the current selection.

do-quick-action(0
: In Motif 1.1, Ends a secondary selection and does the action that was started by either of the actions quick-copy-set or quick-cut-set.

end-of-file(extend)
: Moves the cursor to the end of the text.

end-of-line(extend)
: Moves the cursor to the end of the line.

extend-adjust()
: Selects text that is between the anchor and the pointer location, while deselecting text that is outside this area. As a result of this action, when the pointer moves past lines of text, these lines are selected and the current line is selected up to the position of the pointer.

extend-end()
: Moves the cursor to the pointer location and ends the selection performed by extend-adjust.

extend-start()
: Adjusts the anchor in preparation for selecting text via the extend-adjust action.

forward-character()
: Moves the cursor forward one character.

forward-paragraph(extend)
: Moves the cursor forward to the first non-blank character that follows a blank line. If the cursor is already located at a non-blank character (i.e., if it's already at the beginning of the paragraph), the cursor moves to the start of the next paragraph. (Multi-line edit mode only.)

forward-word(extend)
: Moves the cursor forward to the first blank character that follows a non-blank character (or forward to the end of the line if there is no blank character to move to). If the cursor is already located at a blank character (i.e., if it's already at the end of a word), the cursor moves to the end of the next word.

grab-focus()
: Processes multi-clicks as defined in the XmNselectionArray resource. By default, one click resets the cursor to the pointer location, two clicks select a word, three clicks select a line, and four clicks select all of the text.

Help()
: Invokes the list of callbacks specified by XmNhelpCallback. If the Text widget doesn't have any help callbacks, this action routine invokes those associated with the nearest ancestor that has them.

insert-string(text)
: Inserts text at the cursor, or replaces the current selection with text (when XmNpendingDelete is True).

key-select(direction)
: Extends the selection and moves the cursor one character to the right (when direction is right), one character to the left (direction is left). If no direction is specified, the selection is extended, although the insertion cursor is not moved.

kill-next-character()
kill-next-word()
kill-previous-character()
kill-previous-word()
: These four actions are similar to their delete action counterparts, but the kill actions have the added feature of storing the deleted text in the cut buffer.

kill-selection()
: Deletes the current selection and stores this text in the cut buffer.

kill-to-end-of-line()
: Deletes forward from the character after the cursor up to and including the end of the line; stores this text in the cut buffer.

kill-to-start-of-line()
: Deletes back from the character before the cursor up to and including the beginning of the line; stores this text in the cut buffer.

move-destination()
: Moves the cursor to the pointer location, leaving existing selections unaffected.

move-to()
: Deletes the secondary selection and inserts it at the cursor, or, if there is no secondary selection, deletes the primary selection and inserts it at the pointer location.

newline()
: If the cursor is inside the selection and XmNpendingDelete is True, deletes the selection and inserts a newline at the cursor. Otherwise, only inserts a newline at the cursor.

newline-and-backup()
: If the cursor is inside the selection and XmNpendingDelete is True, deletes the selection, inserts a newline at the cursor and moves the cursor to the end of the previous line. Otherwise, only inserts a newline and then moves the cursor to the end of the previous line.

newline-and-indent()
: If the cursor is inside the selection and XmNpendingDelete is True, deletes the selection, inserts a newline at the cursor, and adds blanks (as needed) so that the cursor aligns with the first non-blank character in the previous line. Otherwise, only inserts a newline and adds blanks (as needed) so that the cursor aligns with the first non-blank character in the previous line.

next-line()
: Places the cursor on the next line.

next-page(extend)
: Moves the cursor one page forward.

next-tab-group()
: Traverses to the next tab group.

page-left()
page-right()
: Scrolls the visible area one page to the left or right.

paste-clipboard()
: Pastes text from the clipboard to the position before the cursor.

prev-tab-group()
: Traverses to the previous tab group.

previous-line()
: Places the cursor on the previous line.

previous-page(extend)
: Moves the cursor one page backward.

process-bdrag()
: In Motif 1.2, copies the current selection to the insertion cursor if text is selected, the location cursor is outside of the selection, and no motion is detected. Performs a secondary selection and copies the selection to the position where text was last edited if the cursor is outside of the selection and motion is detected. Otherwise, initiates a drag and drop operation using the current selection.

process-cancel()
: Cancels the extend-adjust() or secondary-adjust() actions that are currently being applied, restoring the selection to its previous state.

process-down()
process-up()
: If XmNnavigationType is XmNONE, descends/ascends to the adjacent widget in the tab group (single-line edit mode only). Moves the cursor one line down/up (multi-line edit mode only).

process-home() Moves the cursor to the start of the line. (Similar to beginning-of-line.)

process-return() Invokes the XmNactivateCallback callbacks (in single-line editing) or inserts a newline (in multi-line editing).

process-shift-down()
process-shift-up()
: Moves the cursor one line down or up (in multi-line editing only).

process-tab()
: Traverses to the next tab group (in single-line editing) or inserts a tab (in multi-line editing).

quick-copy-set()
: In Motif 1.1, marks this text location as the start of the secondary selection to use in quick copying.

quick-cut-set()
: In Motif 1.1, marks this text location as the start of the secondary selection to use in quick cutting.

redraw-display()
: Redraws the text in the viewing window.

scroll-one-line-down()
scroll-one-line-up()
: Scrolls the text region one line down or up.

secondary-adjust()
: Extends the secondary selection to the location of the pointer.

secondary-notify()
: Inserts a copy of the secondary selection at the destination cursor.

secondary-start()
: In Motif 1.1, marks this text location as the start of a secondary selection.

select-adjust()
: Extends the selection via the multiple mouse clicks defined by the XmNselectionArray resource.

select-all()
: Selects all text.

select-end()
: Ends the selection made using the select-adjust() action.

select-start()
: Begins a text selection.

self-insert()
: The basic method of inserting text. Typing at the keyboard inserts new text and (if XmNpendingDelete is True) replaces selected text that the cursor is in.

set-anchor()
: Changes the anchor point used when making extended selections; changes the destination cursor used for secondary selections.

set-insertion-point()
: Sets the position of the cursor.

set-selection-hint()
: Sets the selection's text source and the selection's location.

toggle-add-mode()
: Turns Add Mode either on or off.

toggle-overstrike()
: Changes the text insertion mode. When a character is typed into a Text widget, by default it is inserted at the location of the insertion cursor. In overstrike mode, an inserted character replaces the current character that immediately follows the insertion cursor. When the insertion cursor is at the end of a line in overstrike mode, inserted characters are appended to the line.

traverse-home()
traverse-next()
traverse-prev()
: Traverse within the tab group to the first widget, the next widget, and the previous widget, respectively.

unkill()
: Restores the most recently deleted text to the cursor's location.

Additional Behavior

Text has the following additional behavior:

<FocusIn>
: Draws a solid insertion cursor and makes it blink.

<FocusOut>
: Draws a stippled I-beam insertion cursor, unless the widget is the destination of a data transfer.

See Also

XmCreateObject(1), XmTextClearSelection(1), XmTextCopy(1), XmTextCut(1), XmTextDisableRedisplay(1), XmTextEnableRedisplay(1), XmTextFindString(1), XmTextFindStringWcs(1), XmTextGetBaseline(1), XmTextGetCursorPosition(1), XmTextGetEditable(1), XmTextGetInsertionPosition(1), XmTextGetLastPosition(1), XmTextGetMaxLength(1), XmTextGetSelection(1), XmTextGetSelectionPosition(1), XmTextGetSelectionWcs(1), XmTextGetSource(1), XmTextGetString(1), XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextGetTopCharacter(1), XmTextInsert(1), XmTextInsertWcs(1), XmTextPaste(1), XmTextPosToXY(1), XmTextRemove(1), XmTextReplace(1), XmTextReplaceWcs(1), XmTextScroll(1), XmTextSetAddMode(1), XmTextSetCursorPosition(1), XmTextSetEditable(1), XmTextSetHighlight(1), XmTextSetInsertionPosition(1), XmTextSetMaxLength(1), XmTextSetSelection(1), XmTextSetSource(1), XmTextSetString(1), XmTextSetStringWcs(1), XmTextSetTopCharacter(1), XmTextShowPosition(1), XmTextXYToPos(1), XmTransfer(1), Core(2), XmRendition(2), XmPrimitive(2), XmPrinShell(2), XmTextField(2).

Name

XmTextField widget class – a single-line text-editing widget.

Synopsis

Public Header:
<Xm/TextF.h>

Class Name:
XmTextField

Class Hierarchy:
Core →XmPrimitive →XmTextField

Class Pointer:
xmTextFieldWidgetClass

Instantiation:
widget = `XmCreateTextField` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmTextWidgetClass,...)

Functions/Macros:
`XmCreateTextField`(), `XmIsTextField`(), XmTextField... routines

Description

A TextField widget provides a single-line text editor that has a subset of the functionality of the Text widget.

Traits

TextField holds the XmQTaccessTextual and XmQTtransfer traits, which are inherited in any derived classes, and uses the XmQTaccessTextual and XmQTspecifyRenderTable traits.

New Resources

TextField defines the following resources:

Name	Class	Type	Default	Access
XmNblinkRate	XmCBlinkRate	int	500	CSG
XmNcolumns	XmCColumns	short	dynamic	CSG
XmNcursorPosition	XmCCursorPosition	XmTextPosition	0	CSG
XmNcursorPositionVisible	XmCCursorPositionVisible	Boolean	dynamic	CSG
XmNeditable	XmCEditable	Boolean	True	CSG
XmNfontList	XmCFontList	XmFontList	dynamic	CSG
XmNmarginHeight	XmCMarginHeight	Dimension	5	CSG
XmNmarginWidth	XmCMarginWidth	Dimension	5	CSG

Name	Class	Type	Default	Access
XmNmaxLength	XmCMaxLength	int	largest integer	CSG
XmNpendingDelete	XmCPendingDelete	Boolean	True	CSG
XmNrenderTable	XmCRenderTable	XmRenderTable	dynamic	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
XmNvalueWcs	XmCValueWcs	wchar_t	(Wchar_t *) ""	CSG
XmNverifyBell	XmCVerifyBell	Boolean	dynamic	CSG

XmNblinkRate
: The time in milliseconds that the cursor spends either being visible or invisible. A value of 0 prevents the cursor from blinking.

XmNcolumns
: The number of character spaces that should fit horizontally in the text window. The XmNwidth resource determines the default value of XmNcolumns, but if no width has been set, the default is 20.

XmNcursorPosition
: The location at which to place the current insert cursor. Values for this resource are relative to the beginning of the text, with the first character position defined as 0.

XmNcursorPositionVisible
: If True, the text cursor will be visible. In Motif 2.1, if the text field has an XmPrintShell as ancestor, the default is False. Otherwise the default is True.

XmNeditable
: If True (default), the user is allowed to edit the text string; if False, the user is not allowed to do so.

XmNfontList
: The font list used for the widget's text. In Motif 2.0 and later, the XmFontList is obsolete, and is subsumed into the XmRenderTable. If both a render table and font list are specified, the render table takes precedence.

XmNmarginHeight

XmNmarginWidth
: The spacing between the edges of the widget and the text. (Top and bottom edges for height; left and right for width.)

XmNmaxLength
: The maximum length of the text string that a user can enter from the keyboard. This resource doesn't affect strings that are entered via the XmNvalue resource or the `XmTextFieldSetString`() routine.

XmNpendingDelete
: If True (default), the TextField widget's pending delete mode is on, meaning that selected text will be deleted as soon as the next text insertion occurs.

XmNrenderTable
: In Motif 2.0 and later, specifies the render table for the TextField. If unspecified, the value of the resource is inherited from the nearest ancestor which holds the XmQTspecifyRenderTable trait, using the XmTEXT_RENDER_TABLE value of the ancestor so found.

XmNresizeWidth
: If False (default), the TextField widget will not expand horizontally to fit its text. If True, the widget tries to change its width.

XmNselectionArray
: The array of possible actions caused by multiple mouse clicks. UIL does not define these values for the Text widget. Possible values:

```
XmSELECT_POSITION     /* single-click; reset position of insert cursor   */
XmSELECT_WORD         /* double-click; select a word                      */
XmSELECT_LINE         /* triple-click; select a line                      */
```

XmNselectionArrayCount
: The number of items in the array specified by XmNselectionArray.

XmNselectThreshold
: The number of pixels the insertion cursor must be dragged during selection in order to select the next character.

XmNvalue
: The string value to display in the TextField widget, expressed as a char *. If XmNvalue and XmNvalueWcs are both defined, XmNvalueWcs takes precedence. Use XtSetValues() to copy string values to the internal buffer and use XtGetValues() to return the value of the internal buffer.

XmNvalueWcs
: In Motif 1.2 and later, the string value to display in the TextField widget, expressed as a wchar_t *. If XmNvalue and XmNvalueWcs are both defined, XmNvalueWcs takes precedence. Use XtSetValues() to copy string values to the internal buffer and use XtGetValues() to return the value of the internal buffer. This resource cannot be set in a resource file.

XmNverifyBell
: If True, a bell will sound when a verification produces no action.

Callback Resources

TextField defines the same callback resources and references the same callback structures as a single line Text widget.

Inherited Resources

TextField inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. TextField sets the default value of XmNnavigationType to XmTAB_GROUP. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNaccelerators	Core	XmNhighlightThickness	XmPrimitive
XmNancestorSensitive	Core	XmNinitialResourcesPersistent	Core
XmNbackground	Core	XmNlayoutDirection	XmPrimitive
XmNbackgroundPixmap	Core	XmNmappedWhenManaged	Core
XmNborderColor	Core	XmNnavigationType	XmPrimitive
XmNborderPixmap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNborderWidth	Core	XmNscreen	Core
XmNbottomShadowColor	XmPrimitive	XmNsensitive	Core
XmNbottomShadowPixmap	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNcolormap	Core	XmNtopShadowColor	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNdepth	Core	XmNtranslations	Core
XmNdestroyCallback	Core	XmNtraversalOn	XmPrimitive
XmNforeground	XmPrimitive	XmNunitType	XmPrimitive
XmNheight	Core	XmNuserData	XmPrimitive
XmNhelpCallback	XmPrimitive	XmNwidth	Core
XmNhighlightColor	XmPrimitive	XmNx	Core
XmNhighlightOnEnter	XmPrimitive	XmNy	Core
XmNhighlightPixmap	XmPrimitive		

Translations

TextField has the same translation as a Text widget whose XmNeditMode resource is set to XmSINGLE_LINE_EDIT.

Action Routines

TextField defines the same action routines as a Text widget whose XmNeditMode resource is set to XmSINGLE_LINE_EDIT.

See Also

XmCreate*Object*(1), XmTextClearSelection(1), XmTextCopy(1), XmTextCut(1), XmTextGetBaseline(1), XmTextGetCursorPosition(1), XmTextGetEditable(1), XmTextGetInsertionPosition(1), XmTextGetLastPosition(1), XmTextGetMaxLength(1), XmTextGetSelection(1), XmTextGetSelectionPosition(1), XmTextGetSelectionWcs(1), XmTextGetString(1), XmTextGetStringWcs(1), XmTextGetSubstring(1), XmTextGetSubstringWcs(1), XmTextInsert(1), XmTextInsertWcs(1), XmTextPaste(1), XmTextPosToXY(1), XmTextRemove(1), XmTextReplace(1), XmTextReplaceWcs(1), XmTextScroll(1), XmTextSetAddMode(1), XmTextSetCursorPosition(1), XmTextSetEditable(1), XmTextSetHighlight(1), XmTextSetInsertionPosition(1), XmTextSetMaxLength(1), XmTextSetSelection(1), XmTextSetSource(1), XmTextSetString(1), XmTextSetStringWcs(1), XmTextSetTopCharacter(1), XmTextShowPosition(1), XmTextXYToPos(1), Core(2), XmPrimitive(2), XmPrintShell(2), XmRendition(2), XmText(2).

Name

XmToggleButton widget class – a button widget that maintains a Boolean state.

Synopsis

Public Header:
<Xm/ToggleB.h>

Class Name:
XmToggleButton

Class Hierarchy:
Core →XmPrimitive →XmLabel →XmToggleButton

Class Pointer:
xmToggleButtonWidgetClass

Instantiation:
widget = `XmCreateToggleButton` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmToggleButtonWidgetClass,...)

Functions/Macros:
`XmCreateToggleButton`(), `XmToggleButtonGetState`(), `XmToggleButtonSetState`(), `XmToggleButtonSetValue`(), `XmIsToggleButton`()

Description

A ToggleButton is a button that is either set or unset. ToggleButtons are typically used in groups, called RadioBoxes and CheckBoxes, depending on the behavior of the buttons. In a RadioBox, a ToggleButton displays *one-of-many* behavior, which means that only one button in the group can be set at a time. When a button is selected, the previously selected button is unset. In a CheckBox, a ToggleButton displays *n-of-many* behavior, which means that any number of ToggleButtons can be set at one time. ToggleButton uses an indicator to show its state; the shape of the indicator specifies the type of behavior. A diamond-shaped indicator is used for one-of-many ToggleButtons and a square-shaped indicator is used for n-of-many ToggleButtons.

In Motif 2.0 and later, a ToggleButton can have three possible states: set, unset, and indeterminate, depending upon the value of the XmNtoggleMode resource. If the value is XmTOGGLE_BOOLEAN, the ToggleButton has the two states, set and unset. If the value is XmTOGGLE_INDETERMINATE, the ToggleButton has three states.

Traits

ToggleButton uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

New Resources

ToggleButton defines the following resources, all with CSG access:

Name	Class	Type	Default
XmNdetailShadowThickness	XmCDetailShadowThickness	Dimension	dynamic
XmNfillOnSelect	XmCFillOnSelect	Boolean	dynamic
XmNindeterminate InsensitivePixmap	XmCIndeterminate InsensitivePixmap	Pixmap	XmUNSPECIFIED_PIXMAP
XmNindeterminatePixmap	XmCIndeterminatePixmap	Pixmap	XmUNSPECIFIED_PIXMAP
XmNindicatorOn	XmCIndicatorOn	unsigned char	XmINDICATOR_FILL
XmNindicatorSize	XmCIndicatorSize	Dimension	dynamic
XmNindicatorType	XmCIndicatorType	unsigned char	dynamic
XmNselectColor	XmCSelectColor	Pixel	dynamic
XmNselectInsensitivePixmap	XmCSelectInsensitivePixmap	Pixmap	XmUNSPECIFIED_PIXMAP
XmNselectPixmap	XmCSelectPixmap	Pixmap	XmUNSPECIFIED_PIXMAP
XmNset	XmCSet	unsigned char	XmUNSET
XmNspacing	XmCSpacing	Dimension	4
XmNtoggleMode	XmCToggleMode	unsigned char	XmTOGGLE_BOOLEAN
XmNunselectColor	XmCUnselectColor	Pixel	dynamic
XmNvisibleWhenOff	XmCVisibleWhenOff	Boolean	dynamic

XmNdetailShadowThickness

In Motif 2.0 and later, specifies the thickness of the shadow on the indicator. In Motif 2.0 the default value is 2. In Motif 2.1 and later, the default value depends upon the XmDisplay XmNenableThinThickness resource: if True the default is 1, otherwise 2.

XmNfillOnSelect

If True, selection of this ToggleButton fills the indicator with the color given by the XmNselectColor resource and switches the button's top and bottom shadow colors.

If the ToggleButton is unselected, the top and bottom shadow colors are switched. In Motif 2.0 and later, the indicator is filled with the color given by the XmNunselectColor resource.

If the ToggleButton is in the indeterminate state as specified by the XmNset resource, the indicator is half filled with the XmNselectColor and half with the XmNunselectColor values.

If fill on select is False, only the top and bottom shadow colors are switched.

When XmNindicatorOn is XmINDICATOR_NONE, XmNfillOnSelect is True, and XmNset is XmSET, the background of the entire button is filled with the XmNselectColor.

In Motif 1.2 and earlier, the default value is set to the value of XmNindicatorOn. In Motif 2.0 and later, the default depends upon both XmNindicatorOn and XmNindicatorType resources.

XmNindeterminateInsensitivePixmap
Specifies the pixmap to use if XmNset is XmINDETERMINATE and the Toggle is insensitive. The resource has no effect if the inherited Label resource XmNlabelType is XmSTRING.

XmNindeterminatePixmap
Specifies the pixmap to use if XmNset is XmINDETERMINATE and the Toggle is sensitive. The resource has no effect if the inherited Label resource XmNlabelType is XmSTRING.

XmNindicatorOn
In Motif 1.2 and earlier, the resource is a Boolean value. If True (default), the indicator is visible and its shadows are switched when the button is toggled. If False, the indicator is invisible and no space is set aside for it; in addition, the shadows surrounding the button are switched when it is toggled.
In Motif 2.0 and later, the resource is an enumerated type, and it specifies the type of indicator required. Possible values:

```
XmINDICATOR_NONE        /* no indicator                */
XmINDICATOR_FILL        /* check box or box            */
XmINDICATOR_BOX         /* shadowed box, in Motif 2.1  */
XmINDICATOR_CHECK       /* checkmark                   */
XmINDICATOR_CHECK_BOX   /* checkmark enclosed in a box */
XmINDICATOR_CROSS       /* cross                       */
XmINDICATOR_CROSS_BOX   /* cross enclosed in a box     */
```

If the value of the XmDisplay object's XmNenableToggleVisual resource is True, XmINDICATOR_FILL is equivalent to XmINDICATOR_CHECK_BOX, otherwise XmINDICATOR_BOX.

XmNindicatorSize
The size of the indicator. This value changes if the size of the button's text string or pixmap changes.

XmNindicatorType

Determines whether the indicator is drawn as a diamond (signifying a one-of-many indicator) or as a square (signifying an n-of-many indicator). Possible values:

```
XmN_OF_MANY                 /* creates a square button                    */
XmONE_OF_MANY       /* creates either round- or diamond-shaped button */
XmONE_OF_MANY_ROUND  /* creates a round-shaped button  (2.0)       */
XmONE_OF_MANY_DIAMOND /* creates a diamond-shaped button (2.1)  */
```

In Motif 2.0, the value XmONE_OF_MANY is diamond shaped. In Motif 2.1, XmONE_OF_MANY produces either a diamond or a round shape, depending upon the value of the XmDisplay XmNenableToggleVisual resource. If this is True, the shape is round. The default value is XmONE_OF_MANY for a ToggleButton in a RadioBox widget, and XmN_OF_MANY otherwise. This resource only sets the indicator; it is RowColumn's XmNradioBehavior resource that actually enforces radioButton or checkButton behavior.

XmNselectColor

The color with which to fill the indicator when the button is selected. On a color display, the default is a value between the background color and the bottom shadow color; on a monochrome display, the default is the foreground color.

In Motif 2.0 and later, the following Pixel values are pre-defined for special meaning:

```
XmDEFAULT_SELECT_COLOR     /* a color between the background      */
                           /*  and bottom shadow                  */
XmREVERSED_GROUND_COLORS    /* select is foreground,              */
                            /* text drawn in background           */
XmHIGHLIGHT_COLOR      /* select color same as highlight color   */
```

XmNselectInsensitivePixmap

The pixmap used for an insensitive ToggleButton when it's selected. An unselected, insensitive ToggleButton uses the pixmap specified by the Label resource XmNlabelInsensitivePixmap. However, if this Label resource wasn't specified, it is set to the value of XmNselectInsensitivePixmap. This resource is meaningful only when the Label resource XmNlabelType is set to XmPIXMAP.

XmNselectPixmap

The pixmap used for a (sensitive) ToggleButton when it's selected. An unselected ToggleButton uses the pixmap specified by the Label resource XmNlabelPixmap. This resource is meaningful only when the Label resource XmNlabelType is set to XmPIXMAP.

XmNset
The selection state of the button. In Motif 1.2 and earlier, a simple Boolean value. In Motif 2.0 and later, a Toggle can be in three states: on, off, and indeterminate, and this resource is changed to an enumerated type. Possible values:

XmUNSET
XmSET
XmINDETERMINATE

If XmNtoggleMode is XmTOGGLE_INDETERMINATE, the Toggle cycles between XmSET, XmINDETERMINATE, XmUNSET, and then back to XmSET when pressed. If toggle mode is XmTOGGLE_BOOLEAN, the widget simply cycles between XmSET and XmUNSET.

Note that not all versions of Motif 2.1 allow the enumerated values to be specified in an external resource file. Version 2.1.30 sources have the problem fixed.

XmNspacing
The distance between the Toggle indicator and its label.

XmNtoggleMode
In Motif 2.0 and later, specifies whether the Toggle has two or three states. Possible values:

```
XmTOGGLE_BOOLEAN        /* two states   */
XmTOGGLE_INDETERMINATE  /* three states */
```

XmNunselectColor
In Motif 2.0 and later, specifies a color for filling the indicator shape. The resource behaves similarly to XmNselectColor, except that it is effective when XmNset is XmUNSET.

XmNvisibleWhenOff
If True, the Toggle indicator remains visible when the button is unselected. This is the default behavior in a RadioBox. The default is False in a menu.

Callback Resources

ToggleButton defines the following callback resources:

Callback	Reason Constant
XmNarmCallback	XmCR_ARM
XmNdisarmCallback	XmCR_DISARM
XmNvalueChangedCallback	XmCR_ACTIVATE

XmNarmCallback
List of callbacks that are called when BSelect is pressed while the pointer is inside the widget.

XmNdisarmCallback
List of callbacks that are called when BSelect is released after it has been pressed inside the widget.

XmNvalueChangedCallback
List of callbacks that are called when the value of the ToggleButton is changed.

Callback Structure

Each callback function is passed the following structure:

```
typedef struct {
    int         reason;         /* the reason that the callback was called */
    XEvent      *event;         /* points to event that triggered callback */
    int         set;            /* the state of the button                 */
} XmToggleButtonCallbackStruct;
```

set indicates the state of the Toggle, and is one of XmSET, XmUNSET, or XmINDETERMINATE.

Inherited Resources

ToggleButton inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. ToggleButton sets the default values of XmNmarginBottom, XmNmarginTop, XmNmarginWidth, and XmNshadowThickness dynamically. The default value of XmNborderWidth is reset to 0 by Primitive. In Motif 2.0 and earlier, the default value of XmNhighlightThickness is reset to 2. In Motif 2.1 and later, the default value depends upon the XmDisplay XmNenableThinThickness resource: if True the default is 1, otherwise 2.

Resource	Inherited From	Resource	Inherited From
XmNaccelerator	XmLabel	XmNlabelType	XmLabel
XmNaccelerators	Core	XmNlayoutDirection	XmPrimitive
XmNacceleratorText	XmLabel	XmNmappedWhenManaged	Core
XmNalignment	XmLabel	XmNmarginBottom	XmLabel
XmNancestorSensitive	Core	XmNmarginHeight	XmLabel
XmNbackground	Core	XmNmarginLeft	XmLabel
XmNbackgroundPixmap	Core	XmNmarginRight	XmLabel
XmNborderColor	Core	XmNmarginTop	XmLabel
XmNborderPixmap	Core	XmNmarginWidth	XmLabel
XmNborderWidth	Core	XmNmnemonicCharSet	XmLabel
XmNbottomShadowColor	XmPrimitive	XmNmnemonic	XmLabel
XmNbottomShadowPixmap	XmPrimitive	XmNnavigationType	XmPrimitive

Resource	Inherited From	Resource	Inherited From
XmNcolormap	Core	XmNpopupHandlerCallback	XmPrimitive
XmNconvertCallback	XmPrimitive	XmNrecomputeSize	XmLabel
XmNdepth	Core	XmNrenderTable	XmLabel
XmNdestroyCallback	Core	XmNscreen	Core
XmNfontList	XmLabel	XmNsensitive	Core
XmNforeground	XmPrimitive	XmNshadowThickness	XmPrimitive
XmNheight	Core	XmNstringDirection	XmLabel
XmNhelpCallback	XmPrimitive	XmNtopShadowColor	XmPrimitive
XmNhighlightColor	XmPrimitive	XmNtopShadowPixmap	XmPrimitive
XmNhighlightOnEnter	XmPrimitive	XmNtranslations	Core
XmNhighlightPixmap	XmPrimitive	XmNtraversalOn	XmPrimitive
XmNhighlightThickness	XmPrimitive	XmNunitType	XmPrimitive
XmNinitialResourcesPersistent	Core	XmNuserData	XmPrimitive
XmNlabelInsensitivePixmap	XmLabel	XmNwidth	Core
XmNlabelPixmap	XmLabel	XmNx	Core
XmNlabelString	XmLabel	XmNy	Core

Translations

The translations for ToggleButton include those from Primitive. In addition, ToggleButtons that are not in a menu system have the following translations:

Event	Action
BTransfer Press	ProcessDrag()
BSelect Press	Arm()
MCtrl BSelect Press	ButtonTakeFocus()
BSelect Release	Select() Disarm()
KHelp	Help()
KSelect	ArmAndActivate()

For ToggleButtons that are in a menu system, translations include the menu traversal translations inherited from the Label widget, as well as the following:

Event	Action
MCtrl BSelect Press	MenuButtonTakeFocus()
BSelect Press	BtnDown()

Event	Action
BSelect Release	BtnUp()
KHelp	Help()
KActivate	ArmAndActivate()
KSelect	ArmAndActivate()
MAny KCancel	MenuShellPopdownOne()

Action Routines

ToggleButton defines the following action routines:

Arm()

Sets the button if it was previously unset, unsets the button if it was previously set, and invokes the callbacks specified by XmNarmCallback. Setting the button means displaying it so that it appears selected. The selected state can be shown by: Highlighting the indicator so it appears pressed in. Filling in the indicator (using the color given by XmNselectColor). Highlighting the button so it appears pressed in. (This is done only if the indicator isn't displayed). Drawing the button face using the pixmap given by XmNselectPixmap.

The unselected state can be shown by: Highlighting the indicator so it appears raised. Filling in the indicator with the background color. Highlighting the button so it appears raised. (This is done only if the indicator isn't displayed). Drawing the button face using the pixmap given by XmNlabelPixmap.

ArmAndActivate()

Sets the button if it was previously unset, unsets the button if it was previously set, and invokes the callbacks specified by XmNarmCallback (if the button isn't yet armed), XmNvalueChangedCallback, and XmNdisarmCallback. Inside a menu, this action unposts the menu hierarchy. Outside a menu, this action displays the button as selected or unselected, as described for Arm().

BtnDown()

Unposts any menus that were posted by the parent menu of the ToggleButton, changes from keyboard traversal to mouse traversal, draws a shadow to show the ToggleButton as armed, and (assuming the button is not yet armed) invokes the callbacks specified by XmNarmCallback.

BtnUp()
: Unposts the menu hierarchy, changes the ToggleButton's state, and invokes first the callbacks specified by XmNvalueChangedCallback and then those specified by XmNdisarmCallback.

ButtonTakeFocus()
: In Motif 2.0 and later, moves the current keyboard focus to the ToggleButton, without activating the widget.

Disarm()
: Invokes the callbacks specified by XmNdisarmCallback.

Help()
: Unposts the menu hierarchy, restores the previous keyboard focus, and invokes the callbacks specified by the XmNhelpCallback resource.

MenuButtonTakeFocus()
: In Motif 2.0 and later, moves the current keyboard focus to the ToggleButton, without activating the widget.

MenuShellPopdownOne()
: Unposts the current menu and (unless the menu is a pulldown submenu) restores keyboard focus to the tab group or widget that previously had it. In a top-level pulldown menu pane attached to a menu bar, this action routine also disarms the cascade button and the menu bar.

ProcessDrag()
: In Motif 1.2, initiates a drag and drop operation using the label of the ToggleButton.

Select()
: Switches the state of the ToggleButton and invokes the callbacks specified by the resource XmNvalueChangedCallback.

Additional Behavior

ToggleButton has the following additional behavior:

<EnterWindow>
: Displays the ToggleButton as armed.

<LeaveWindow>
: Displays the ToggleButton as unarmed.

See Also

XmCreateObject (1), XmToggleButtonGetState (1), XmToggleButtonSetState (1), XmToggleButtonSetValue(1), XmToggleButtonGetValue(1),Core (2), XmPrimitive (2), XmLabel (2), XmCheckBox (2), XmRadioBox (2), XmRowColumn (2).

Name

XmToggleButtonGadget widget class – a button gadget that maintains a Boolean state.

Synopsis

Public Header:
<Xm/ToggleBG.h>

Class Name:
XmToggleButtonGadget

Class Hierarchy:
Object →RectObj →XmGadget →XmLabelGadget →XmToggleButtonGadget

Class Pointer:
xmToggleButtonGadgetClass

Instantiation:
widget = `XmCreateToggleButtonGadget` (parent, name,...)
or
widget = `XtCreateWidget` (name, xmToggleButtonGadgetClass,...)

Functions/Macros:
`XmCreateToggleButtonGadget`(), `XmToggleButtonGadgetGetState`(), `XmToggleButtonGadgetSetState`(), `XmToggleButtonGadgetSetValue`(), `XmIsToggleButtonGadget`()

Description

ToggleButtonGadget is the gadget variant of ToggleButton.

ToggleButtonGadget's new resources, callback resources, and callback structure are the same as those for ToggleButton.

Traits

ToggleButtonGadget holds the XmQTcareParentVisual trait, which is inherited in any derived classes, and clones the XmQTmenuSavvy trait from the LabelGadget class. In addition, the widget uses the XmQTmenuSystem and XmQTspecifyRenderTable traits.

Inherited Resources

ToggleButtonGadget inherits the following resources. The resources are listed alphabetically, along with the superclass that defines them. ToggleButtonGadget sets the default values of XmNmarginBottom[1], XmNmarginTop, XmNmarginWidth, and XmNshadowThickness dynamically. The default value of XmNborderWidth is reset to 0 by Primitive.

Resource	Inherited From	Resource	Inherited From
XmNancestorSensitive	RectObj	XmNhighlightThickness	XmGadget
XmNbackground	XmGadget	XmNlayoutDirection	XmGadget
XmNbackgroundPixmap	XmGadget	XmNnavigationType	XmGadget
XmNbottomShadowColor	XmGadget	XmNsensitive	RectObj
XmNbottomShadowPixmap	XmGadget	XmNshadowThickness	XmGadget
XmNborderWidth	RectObj	XmNtopShadowColor	XmGadget
XmNdestroyCallback	Object	XmNtopShadowPixmap	XmGadget
XmNforeground	XmGadget	XmNtraversalOn	XmGadget
XmNheight	RectObj	XmNunitType	XmGadget
XmNhelpCallback	XmGadget	XmNuserData	XmGadget
XmNhighlightColor	XmGadget	XmNwidth	RectObj
XmNhighlightOnEnter	XmGadget	XmNx	RectObj
XmNhighlightPixmap	XmGadget	XmNy	RectObj

Behavior

As a gadget subclass, ToggleButtonGadget has no translations associated with it. However, ToggleButtonGadget behavior corresponds to the action routines of the ToggleButton widget. See the ToggleButton action routines for more information.

Event	Action
BTransfer Press	ProcessDrag()
BSelect Press	Arm() BtnDown() in a menu
BSelect Release	Select(), Disarm() BtnUp() in a menu
KActivate	ArmAndActivate()
KSelect	ArmAndActivate()

1.Erroneously given as XmNmarginBotton in 1st and 2nd editions.

Event	Action
KHelp	Help()
MAny KCancel	MenuShellPopdownOne()
MCtrl BSelect Press	ButtonTakeFocus() MenuButtonTakeFocus() in a menu

ToggleButtonGadget has additional behavior associated with <Enter> and <Leave>, which draw the shadow in the armed or unarmed state, respectively.

See Also

`XmCreateObject` (1), `XmToggleButtonGetState` (1), `XmToggleButtonSetState` (1), XmToggleButtonGadgetSetValue(1), XmToggleButtonGadgetSetValue(1), `Object` (2), `RectObj` (2), `XmCheckBox` (2), `XmGadget` (2), `XmLabelGadget` (2), `XmRadioBox` (2), `XmRowColumn` (2), `XmToggleButton` (2).

Name

XmWarningDialog – an unmanaged MessageBox as a child of a DialogShell.

Synopsis

Public Header:
<Xm/MessageB.h>

Instantiation:
widget = `XmCreateWarningDialog` (parent, name,...)

Functions/Macros:
`XmCreateWarningDialog`(), `XmMessageBoxGetChild`()

Description

An XmWarningDialog is a compound object created by a call to `XmCreateWarningDialog`() that an application can use to warn the user about a potentially hazardous action. A WarningDialog consists of a DialogShell with an unmanaged MessageBox widget as its child. The MessageBox resource XmNdialogType is set to XmDIALOG_WARNING.

A WarningDialog includes four components: a symbol, a message, three buttons, and a separator between the message and the buttons. By default, the symbol is an exclamation point. In Motif 1.2, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Cancel**, and **Help** by default.

Default Resource Values

A WarningDialog sets the following default values for MessageBox resources:

Name	Default
XmNdialogType	XmDIALOG_WARNING
XmNsymbolPixmap	xm_warning

Widget Hierarchy

When a WarningDialog is created with a specified name, the DialogShell is named name*_popup* and the MessageBox is called *name*.

See Also

`XmCreateObject` (1), `XmMessageBoxGetChild` (1), `XmDialogShell` (2), `XmMessageBox` (2).

Name

XmWorkingDialog – an unmanaged MessageBox as a child of a DialogShell.

Synopsis

Public Header:
<Xm/MessageB.h>

Instantiation:
widget = `XmCreateWorkingDialog` (parent, name,...)

Functions/Macros:
`XmCreateWorkingDialog`(), `XmMessageBoxGetChild`()

Description

An XmWorkingDialog is a compound object created by a call to `XmCreate-WorkingDialog`() that an application can use to warn the user that the current action is in progress, and likely to take some time. A WorkingDialog consists of a DialogShell with an unmanaged MessageBox widget as its child. The MessageBox resource XmNdialogType is set to XmDIALOG_WORKING.[1]

A WorkingDialog includes four components: a symbol, a message, three buttons, and a separator between the message and the buttons. By default, the symbol is an hourglass. In Motif 1.2, the default button labels can be localized. In the C locale, and in Motif 1.1, the PushButtons are labelled **OK**, **Cancel**, and **Help** by default.

Default Resource Values

A WorkingDialog sets the following default values for MessageBox resources:

Name	Default
XmNdialogType	XmDIALOG_WORKING
XmNsymbolPixmap	xm_working

Widget Hierarchy

When a WorkingDialog is created with a specified name, the DialogShell is named name*_popup* and the MessageBox is called *name*.

See Also

`XmCreateObject` (1), `XmMessageBoxGetChild` (1), `XmDialogShell` (2), `XmMessageBox` (2).

1.In the 1st and 2nd editions, this paragraph erroneously duplicated that of the XmWarningDialog description.

Section 3 - Mrm Functions

This page describes the format and contents of each reference page in Section 3, which covers the Motif Resource Manager (Mrm) functions.

Name

Function – a brief description of the function.

Synopsis

This section shows the signature of the function: the names and types of the arguments, and the type of the return value. The header file *<Mrm/MrmPublic.h>* declares all of the public Mrm functions.

Inputs

This subsection describes each of the function arguments that pass information to the function.

Outputs

This subsection describes any of the function arguments that are used to return information from the function. These arguments are always of some pointer type, so you should use the C address-of operator (**&**) to pass the address of the variable in which the function will store the return value. The names of these arguments are sometimes suffixed with *_return* to indicate that values are returned in them. Some arguments both supply and return a value; they will be listed in this section and in the "Inputs" section above. Finally, note that because the list of function arguments is broken into "Input" and "Output" sections, they do not always appear in the same order that they are passed to the function. See the function signature for the actual calling order.

Returns

This subsection explains the return values of the function. Mrm functions typically return one of the following values: MrmSUCCESS, MrmPARTIAL_SUCCESS, MrmBAD_HIERARCHY, MrmNOT_FOUND, MrmWRONG_TYPE, MrmNOT_VALID, MrmDISPLAY_NOT_OPENED, or MrmFAILURE. To be safe, you should check the return value against MrmSUCCESS or MrmPARTIAL_SUCCESS, and then check for specific errors on non-success. When an error occurs, the functions call `XtWarning`() with a descriptive error message.

Availability

This section appears for functions that were added in Motif 2.0 or later, and also for functions that are now superseded by other, preferred, functions.

Description

This section explains what the function does and describes its arguments and return value. If you've used the function before and are just looking for a refresher, this section and the synopsis above should be all you need.

Usage

This section appears for most functions and provides less formal information about the function: when and how you might want to use it, things to watch out for, and related functions that you might want to consider.

Example

This section provides an example of the use of the function. It also shows the corresponding UIL code needed for the example.

Structures

This section shows the definition of any structures, enumerated types, typedefs, or symbolic constants used by the function.

Procedures

This section shows the syntax of any prototype procedures used by the function.

See Also

This section refers you to related functions, UIL file format sections, and UIL data types. The numbers in parentheses following each reference refer to the sections of this book in which they are found.

Name

MrmCloseHierarchy – close an Mrm hierarchy.

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmCloseHierarchy (MrmHierarchy *hierarchy*)

Inputs

hierarchy Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().

Returns

MrmSUCCESS	On success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmFAILURE	On failure.

Description

`MrmCloseHierarchy`() closes an Mrm hierarchy that has been previously opened with a call to `MrmOpenHierarchy`() or `MrmOpenHierarchyPerDisplay`(). The UID files associated with the *hierarchy* are closed and the memory used by the *hierarchy* is freed. However, as of Motif 1.2, the memory used by Mrm to register any values or procedures with `MrmRegisterNamesInHierarchy`() is not freed.

Usage

An application calls `MrmCloseHierarchy`() when it is done accessing an Mrm hierarchy in order to free file descriptions and memory consumed by the hierarchy. As of Motif 1.2, this function cannot fail; it always returns MrmSUCCESS or MrmBAD_HIERARCHY.

Example

The following code fragment illustrates the use of `MrmCloseHierarchy`():

```
...
extern MrmHierarchy hierarchy; /* Previously opened Mrm hierarchy. */

if (MrmCloseHierarchy (hierarchy) != MrmSUCCESS)
    error_handler();
hierarchy = NULL;           /* Protect from future misuse. */
...
```

See Also

`MrmOpenHierarchy`(3), `MrmOpenHierarchyPerDisplay`(3).

Name

MrmFetchBitmapLiteral – retrieve an exported bitmap from an Mrm hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchBitmapLiteral[1] (MrmHierarchy hierarchy,
                                   String       name,
                                   Screen       *screen,
                                   Display      *display,
                                   Pixmap       *pixmap,
                                   Dimension    *width,
                                   Dimension    *height)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
name	Specifies the name of an icon to retrieve as a bitmap.
screen	Specifies the screen of the display on which the pixmap is created.
display	Specifies the display on which the pixmap is created.

Outputs

pixmap	Returns the specified bitmap as a pixmap of depth 1 on the specified screen of the specified display.
width	Returns the width of the pixmap.
height	Returns the height of the pixmap.

Returns

MrmSUCCESS	On success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmNOT_FOUND	If the icon is not found.
MrmWRONG_TYPE	If the named value is not an icon.
MrmNOT_VALID	If the icon uses a color table which contains colors other than foreground- color and background- color.
MrmFAILURE	On failure.

Availability

Motif 1.2 and later.

Description

1.Erroneously given as MrFetchBitmapLiteral in 1st edition.

`MrmFetchBitmapLiteral`() retrieves the named icon and converts it to a pixmap of depth 1 on the specified *screen* of the specified *display.* The icon must be defined as an exported value in a UIL source module. Foreground color pixels in the icon are set to 1 in the pixmap and background color pixels in the icon are set to 0 (zero) in the pixmap. The application is responsible for freeing the pixmap using `XFreePixmap`().

Usage

An icon retrieved with `MrmFetchBitmapLiteral`() can only use the special colors foreground color and background color in its color table. If the color table contains any other colors, `MrmFetchBitmapLiteral`() fails and returns MrmNOT_VALID.

As of Motif 1.2, values of type xbitmapfile cannot be converted to a pixmap using this function. xbitmapfile values can only be retrieved using `MrmFetchIconLiteral`().

Example

The following UIL and C code fragments show the retrieval of a bitmap from an Mrm hierarchy:

UIL:

```
...
! Declare a cursor icon using the default color table.
 value
      resize_down : exported icon (   '********',
                                      "   **   ",
                                      '   **   ',
                                      '** ** **',
                                      ' ****** ',
                                      '   **   ');
      ...
```

C:

```
...
extern MrmHierarchy    hierarchy;   /* Previously opened hierarchy. */
extern Widget          w;           /* Previously created widget. */

Pixmap          cursor_bits;
Dimension       width, height;
Cardinal        status;
static XColor   white = { 0, ~0, ~0, ~0, DoRed | DoGreen | DoBlue };
static XColor   black = { 0,  0,  0,  0, DoRed | DoGreen | DoBlue };

/* Get the icon as a pixmap of depth 1. */
```

```
status = MrmFetchBitmapLiteral (hierarchy, "resize_down", XtScreen
                                (w), XtDisplay (w), &cursor_bits,
                                &width, &height);
if (status != MrmSUCCESS)
    error_handler();
else {
    /* Create a cursor using the pixmap. */
    cursor = XCreatePixmapCursor (XtDisplay (w), cursor_bits,
                                  cursor_bits, &black, &white,
                                  width/2, height-1);
    /* Set the cursor in the widget. */
    XDefineCursor (XtWindow (w), cursor);
}
...
```

See Also

`MrmFetchIconLiteral`(3), `MrmFetchLiteral`(3), `value`(5), `color_table`(6), `icon`(6), `xbitmapfile`(6).

Name

MrmFetchColorLiteral – retrieve an exported color value from an Mrm hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchColorLiteral (MrmHierarchy   hierarchy,
                               String         name,
                               Display        *display,
                               Colormap       colormap,
                               Pixel          *pixel)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
name	Specifies the name of the color to retrieve.
display	Specifies the display.
colormap	Specifies the colormap in which the color is allocated.

Outputs

pixel	Returns a pixel value for the named color.

Returns

MrmSUCCESS	On success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmNOT_FOUND	If the specified color is not found or cannot be allocated.
MrmWRONG_TYPE	If the named value is not a color or rgb value.
MrmFAILURE	On failure.

Description

`MrmFetchColorLiteral`() retrieves a named color value and attempts to allocate a color cell containing it. The color must be defined as an exported value in a UIL source module. The color cell is allocated with `XAllocColor`() if the type of the value is rgb or with XAllocNamedColor if the type of the value is color. The *colormap* argument is used as a parameter to these functions. If *colormap* is NULL, Mrm uses the colormap returned by the `DefaultColormap`() macro.

Usage

If the color cannot be allocated because the specified *colormap* is full, `MrmFetchColorLiteral`() fails and returns MrmNOT_FOUND, not MrmFAILURE. The OSF documentation claims that when a color cannot be allocated, black or white is substituted. This was not true of Motif 1.2 variants: this transla-

tion did not take place, and you had to handle the error yourself. In Motif 2.1, however, MrmFetchColorLiteral() most certainly does substitute `XBlackPixelOfScreen`() if `XAllocColor`() fails; it does not use `XWhitePixelOfScreen`().

Example

The following UIL and C code fragments show the retrieval of color values from an Mrm hierarchy:

UIL:

```
...
value
foreground : exported rgb (255, 167, 0);
background : exported color ('mutant ninja turtle');
...
```

C:

```
Widget          toplevel;       /* Previously created widget. */
MrmHierarchy    hierarchy;      /* Previously opened Mrm hierarchy. */
Pixel           foreground, background;
Cardinal        status;
...
status = MrmFetchColorLiteral (hierarchy, "foreground", XtDisplay
(toplevel),
                                        NULL, &foreground);
if (status != MrmSUCCESS)
    error_handler();
status =    MrmFetchColorLiteral (hierarchy, "background", XtDisplay
(toplevel),
                                            NULL, &background);
if (status != MrmSUCCESS)
    error_handler();
...
```

See Also

`MrmFetchBitmapLiteral`(3), `MrmFetchIconLiteral`(3), `MrmFetchLiteral`(3), `value`(5), `color`(6), `color_table`(6), `rgb`(6).

Name

MrmFetchIconLiteral – retrieve an exported icon from an Mrm hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchIconLiteral ( MrmHierarchy   hierarchy,
                               String         name,
                               Screen         *screen,
                               Display        *display,
                               Pixel          foreground,
                               Pixel          background,
                               Pixmap         *pixmap)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
name	Specifies the name of an icon or xbitmapfile to retrieve.
screen	Specifies the screen of the display on which the pixmap is created.
display	Specifies the display.
foreground	Specifies the foreground color to use for the pixmap.
background	Specifies the background color to use for the pixmap.

Outputs

pixmap	Returns a pixmap created on the specified screen and display.

Returns

MrmSUCCESS	On success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmNOT_FOUND	If the specified icon or xbitmapfile is not found or a color in the icon's color table cannot be allocated.
MrmWRONG_TYPE	If the named value is not an icon or xbitmapfile value.
MrmFAILURE	On failure.

Description

`MrmFetchIconLiteral`() retrieves the named icon or xbitmapfile value and attempts to convert it to a pixmap on the specified *screen* of the display. The icon or xbitmap-file must be defined as an exported value in a UIL source module. The *foreground* pixel argument is used as the color for foreground pixels in an icon and pixels set to 1 in an xbitmapfile. The *background* pixel argument is used as the color for background pixels in an icon and pixels set to 0 (zero) in an xbitmapfile. Additional colors used by an icon are allocated in the colormap returned

by the DefaultColormap() macro. The application is responsible for freeing the pixmap using XFreePixmap().

Usage

If a color cannot be allocated because the specified colormap is full, MrmFetchIconLiteral() fails and returns MrmNOT_FOUND, not MrmFAILURE. The OSF documentation claims that when a color cannot be allocated, black or while is substituted, but in Motif 1.2 this translation did not take place, so you had to handle the error yourself. In Motif 2.1, XBlackPixelOfScreen() is used as a substitute if XAllocColor() fails; it does not use a corresponding XWhitePixelOfScreen().

Example

The following UIL and C code fragments illustrate the retrieval of a pixmap from an Mrm hierarchy:

UIL:

```
...
! Declare an icon using the default color table
value
    box : exported icon (   '****',
                            '*  *',
                            '*  *',
                            '****');
    ...
```

C:

```
extern MrmHierarchy    hierarchy;          /* Previously opened   */
                                           /*      hierarchy. */
extern Widget          drawing_area;       /* Previously created  */
                                           /*      widget.   */
extern GC              drawing_area_gc;    /* Previously defined   */
                                           /*     graphics context. */
Pixel                  foreground, background;
Pixmap                 box_pixmap;
unsigned int           box_width, box_height;
unsigned int           dont_care;
Cardinal               status;

/* Get values to use for pixmap foreground and background. */
XtVaGetValues (drawing_area, XmNforeground, &foreground,
                             XmNbackground, &background,
                             NULL);
/* Create the pixmap from the box icon in the hierarchy. */
```

```
status = MrmFetchIconLiteral (hierarchy, "box", XtScreen
(drawing_area),
                                XtDisplay (drawing_area), fore-
                                ground, background, &box_pixmap);
if (status != MrmSUCCESS)
    error_handler();
else {
    /* Get the size of the pixmap. */
    XGetGeometry (XtDisplay (drawing_area), box_pixmap, (Window
    *) &dont_care,
                  (int *) &dont_care, (int *) &dont_care,
                  &box_width, &box_height, &dont_care,
                  &dont_care);
    /* Draw the box in the drawing area. */
    XCopyArea (XtDisplay (drawing_area), box_pixmap, XtWindow
                (drawing_area),          drawing_area_gc, 0, 0,
                box_width, box_height, 10, 10);
    /* Free the pixmap. */
    XFreePixmap (box_pixmap);
}
```

See Also

`MrmFetchBitmapLiteral`(3), `MrmFetchColorLiteral`(3), `MrmFetchLiteral`(3), `value`(5), `color`(6), `color_table`(6), `icon`(6), `rgb`(6), `xbitmapfile`(6).

Name

MrmFetchLiteral – retrieve an exported value from an Mrm hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchLiteral (MrmHierarchy    hierarchy,
                          String          name,
                          Display         *display,
                          XtPointer       *value,
                          MrmCode         *type)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
name	Specifies the name of the value to retrieve.
display	Specifies the display.

Outputs

value	Returns a pointer to the value with the specified name.
type	Returns the type of the value retrieved.

Returns

MrmSUCCESS	On success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmNOT_FOUND	If the specified value is not found.
MrmWRONG_TYPE	If the type of the value specified cannot be converted by this procedure.

Description

`MrmFetchLiteral`() retrieves the named value and its type from the specified Mrm hierarchy. The value must be defined as an exported value in a UIL source module. The *display* argument is used to convert values of type font, fontset, and font_table. On success, this routine returns a pointer to the named value and the type of the value. The possible type values begin with MrmR*type* and are defined in the include file *<Mrm/MrmPublic.h>*. The application is responsible for freeing the returned value, except when it is a font or a fontset. font and fontset values are cached by Mrm and freed when the display is closed.

Usage

`MrmFetchLiteral`() cannot be used to retrieve values of certain types. You should retrieve icon values with `MrmFetchIconLiteral`() or `MrmFetchBitmapLiteral`(), xbitmapfile values with MrmFetchIconLiteral(), and color or rgb values with `MrmFetchColorLiteral`().

The storage allocated by Mrm for a boolean value is sizeof(int) not sizeof(Boolean). Because sizeof(Boolean) is less than sizeof(int) on many systems, applications should use an int pointer rather than a Boolean pointer as the value argument when retrieving a boolean.

Example

The following UIL and C code fragments illustrate the use of MrmFetchLiteral() to fetch various values from an Mrm hierarchy:

UIL:

```
...
value
    int_val     : 10;
    string_val  : 'okemo';
    ...
```

C:

```
...
extern MrmHierarchy     hierarchy;      /* Previously opened hierarchy. */
extern Display          *display;       /* Previously opened display.   */
int                     *int_ptr;
String                  string;
MrmCode                 type;
Cardinal                status;

status = MrmFetchLiteral (hierarchy, "int_val", display, (XtPointer *)
&int_ptr, &type);

if (status != MrmSUCCESS || type != MrmRtypeInteger)
    error_handler();
else
    printf ("Fetched integer %d\n", *int_ptr);

status = MrmFetchLiteral (hierarchy, "string_val", display, (XtPointer[1] *)
&string, &type);

if (status != MrmSUCCESS || type != MrmRtypeCString)
    error_handler();
else
    printf ("Fetched string '%s'\n", string);
...
```

1.Erroneously given as XtPoitner in 1st edition.

See Also

MrmFetchBitmapLiteral(3), MrmFetchColorLiteral(3), MrmFetchIconLiteral(3), MrmFetchSetValues(3), value(5), asciz_string_table(6), boolean(6), class_rec_name(6), color(6), compound_string(6), compound_string_table(6), float(6), font(6), font_table(6), fontset(6), icon(6), integer(6), integer_table(6), keysym(6), rgb(6), single_float(6), string(6), translation_table(6), wide_character(6), widget(6), xbitmapfile(6).

Name

MrmFetchSetValues – set widget resources to values retrieved from an Mrm hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchSetValues ( MrmHierarchy    hierarchy,
                             Widget          widget,
                             ArgList         arg_list,
                             Cardinal        num_args)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
widget	Specifies the object whose resources are modified.
arg_list	Specifies an array of name/UID-value pairs to be set.
num_args	Specifies the number of elements in arg_list.

Returns

MrmSUCCESS	On success.
MrmPARTIAL_SUCCESS	On partial success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmFAILURE	On failure.

Description

`MrmFetchSetValues`() sets the resources for an widget to named values obtained from the specified Mrm *hierarchy.* If a named value is not found or cannot be converted, the resource corresponding to that value is not set. If all the named values in *arg_list* are successfully retrieved, `MrmFetchSetValues`() returns MrmSUCCESS. If some values are successfully retrieved and others are not, MrmPARTIAL_SUCCESS is returned. If no values are successfully retrieved, MrmFAILURE is returned. When at least one value is successfully retrieved, `XtSetValues`() is called to modify the resources of object.

Usage

`MrmFetchSetValues`() sets the resources named in the name member of each item in *arg_list* to the value from the Mrm hierarchy named by the value member. This use differs from `XtSetValues`(), in that value member contains the name of a value to retrieve, not the value itself. Each named value must be defined as an exported value in a UIL source module.

The conversion of certain types may require a display pointer, screen pointer, background color, or foreground color. When these values are needed, Mrm

obtains them from widget. If foreground and background colors are needed for a conversion and widget does not have a background or foreground resource, Mrm uses black or white instead. If foreground and background colors are needed for a conversion and the XmNbackground or XmNforeground resources are specified in arg_list, they are used instead of the foreground and background of widget. As a result, if both an icon and foreground and/or background values are specified in the same argument list, the icon uses the colors specified in the list, rather than the colors of the widget.

Example

The following UIL and C code fragments illustrate the use of `MrmFetchSetValues()` to fetch a resource value from an Mrm hierarchy:

UIL:

```
...
value
! English language version of the confirm quit message:
confirm_quit_msg : 'Do you really want to quit?';
...
```

C:

```
extern MrmHierarchy    hierarchy;        /* Previously opened Mrm   */
                                         /*     hierarchy. */
extern Widget          yes_no_dialog;    /* Previously created yes/no  */
                                         /*     dialog. */

void DisplayConfirmQuit (void)
{
    static Arg args[] = {
        { XmNmessageString, (XtArgVal) "confirm_quit_msg" }
    };

    /* Set the message string for confirm quit. */
    MrmFetchSetValues (hierarchy, yes_no_dialog, args, XtNumber (args));
    /* Make the dialog appear. */
    XtManageChild (yes_no_dialog);
}
```

Structures

ArgList is defined as follows:

```
typedef struct {
    String          name;
    XtArgVal        value;
} Arg, *ArgList;
```

See Also

MrmFetchBitmapLiteral(3), MrmFetchColorLiteral(3), MrmFetchIconLiteral(3), MrmFetchLiteral(3), value(5).

Name

MrmFetchWidget – create the widget tree rooted at a named widget.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchWidget ( MrmHierarchy   hierarchy,
                          String         name,
                          Widget         parent,
                          Widget         *widget,
                          MrmType        *class)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
name	Specifies the name of the root widget of the widget tree to create.
parent	Specifies the parent of the root widget.

Outputs

widget	Returns the widget ID of the root widget.
class	Returns the UID class code for the widget class of the root widget.

Returns

MrmSUCCESS	On success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmNOT_FOUND	If the specified widget is not found.
MrmFAILURE	On failure.

Description

`MrmFetchWidget`() creates the named *widget* and recursively creates all of its children. Each child is managed by Mrm, unless declared unmanaged in the UIL source module. The root widget should be defined as exported in a UIL source module. Mrm supports the MrmNcreateCallback, which if defined, is called after a widget is created. The prototype of an MrmNcreateCallback is the same as any other Xt callback procedure. The call_data passed to the callback is an XmAnyCallbackStruct.

Usage

Each successful call to `MrmFetchWidget`() results in the creation of a new widget tree, even if *name* has been fetched previously. As a result, you can use a widget tree definition from an Mrm hierarchy as a template for creating multiple instances of the same widget tree. The widget at the root of the tree is not managed by Mrm, so your application must manage this widget to make the tree visible.

In Motif 1.2 and earlier, `MrmFetchWidget`() returns MrmSUCCESS if the root widget is retrieved successfully, even if one or more of its children are not. As of Motif 1.2.1, if `MrmFetchWidget`() cannot find a child widget, it returns MrmNOT_FOUND and does not create any widgets.

As of Motif 1.2, the possible MrmType values returned in class are not defined in any of the Mrm include files, although the OSF documentation claims that they are defined in <*Mrm/Mrm.h*>. If you need to check the widget class of a widget created with `MrmFetchWidget`(), use `XtClass`() or one of the XmIs*() macros.

Example

The following UIL and C code fragments illustrate the retrieval of a widget hierarchy from an Mrm hierarchy:

UIL:

```
...
! Define a simple widget tree, with form at the root.
object label     : XmLabel { };
object button    : XmPushButton { };
object form      : exported XmForm {
    controls {
        XmLabel          label;
        XmPushButton     button;
    };
};
...
```

C:

```
extern Widget           toplevel;       /* Previously defined widget. */
extern MrmHierarchy     hierarchy;      /* Previously opened hierarchy. */
Widget                  form;
MrmType                 class;
Cardinal                status;

status = MrmFetchWidget (hierarchy, "form", toplevel, &form, &class);

if (status != MrmSUCCESS)
    error_handler();
...
```

Structures

The MrmNcreateCallback function is passed an XmAnyCallbackStruct, which is defined as follows:

```
typedef struct {
    int         reason;         /* MrmCR_CREATE */
    XEvent      *event;         /* NULL */
} XmAnyCallbackStruct;
```

See Also

`MrmFetchWidgetOverride`(3), `MrmOpenHierarchy`(3), `MrmOpenHierarchyPerDisplay`(3), `object`(5), `widget`(6).

Name

MrmFetchWidgetOverride – create the widget tree rooted at a named widget and override the resources set in the UID file.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmFetchWidgetOverride (MrmHierarchy  hierarchy,
                                 String        name,
                                 Widget        parent,
                                 String        override_name,
                                 ArgList       arg_list,
                                 Cardinal      num_args,
                                 Widget        *widget,
                                 MrmType       *class)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
name	Specifies the name of the root widget of the widget tree to create.
parent	Specifies the parent of the root widget.
override_name	Specifies the name to use when creating the root widget. If NULL, name is used.
arg_list	Specifies an array of resource/value pairs to set on the root widget when it is created. If NULL, no resources are set.
num_args	Specifies the number of elements in arg_list. Must be 0 (zero) if arg_list is NULL.

Outputs

widget	Returns the widget ID of the root widget.
class	Returns the UID class code for the widget class of the root widget.

Returns

MrmSUCCESS	On success.
MrmBAD_HIERARCHY	If hierarchy is NULL or does not point to a valid Mrm hierarchy.
MrmNOT_FOUND	If the specified widget is not found.
MrmFAILURE	On failure.

Description

`MrmFetchWidgetOverride`() creates the named *widget* and recursively creates all of its children. The root widget should be defined as exported in a UIL source module. *arg_list* is used to specify additional resource/value pairs that override those specified in the widget definition in a UIL source module. Each

child is managed by Mrm unless declared unmanaged in the UIL source module. Mrm supports the MrmNcreateCallback, which if defined, is called after a widget is created. The prototype of an MrmNcreateCallback is the same as any other Xt callback procedure. The call_data passed to the callback is an XmAnyCallback-Struct.

Usage

`MrmFetchWidgetOverride`() allows an application to create a widget defined in an Mrm hierarchy while specifying application-defined resource values that can supplement or override those specified in the UIL definition. The function sets the resources of the root widget that are named in the name member of each item in *arg_list* to value specified in the value member. The resource of any children of the root widget are not affected.

Each successful call to `MrmFetchWidgetOverride`() results in the creation of a new widget tree, even if *name* has been fetched previously. As a result, you can use a widget tree definition from an Mrm hierarchy as a template for creating multiple instances of the same widget tree. The widget at the root of the tree is not managed by Mrm, so your application must manage this widget to make the tree visible.

In Motif 1.2 and earlier, `MrmFetchWidget`() returns MrmSUCCESS if the root widget is retrieved successfully, even if one or more of its children are not. As of Motif 1.2.1, if `MrmFetchWidget`() cannot find a child widget, it returns MrmNOT_FOUND and does not create any widgets.

As of Motif 1.2, the possible MrmType values returned in class are not defined in any of the Mrm include files, although the OSF documentation claims that they are defined in <*Mrm/Mrm.h*>. If you need to check the widget class of a widget created with `MrmFetchWidgetOverride`(), *use* `XtClass`*()* or one of the XmIs*() macros.

Example

The following UIL and C code fragments illustrate the retrieval of a widget hierarchy from an Mrm hierarchy using MrmFetchWidgetOverride()[1]:

```
UIL:
    ...
    object error_dialog: exported XmErrorDialog {
        arguments {
            XmNmessageString = "If you can read this, file a bug report.";
            XmNdialogStyle =
            XmDIALOG_FULL_APPLICATION_MODAL;
```

1.Erroneously given as MwmFetchWidgetOverride() in 1st edition.

```
        };
    };
    ...
```

C:

```
    extern Widget           toplevel;     /* Previously created widget. */
    extern MrmHierarchy     hierarchy;    /* Previously opened hierarchy. */

    void display_error (String message)
    {
        Arg         arg_list[1];
        XmString    s;
        Cardinal    status;
        Widget      error_dialog;
        MrmType     class;

        s = XmStringCreateLocalized (message);
        XtSetArg (arg_list[0], XmNmessageString, s);
        status = MrmFetchWidgetOverride (hierarchy, "error_dialog",
                                         toplevel,  "error_dialog",
                                         arg_list, 1, &error_dialog,
                                         &class);
        XmStringFree (s);

        if (status != MrmSUCCESS)
            handle_error();
        else
            XtManageChild (error_dialog);
    }
```

Structures

ArgList is defined as follows:

```
typedef struct {
    String      name;
    XtArgVal    value;
} Arg, *ArgList;
```

The MrmNcreateCallback function is passed an XmAnyCallbackStruct, which is defined as follows:

```
typedef struct {
    int         reason;     /* MrmCR_CREATE */
    XEvent      *event;     /* NULL */
} XmAnyCallbackStruct;
```

See Also

MrmFetchWidget(3), MrmOpenHierarchy(3), MrmOpenHierarchyPerDisplay(3), object(5), widget(6).

Name

MrmInitialize – prepare the Mrm library for use.

Synopsis

#include <Mrm/MrmPublic.h>

void MrmInitialize (void)

Description

`MrmInitialize`() initializes the Mrm library. As part of the initialization, all Motif widget classes are registered in the Mrm widget class database with `MrmRegisterClass`().

Usage

Applications should call `MrmInitialize`() before the Xt Toolkit is initialized and before calling any other Mrm functions. If the routine is not called before `MrmOpenHierarchyPerDisplay`(), future calls to `MrmFetchWidget`() and `MrmFetchWidgetOverride`() will fail. Applications should only call `MrmInitialize`() once.

Example

The following code fragment illustrates the use of `MrmInitialize`(): [1]

```
...
Widget          toplevel;
XtAppContext    app_context;
MrmHierarchy    hierarchy;
Cardinal        status;

XtSetLanguageProc (NULL, (XtLanguageProc) NULL, NULL);

MrmInitialize();

toplevel = XtVaOpenApplication (&app_context, "App", NULL, 0, (Cardi-
                                nal *) &argc, &argv, NULL, session-
                                ShellWidgetClass, NULL);
...
```

See Also

`MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `MrmOpenHierarchy`(3), `MrmOpenHierarchyPerDisplay`(3), `MrmRegisterClass`(3).

1.From X11R6, XtAppInitialize() is marked as obsolete. The SessionShell is only available from X11R6 onwards, and it replaces the deprecated ApplicationShell widget class.

Name

MrmOpenHierarchy – open an Mrm hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchy ( MrmCount            num_files,
                            String              file_name_list[],
                            MrmOsOpenParamPtr   *os_params,
                            MrmHierarchy        *hierarchy)
```

Inputs

num_files	Specifies the number of files in file_name_list.
file_name_list	Specifies an array of UID file names to associate with the hierarchy.
os_params	Specifies operating system dependent settings.

Outputs

hierarchy	Returns an open Mrm hierarchy consisting of the specified files.

Returns

MrmSUCCESS	On success.
MrmNOT_FOUND	If one or more files cannot be opened.
MrmNOT_VALID	If the version of Mrm is older than the version of any UID file.
MrmDISPLAY_NOT_OPENED	If a display pointer cannot be found.
MrmFAILURE	On failure.

Availability

In Motif 1.2, `MrmOpenHierarchy`() is obsolete. It has been superseded by `MrmOpenHierarchyPerDisplay`().

Description

`MrmOpenHierarchy`() opens an Mrm hierarchy consisting of one or more UID files. This routine is similar to `MrmOpenHierarchyPerDisplay`(), except that it does not take a display parameter. `MrmOpenHierarchy`() is retained for compatibility with Motif 1.1 and should not be used in newer applications.

Usage

`MrmOpenHierarchy`() relies on the Motif widget library to locate a display pointer. To ensure that a display pointer can be found, an application must create an ApplicationShell before calling `MrmOpenHierarchy`(). The display pointer is used as a parameter to `XtResolvePathname`(), which locates the files in

file_name_list. If an application creates multiple ApplicationShells on different displays, the display pointer chosen by this routine is undefined.

See the `MrmOpenHierarchyPerDisplay`() manual page for a full explanation of the process of opening an Mrm hierarchy, including the search path that is used to find the UID files.

See Also

`MrmCloseHierarchy`(3), `MrmOpenHierarchyPerDisplay`(3).

Name

MrmOpenHierarchyFromBuffer – open an Mrm hierarchy from a buffer

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchyFromBuffer (unsigned char **buffer*, MrmHierarchy **hierarchy_id*)

Inputs

buffer	Specifies a stream of bytes representing a UID file contents.

Outputs

hierarchy_id	Returns an open Mrm hierarchy.

Returns

MrmSUCCESS	On Success.
MrmNOT_VALID	If the version of Mrm is older than the data contained within the buffer.
MrmDISPLAY_NOT_OPENED	If a display pointer cannot be found.
MrmFAILURE	On failure.

Availability

Motif 2.0 and later.

Description

`MrmOpenHierarchyFromBuffer`() opens an Mrm hierarchy using the stream of data specified by buffer, which is presumably the contents of a previously opened UID file loaded into memory. It could, however, be dynamically constructed.

Usage

`MrmOpenHierarchyFromBuffer`() relies on the Motif widget library to locate a display pointer using internal default values. A pointer is only found if a ApplicationShell has been created before calling `MrmOpenHierarchyFromBuffer`().

See Also

`MrmOpenHierarchy`(3), `MrmOpenHierarchyPerDisplay`(3), `MrmCloseHierarchy`(3).

Name

MrmOpenHierarchyPerDisplay – open an Mrm hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmOpenHierarchyPerDisplay (Display          *display,
                                     MrmCount         num_files,
                                     String           file_name_list[],
                                     MrmOsOpenParamPtr os_params_list[],
                                     MrmHierarchy     *hierarchy)
```

Inputs

display	Specifies the display.
num_files	Specifies the number of files in file_name_list.
file_name_list	Specifies an array of file names to associate with the hierarchy.
os_params_list	Specifies an array of operating system dependent settings.

Outputs

hierarchy	Returns an open Mrm hierarchy consisting of the specified files.

Returns

MrmSUCCESS	On success.
MrmNOT_FOUND	If one or more files cannot be opened.
MrmNOT_VALID	If the version of Mrm is older than version of the any UID file.
MrmDISPLAY_NOT_OPENED	If a display pointer cannot be found.
MrmFAILURE	On failure.

Availability

Motif 1.2 and later.

Description

`MrmOpenHierarchyPerDisplay`() opens an Mrm hierarchy consisting of one or more UID files. An Mrm hierarchy must be opened before any values are retrieved or widgets created with the MrmFetch*() routines. When an Mrm hierarchy is successfully opened, each UID file specified in *file_name_list* is opened and consumes a file descriptor. No files are opened if a value other than MrmSUCCESS is returned. The UID files are subsequently closed when the hierarchy is closed with `MrmCloseHierarchy`(). As of Motif 1.2, settings in the *os_params_list* parameter are only useful to the UIL compiler. Application programs should always specify NULL for this argument.

Usage

The MrmFetch*() routines retrieve a named value or widget by searching the UID files for a hierarchy in the order that they are specified in *file_name_list*. If a named value or widget occurs in more than one of the UID files, the value is retrieved from the file that occurs first in the array. Once an Mrm hierarchy has been opened, the UID files associated with the hierarchy must not be modified or deleted until the hierarchy is closed.

Files specified in *file_name_list* may be full or partial path names. When a file name starts with a slash (/), it specifies a full path name and `MrmOpenHierarchyPerDisplay`() opens the file. Otherwise, the file name specifies a partial path name which causes `MrmOpenHierarchyPerDisplay`() to look for the file using a search path.

`XtResolvePathname`() is used to locate the file in the search path. The UIDPATH environment variable specified the search path for UID files. Each directory in the search path can contain the substitution character %U; the partial path name specified by file_name_list is substituted for %U. In addition, the path can also use the substitution characters accepted by `XtResolvePathname()`. The path is first searched with %S mapped to *.uid*. If the file is not found the path is searched again with %S mapped to NULL.

If UIDPATH is not set, `MrmOpenHierarchyPerDisplay`() uses a default search path. If the XAPPLRESDIR environment variable is set, the routine searches the following path; the class name of the application is substituted for %N, the language string of the display argument is substituted for %L, and the language component of the language string is substituted for %l.

$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/1%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 XAPPLRESDIR is not set, MrmOpenHierarchyPerDisplay() searches the same path, except that XAPPLRESDIR is replaced by HOME. These paths are vendor-dependent and a vendor may use different directories for /usr/lib/X11 and /usr/include/X11.

Example

The following code fragment illustrates the use of MrmOpenHierarchyPerDisplay():[1]

```
...
MrmHierarchy    hierarchy;
XtAppContext    app_context;
Widget          toplevel;
String          uid_files[] = { "/usr/lib/app/app", "strings" };
Cardinal        status;

XtSetLanguageProc (NULL, NULL, NULL);

MrmInitialize();
toplevel = XtVaOpenApplication (&app_context, "App", NULL, 0, &argc,
                argv, NULL, sessionShellWidgetClass, NULL);

status = MrmOpenHierarchyPerDisplay (XtDisplay (toplevel),
                                     XtNumber (uid_files), uid_files,
                                     NULL, &hierarchy);

if (status != MrmSUCCESS)
    error_handler();
...
```

See Also

MrmCloseHierarchy(3), MrmFetchBitmapLiteral(3), MrmFetchColorLiteral(3), MrmFetchIconLiteral(3), MrmFetchLiteral(3), MrmFetchWidget(3), MrmFetchWidgetOverride(3).

1.From X11R6, XtAppInitialize() is marked as obsolete. The SessionShell is only available from X11R6 onwards, and it replaces the deprecated ApplicationShell widget class.

Name

MrmRegisterClass – register a widget creation function for a non-Motif widget.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterClass (MrmType      class_code,
                           String       class_name,
                           String       create_proc_name,
                           Widget       (*create_proc) (Widget, char *,
ArgList, Cardinal),
                           WidgetClass  widget_class)
```

Inputs

class_code	This argument is obsolete and is ignored.
class_name	This argument is obsolete and is ignored.
create_proc_name	Specifies the case-sensitive name of the widget creation function.
create_proc	Specifies a pointer to the widget creation procedure.
widget_class	Specifies a pointer to the widget class record or NULL.

Returns

MrmSUCCESS	On success.
MrmFAILURE	On failure.

Description

`MrmRegisterClass`() supplies Mrm with information it needs to create a user-defined widget, which is any widget that is not built into UIL and Mrm. A user-defined widget cannot be created until its class is registered.

Usage

A user-defined widget is defined in a UIL source module by specifying the *create_proc_name* in its declaration. *create_proc_name* must be all uppercase characters when used in a UIL module compiled with case-insensitive names because this setting causes the UIL compiler to store procedure name references in all uppercase characters.

If `MrmRegisterClass`() is called with a *class_name* that has been registered previously, the new *`create_proc`* and *widget_class* replace the previous values. There is no way to unregister a previously registered class. As of Motif 1.2, a small amount of memory may be leaked when a class is registered multiple times.

The *widget_class* argument allows Mrm to convert a class name specified in a UIL *class_rec_name* literal into a widget class pointer. If NULL is specified, the widget class pointer is not accessible with the *class_rec_name* type.

Example

The following UIL and C code fragments illustrate the creation of an instance of the Athena panner widget from UIL. Like any other widget defined in a UIL module, it is created with a call to MrmFetchWidget() or `MrmFetchWidgetOverride`():

UIL:

```
...
procedure XawCreatePannerWidget;

object panner : user_defined procedure XawCreatePannerWidget { };
...
```

C:

```
 Widget XawCreatePannerWidget (Widget parent, String name, ArgList
                args, Cardinal num_args)
{
     return XtCreateWidget (name, pannerWidgetClass, parent, args,
                num_args);
}
...
MrmRegisterClass (0, NULL,
                        "XawCreatePannerWidget",
                        XawCreatePannerWidget,
                        &pannerWidgetClass);
...
```

Procedures

The create_proc parameter has the following syntax:

Widget create_proc (Widget *parent*, String *name*, ArgList *args*,

Cardinal *num_args*)

The procedure takes four arguments. The first, *parent*, is the parent of the widget that is being created. *name* is the name of the widget. The last two arguments, *args* and *num_args*, specify the initial resource settings for the widget. The procedure returns the widget ID of the newly created widget.

See Also

`MrmInitialize`(3), `MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `object`(5), `class_rec_name`(6).

Name

MrmRegisterNames – register application-defined values and procedures.

Synopsis

#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterNames (MrmRegisterArgList *name_list*, MrmCount *count*)

Inputs

name_list	Specifies an array of name/value pairs to be registered with Mrm.
count	Specifies the number of elements in name_list.

Returns

MrmSUCCESS	On success.
MrmFAILURE	On failure.

Description

`MrmRegisterNames`() registers an array of name/value pairs that are used as identifiers and procedures in a UIL source module. Names registered with this routine are accessible from any open Mrm hierarchy. By contrast, names registered with `MrmRegisterNamesInHierarchy`() are only accessible from the hierarchy in which they are registered.

If `MrmRegisterNames`() is called with a name that has been registered previously, the old value associated with the name is replaced by the new value. There is no way to unregister a previously registered name.

Usage

The MrmRegisterArg structure consists of a name and an associated value. The case of name is significant. name must be in all uppercase characters if name is used in a UIL module compiled with case-insensitive names, because this setting causes the UIL compiler to store procedure and identifier name references in all uppercase characters.

The *name_list* array can contain names that represent both callback procedures and identifier values. A procedure value in *name_list* should be a pointer to a function of type XtCallbackProc. An identifier value is any application-defined value that is exactly sizeof (XtPointer). Mrm makes no distinction between procedures and identifiers, although an application may organize them in two separate arrays for clarity. A distinction is made in a UIL source module, where any name used must be declared as either a procedure or an identifier.

Procedures and identifiers must be registered with `MrmRegisterNames`() or `MrmRegisterNamesInHierarchy`() before an application attempts to cre-

ate a widget that references them. Mrm converts a procedure or identifier reference to a value by first searching hierarchy-local names registered with MrmRegisterNamesInHierarchy(). If the value is not found, the search continues with global names registered with MrmRegisterNames().

Example

The following UIL and C code fragments illustrate the use of MrmRegisterNames():

UIL:

```
...
identifier user_id;
procedure activate();

object button : XmPushButton {
    callbacks {
        XmNactivateCallback = procedure activate;
    };
};
...
```

C:

```
...
extern XtCallbackProc activate;
int                 user_id = getuid();
MrmRegisterArg      names[2];

names[0].name = "activate";
names[0].value = (XtPointer) activate;
names[1].name = "user_id";
names[1].value = (XtPointer) user_id;

MrmRegisterNames (names, XtNumber (names));
...
```

Structures

MrmRegisterArgList is defined as follows:

```
typedef struct {
    String      name;       /* case-sensitive name */
    XtPointer   value;      /* procedure/value to associate with name */
} MrmRegisterArg, *MrmRegisterArglist;
```

See Also

MrmRegisterNamesInHierarchy(3), identifier(5), procedure(5).

Name

MrmRegisterNamesInHierarchy – register application-defined values and procedures for use in a specific UIL hierarchy.

Synopsis

```
#include <Mrm/MrmPublic.h>

Cardinal MrmRegisterNamesInHierarchy (  MrmHierarchy          hierarchy,
                                        MrmRegisterArgList    name_list,
                                        MrmCount              count)
```

Inputs

hierarchy	Specifies an Mrm hierarchy obtained from a previous call to MrmOpenHierarchy() or MrmOpenHierarchyPerDisplay().
name_list	Specifies an array of name/value pairs to be registered with Mrm.
count	Specifies the number elements in name_list.

Returns

MrmSUCCESS	On success.
MrmFAILURE	On failure.

Description

`MrmRegisterNamesInHierarchy`()[1] registers an array of name/value pairs that are used as identifiers and procedures in a UIL source module. Names registered with this routine are accessible only within the specified *hierarchy.* By contrast, names registered with `MrmRegisterNames`() are accessible from any open hierarchy.

If `MrmRegisterNamesInHierarchy`() is called with a name that has been registered previously in the same hierarchy, the old value associated with the name is replaced by the new value. There is no way to unregister a previously registered name while the hierarchy is open. However, closing the hierarchy automatically unregisters all names.

Usage

The MrmRegisterArg structure consists of a name and an associated value. The case of name is significant. name must be in all uppercase characters if name is used in a UIL module compiled with case-insensitive names, because this setting causes the UIL compiler to store procedure and identifier name references in all uppercase characters.

1.Erroneously given as MrmRegisterNames() in 1st edition.

The *name_list* array can contain names that represent both callback procedures and identifier values. A procedure value in *name_list* should be a pointer to a function of type XtCallbackProc. An identifier value is any application-defined value that is exactly sizeof (XtPointer). Mrm makes no distinction between procedures and identifiers, although an application may orgranize them in two separate arrays for clarity. A distinction is made in a UIL source module, where any name used must be declared as either a procedure or an identifier.

Procedures and identifiers must be registered with `MrmRegisterNames`() or `MrmRegisterNamesInHierarchy`() before an application attempts to create a widget which references them. Mrm converts a procedure or identifier reference to a value by first searching hierarchy-local names registered with `MrmRegisterNamesInHierarchy`(). If the value is not found, the search continues with global names registered with `MrmRegisterNames`().

Example

The following code fragment illustrates the use of `MrmRegisterNamesInHierarchy`():

```
/* Open a hierarchy and register it's file name list. */

Cardinal register_and_open (Display display, MrmCount count, String *files)
{
    Cardinal            status;
    int                 *count = (int *) malloc ((unsigned) sizeof (int));
    MrmRegisterArg      names[2];
    if (count == NULL)
        return (MrmFAILURE);

    names[0].name = "file_list";
    names[0].value = (XtPointer) file_list;
    names[1].name = "file_count";
    names[1].value = (XtPointer) file_count;

    status = MrmOpenHierarchyPerDisplay (display, count, files, NULL,

                &hierarchy);

    if (status != MrmSUCCESS)
        return (status);

    status = MrmRegisterNamesInHierarchy (*hierarchy, names, XtNumber
    (names));

    return (status);
}
```

Structures

MrmRegisterArgList is defined as follows:

```
typedef struct {
    String      name;       /* case-sensitive name */
    XtPointer   value;      /* procedure/value to associate with name */
} MrmRegisterArg, *MrmRegisterArglist;
```

See Also

`MrmRegisterNames`(3), `identifier`(5), `procedure`(5).

Section 4 - Mrm Clients

This page describes the format and contents of each reference page in Section 4, which covers the Motif clients.

Name

Client – a brief description of the client.

Syntax

This section describes the command-line syntax for invoking the client. Anything in **bold** should be typed exactly as shown. Items in *italics* are parameters that should be replaced by actual values when you enter the command. Anything enclosed in brackets is optional.

Availability

This section appears for functions that were added in Motif 2.0 or later.

Description

This section explains the operation of the client. In some cases, additional descriptive sections appear later on in the reference page.

Options

This section lists available command-line options.

Environment

If present, this section lists shell environment variables used by the client. This section does not list the DISPLAY and XENVIRONMENT variables, which are used by all clients. These variables are used as follows:

DISPLAY

To get the default display name (specifically, the host, server/display, and screen). The DISPLAY variable typically has the form:

hostname:*server.screen*

XENVIRONMENT

To get the name of a resource file containing host-specific resources. If this variable is not set, the resource manager will look for a file called .Xdefaults-**hostname** (where **hostname** is the name of a particular host) in the user's home directory.

Bugs

If present, this section lists any problems that could arise when using the client.

See Also

This section refers you to related clients, functions, or widget classes. The numbers in parentheses following each reference refer to the sections of the book in which they are found.

Name

mwm – the Motif Window Manager (*mwm*).

Syntax

mwm [*options*]

Description

The Motif Window Manager, *mwm*, provides all of the standard window management functions. It allows you to move, resize, iconify/deiconify, maximize, and close windows and icons, focus input to a window or icon, and refresh the display. *mwm* provides much of its functionality via a frame that (by default) is placed around every window on the display. The *mwm* frame has the three-dimensional appearance characteristic of the OSF/Motif graphical user interface.

The rest of this reference page describes how to customize *mwm.* It does not provide information on using *mwm*. For information on using the window manager, see Volume 3, *X Window System User's Guide, Motif Edition*.

Options

-display [*host]*:*server*[.*screen*]

Specifies the name of the display on which to run *mwm. host* is the hostname of the physical display, *server* specifies the server number, and *screen* specifies the screen number. Either or both of the *host* and *screen* elements can be omitted. If *host* is omitted, the local display is assumed. If *screen* is omitted, screen 0 is assumed (and the period is unnecessary). The colon and (display) *server* are necessary in all cases.

-multiscreen

Specifies that *mwm* should manage all screens on the display. The default is to manage only screen 0. You can specify an alternate screen by setting the DISPLAY environment variable or using the -display option. You can also specify that mwm manage all screens by assigning a value of True to the multiScreen resource variable.

-name *app_name*

Specifies the name under which resources for the window manager should be found.

-screens *screen_name*[*screen_name*]...

Assigns resource names to the screens *mwm* is managing. (By default, the screen number is used as the *screen_name*.) 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. If there are more screens

than names, resources for the remaining screens will be retrieved using the first *screen_name*.

-xrm *resourcestring*

Specifies a resource name and value to override any defaults. This option is very useful for setting resources that do not have explicit command-line arguments.

Window Manager Components

The *mwm* window frame contains various components that perform different functions. The title bar stretches across the top of the window and contains the title area and the minimize, maximize, and window menu buttons. The title area displays the window title and can be used to move the window. The minimize button iconifies the window, while the maximize button enlarges the window to fill the entire screen. The window menu button posts the **Window Menu**. The resize border handles surround the window; they are used to resize the window in a particular direction. A window can also have an optional matte decoration between the client area and the window frame. The matte is not part of the window frame and it has no functionality. At times, *mwm* uses dialog boxes or feedback windows to communicate with the user.

An icon is a small graphic representation of a window. When a window is iconified using the minimize button, it is replaced on the screen by its icon. Iconifying windows reduces clutter on the screen. *mwm* provides a separate window, call the icon box, that can hold icons. Using the icon box keeps icons from cluttering the screen.

By default, *mwm* uses an explicit keyboard selection policy, which means that once a window has the keyboard focus, it keeps it until another window is explicitly given the focus. Windows can overlap, which means that they are arranged in a global stacking order on the screen. A window that is higher in the stacking order obscures windows below it in the stacking order if they overlap. Each application has its own local stacking order; transient windows remain above their parents by default in the local stacking order.

Customization

Like any X application, *mwm* uses resources to control its appearance and behavior. The window manager builds its resource database just like any other X client. Mwm is the resource class name for *mwm*. You can place mwm resources in your regular resource file (*.Xdefaults*) in your home directory or you can create a file called *Mwm* (also in your home directory) for *mwm* resources only. If you place conflicting specifications in both files, the resources in*.Xdefaults* take precedence.

The default operation of the mouse, the keyboard, and menus in *mwm* is controlled by a system-wide resource description file, *system.mwmrc*. This file describes the contents of the **Window Menu** and **Root Menu**, as well as the key and button combinations that manage windows. To modify the behavior of *mwm*, you can edit a copy of this file in your home directory. The version of the file in your home directory should be called.*mwmrc*, unless you specify an alternate name using the configFile resource.

An *mwm* resource description file is a standard text file. Items are separated by blanks, tabs, and newlines. A line that begins with an exclamation mark (!) or a number sign (#) is treated as a comment. If a line ends with a backslash (\), the subsequent line is considered a continuation of that line.

Component Appearance Resources

mwm provides some resources that specify the appearance of particular window manager components, such as the window frame, menus, and icons. Component appearance resources can be specified for particular window manager components or all components. To specify a resource for all components, use the following syntax:

Mwm**resource_name: resource_value*

The window manager components have the following resource names associated with them:

Component	ResourceName
Menu	menu
Icon	icon
Client window frame	client
Feedback/dialog box	feedback
Title bar	title

These resource names can be used to specify particular window manager components in a resource specification. To specify a resource for a specific component, use the following syntax:

Mwm*[*component_name*]**resource_name: resource_value*

The title bar is a descendant of the client window frame, so you can use title to specify the appearance of the title bar separately from the rest of the window frame. You can also specify resources for individual menus by using *menu*, followed by the name of the menu.

The following component appearance resources apply to all window manager components. Unless a default value is specified, the default varies based on system specifics such as the visual type of the screen:

background (class Background)
: Specifies the background color.

backgroundPixmap (class BackgroundPixmap)
: Specifies the background pixmap of the *mwm* decoration when the window does not have the input focus.

bottomShadowColor (class Foreground)
: Specifies the color to be used for the lower and right bevels of the window manager decoration.

bottomShadowPixmap (class BottomShadowPixmap)
: Specifies the pixmap to be used for the lower and right bevels of the window manager decoration.

fontList (class FontList)
: Specifies the font to be used in the window manager decoration. The default is fixed.

foreground (class Foreground)
: Specifies the foreground color.

saveUnder (class SaveUnder)
: Specifies whether save unders are used for *mwm* components. The default value is False, which means that save unders are not used on any window manager frames.

topShadowColor (class Background)
: Specifies the color to be used for the upper and left bevels of the window manager decoration.

topShadowPixmap (class TopShadowPixmap)
: Specifies the pixmap to be used for the upper and left bevels of the window manager decoration.

The following component appearance resources apply to the window frame and icons. Unless a default value is specified, the default varies based on system specifics such as the visual type of the screen:

activeBackground (class Background)
: Specifies the background color of the *mwm* decoration when the window has the input focus.

activeBackgroundPixmap (class ActiveBackgroundPixmap)
Specifies the background pixmap of the *mwm* decoration when the window has the input focus.

activeBottomShadowColor (class Foreground)
Specifies the bottom shadow color of the *mwm* decoration when the window has the input focus.

activeBottomShadowPixmap (class BottomShadowPixmap)
Specifies the bottom shadow pixmap of the *mwm* decoration when the window has the input focus.

activeForeground (class Foreground)
Specifies the foreground color of the *mwm* decoration when the window has the input focus.

activeTopShadowColor (class Background)
Specifies the top shadow color of the *mwm* decoration when the window has the input focus.

activeTopShadowPixmap (class TopShadowPixmap)
Specifies the top shadow Pixmap of the *mwm* decoration when the window has the input focus.

General Appearance and Behavior Resources

mwm also provides resources that control the appearance and behavior of the window manager as a whole. These resources specify features such as the focus policy, interactive window placement, and the icon box. To specify a general appearance and behavior resource, use the following syntax:

Mwm**resource_name: resource_value*

The following general appearance and behavior resources can be specified:

autoKeyFocus (class AutoKeyFocus)
If True (the default), when the focus window is withdrawn from window management or is iconified, the focus bounces back to the window that previously had the focus. This resource is available only when keyboardFocusPolicy is explicit. If False, the input focus is not set automatically. autoKeyFocus and startupKeyFocus should both be True to work properly with tear-off menus.

autoRaiseDelay (class AutoRaiseDelay)
Specifies the amount of time (in milliseconds) that *mwm* will wait before raising a window after it receives the input focus. The default is 500. This resource is available only when focusAutoRaise is True and the keyboardFocusPolicy is pointer.

bitmapDirectory (class BitmapDirectory)

Identifies the directory to be searched for bitmaps referenced by *mwm* resources (if an absolute pathname to the bitmap file is not given). The default is */usr/-include/-X11/-bitmaps*, which is considered the standard location on many systems. Note, however, that the location of the bitmap directory may vary in different environments. If a bitmap is not found in the specified directory, XBMLANGPATH is searched.

clientAutoPlace (class ClientAutoPlace)

Specifies the location of a window when the user has not specified a location. If True (the default), windows are positioned with the upper-left corners of the frames offset horizontally and vertically, so that no two windows completely overlap. If False, the currently configured position of the window is used. In either case, *mwm* attempts to place the windows totally on screen.

colormapFocusPolicy (class ColormapFocusPolicy)

Specifies the colormap focus policy. Takes three possible values: keyboard, pointer, and explicit. If keyboard (the default) is specified, the input focus window has the colormap focus. If explicit is specified, a colormap selection action is done on a client window to set the colormap focus to that window. If pointer is specified, the client window containing the pointer has the colormap focus.

configFile (class ConfigFile)

Specifies the pathname for the *mwm* startup file. The default startup file is.*mwmrc*.
mwm searches for the configuration file in the user's home directory. If the configFile resource is not specified or the file does not exist, *mwm* defaults to an implementation-specific standard directory (the default is */usr/lib/X11/system.mwmrc*).

If the LANG environment variable is set, *mwm* looks for the configuration file in a *$LANG* subdirectory first. For example, if the LANG environment variable is set to Fr (for French), *mwm* searches for the configuration file in the directory *$HOME/Fr* before it looks in *$HOME*. Similarly, if the configFile resource is not specified or the file does not exist, *mwm* defaults to */usr/lib/X11/$LANG/system.mwmrc* before it reads */usr/lib/X11/system.mwmrc*.

If the configFile pathname does not begin with ~/, *mwm* considers it to be relative to the current working directory.

deiconifyKeyFocus (class DeiconifyKeyFocus)
: If True (the default), a window receives the input focus when it is normalized (deiconified). This resource applies only when the keyboardFocusPolicy is explicit.

doubleClickTime (class DoubleClickTime)
: Specifies the maximum time (in milliseconds) between the two clicks of a double click. The default is the display's multi-click time.

enableWarp (class EnableWarp)
: If True (the default), causes *mwm* to *warp* the pointer to the center of the selected window during resize and move operations invoked using keyboard accelerators. (The cursor symbol disappears from its current location and reappears at the center of the window.) If False, *mwm* leaves the pointer at its original place on the screen, unless the user explicitly moves it.

enforceKeyFocus (class EnforceKeyFocus)
: If True (the default), the 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 scrollbar that can be operated without setting the focus to that client.) If the resource explicitly set to globally active windows.

iconAutoPlace (class IconAutoPlace)
: Specifies 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. If True (the default), icons are arranged in a particular area of the screen, determined by the iconPlacement resource. If False, an icon is placed at the location of the window when it is iconified.

iconClick (class IconClick)
: If True (the default), the **Window Menu** is displayed when the pointer is clicked on an icon.

interactivePlacement (class InteractivePlacement)
: If True, specifies that new windows are to be placed interactively on the screen using the pointer. When a client is run, the pointer shape changes to an upper-left corner cursor; move the pointer to the location you want the window to appear and click the first button; the window is displayed in the selected location. If False (the default), windows are placed according to the initial window configuration attributes.

keyboardFocusPolicy (class KeyboardFocusPolicy)

If explicit focus is specified (the default), placing the pointer on a window (including the frame) or icon and pressing the first pointer button focuses keyboard input on the client. If pointer is specified, the keyboard input focus is directed to the client window on which the pointer rests (the pointer can also rest on the frame).

lowerOnIconify (class LowerOnIconify)

If True (the default), a window's icon is placed on the bottom of the stack when the window is iconified. If False, the icon is placed in the stacking order at the same place as its associated window.

moveThreshold (class MoveThreshold)

Controls the sensitivity of dragging operations, such as those used to move windows and icons on the display. Takes a value of the number of pixels that the pointing device is moved while a button is held down before the move operation is initiated. The default is 4. This resource helps prevent a window or icon from moving when you click or double click and inadvertently jostle the pointer while a button is down.

multiScreen (class MultiScreen)

If False (the default), *mwm* manages only a single screen. If True, *mwm* manages all screens on the display.

passButtons (class PassButtons)

Specifies whether button press events are passed to clients after the events are used to invoke a window manager function in the client context. If False (the default), button presses are not passed to the client. If True, button presses are passed to the client. The window manager function is done in either case.

passSelectButton (class PassSelectButton)

Specifies whether select button press events are passed to clients after the events are used to invoke a window manager function in the client context. If True (the default), button presses are passed to the client window. If False, button presses are not passed to the client. The window manager function is done in either case.

positionIsFrame (class PositionIsFrame)

Specifies how *mwm* should interpret window position information from the WM_NORMAL_HINTS property and from configuration requests. If True (the default), the information is interpreted as the position of the *mwm* client window frame. If False, it is interpreted as being the position of the client area of the window.

positionOnScreen (class PositionOnScreen)

If True (the default), specifies that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen. If a window is larger than the size of the screen, at least the upper-left corner of the window is placed is on the screen. If False, windows are placed in the requested position even if totally off the screen.

quitTimeout (class QuitTimeout)

Specifies the amount of time (in milliseconds) that *mwm* will wait for a client to update the WM_COMMAND property after *mwm* has sent the WM_SAVE_-YOURSELF message. The default is 1000. (See the f.kill function for additional information.)

raiseKeyFocus (class RaiseKeyFocus)

If True, specifies that a window raised by means of the f.normalize_-and_raise function also receives the input focus. This function is available only when the keyboardFocusPolicy is explicit. The default is False.

screens (class Screens)

Assigns resource names to the screens *mwm* is managing. 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.

showFeedback (class ShowFeedback)

Specifies whether *mwm* feedback windows and confirmation dialog boxes are displayed. (Feedback windows are used to display: window coordinates during interactive placement and subsequent moves; and dimensions during resize operations. A typical confirmation dialog is the window displayed to allow the user to allow or cancel a window manager restart operation.)

showFeedback accepts a list of options, each of which corresponds to the type of feedback given in a particular circumstance. Depending on the syntax in which the options are entered, you can either enable or disable a feedback option (as explained later).

The possible feedback options are: all, which specifies that *mwm* show all types of feedback (this is the default); behavior, which specifies that feedback is displayed to confirm a behavior switch; kill, which specifies that feedback is displayed on receipt of a KILL signal; move, which specifies that a box containing the coordinates of a window or icon is displayed during a move operation; place-

ment, which specifies that a box containing the position and size of a window is displayed during initial (interactive) placement; quit, which specifies that a dialog box is displayed so that the user can confirm (or cancel) the procedure to quit *mwm*; resize, which specifies that a box containing the window size is displayed during a resize operation; restart, which displays a dialog box so that the user can confirm (or cancel) an *mwm* restart procedure; the none option specifies that no feedback is shown.

To limit feedback to particular cases, you can use one of two syntaxes: with the first syntax, you disable feedback in specified cases (all other default feedback is still used); with the second syntax, you enable feedback only in specified cases. You supply this resource with a list of options to be enabled or disabled. If the first item is preceded by a minus sign, feedback is disabled for all options in the list. If the first item is preceded by a plus sign (or no sign is used), feedback is enabled only for options in the list.

startupKeyFocus (class StartupKeyFocus)
: If True (the default), the input focus is transferred to a window when the window is mapped (i.e., initially managed by the window manager). This function is available only when keyboardFocusPolicy is explicit. startupKeyFocus and autoKeyFocus should both be True to work properly with tear-off menus.

wMenuButtonClick (class WMenuButtonClick)
: If True (the default), a pointer button click on the window menu button displays the **Window Menu** and leaves it displayed.

wMenuButtonClick2 (class WMenuButtonClick2)
: If True, double clicking on the window menu button removes the client window, which means that f.kill is invoked.

Screen-Specific Resources

Some *mwm* resources can be applied on a per-screen basis. To specify a screen-specific resource, use the following syntax:

Mwm**screen_number*resource_name: resource_value*

Screen-specific specifications take precedence over specifications for all screens. Screen-specific resources can be specified for all screens using the following syntax:

Mwm**resource_name: resource_value*

buttonBindings (class ButtonBindings)

Identifies the set of button bindings to be used for window management functions; must correspond to a set of button bindings specified in the *mwm* startup file. Button bindings specified in the startup file are merged with built-in default bindings. The default is DefaultButtonBindings.

cleanText (class CleanText)

Specifies whether text that appears in *mwm* title and feedback windows is displayed over the existing background pattern. If True (the default), text is drawn with a clear (no stipple) background. (Only the stippling in the area immediately around the text is cleared.) This enhances readability, especially on monochrome systems where a backgroundPixmap is specified. If False, text is drawn on top of the existing background.

fadeNormalIcon (class FadeNormalIcon)

If True, an icon is greyed out when it has been normalized. The default is False.

feedbackGeometry (class FeedbackGeometry)

Specifies the position of the small, rectangular feedback box that displays coordinate and size information during move and resize operations. By default, the feedback window appears in the center of the screen. This resource takes the argument:

[=]±*xoffset*±*yoffset*

With the exception of the optional leading equal sign, this string is identical to the second portion of the standard geometry string. Note that feedbackGeometry allows you to specify location only. The size of the feedback window is not configurable using this resource. Available as of *mwm* version 1.2 and later.

frameBorderWidth (class FrameBorderWidth)

Specifies the width in pixels of a window frame border, without resize handles. (The border width includes the three-dimensional shadows.) The default is determined according to screen specifics.

frameStyle

In Motif 2.0 and later, specifies the frame appearance of decoration windows and borders: the value WmRECESSED makes the window appear recessed into the border, the value WmSLAB gives a flat window and border.

iconBoxGeometry (class IconBoxGeometry)

Specifies the initial position and size of the icon box. Takes as its argument the standard geometry string:

*width*x*height*±*xoff*±*yoff*

where *width* and *height* are measured in icons. The default geometry string is 6x1+0-0, which places an icon box six icons wide by one icon high in the lower-left corner of the screen.

You can omit either the dimensions or the x and y offsets from the geometry string and the defaults apply. If the offsets are not provided, the iconPlacement resource is used to determine the initial placement.

The actual screen size of the icon box depends on the iconImageMaximum and iconDecoration resources, which specify icon size and padding. The default value for size is ($6 \times$ *icon_width* + padding) wide by ($1 \times$ *icon_height* + padding) high.

iconBoxName (class IconBoxName)

Specifies the name under which icon box resources are to be found. The default is iconbox.

iconBoxSBDisplayPolicy (class IconBoxSBDisplayPolicy)

Specifies what scrollbars are displayed in the icon box. The resource has three possible values: all, vertical, and horizontal. If all is specified (the default), both vertical and horizontal scrollbars are displayed at all times. vertical specifies that a single vertical scrollbar is displayed and sets the orientation of the icon box to horizontal, regardless of the iconBoxGeometry specification. horizontal specifies that a single horizontal scrollbar is displayed in the icon box and sets the orientation of the icon box to vertical, regardless of the iconBoxGeometry specification.

iconBoxTitle (class IconBoxTitle)

Specifies the name to be used in the title area of the icon box. The default is Icons.

iconDecoration (class IconDecoration)

Specifies how much icon decoration is used. The resource value takes four possible values (multiple values can also be supplied): label, which specifies that only the label is displayed; image, which specifies that only the image is displayed; and activelabel, which specifies that a label (not truncated to the width of the icon) is used when the icon has the focus.

The default decoration for icons in an icon box is label image, which specifies that both the label and image parts are displayed. The default decoration for individual icons on the screen proper is activelabel label image.

iconImageMaximum (class IconImageMaximum)

Specifies the maximum size of the icon image. Takes a value of *widthxheight* (e.g., 80×80). The maximum size supported is 128×128. The default is 50×50.

iconImageMinimum (class IconImageMinimum)

Specifies the minimum size of the icon image. Takes a value of *widthxheight* (e.g., 36×48). The minimum size supported is 16×16 (which is also the default).

iconPlacement (class IconPlacement)

Specifies an icon placement scheme. Note that this resource is only useful when useIconBox is False (the default). The iconPlacement resource takes a value of the syntax:

primary_layout secondary_layout [tight]

There are four possible layout policies. top specifies that icons are placed from the top of the screen to the bottom, bottom specifies a bottom-to-top arrangement, left specifies that icons are placed from the left to the right, and right specifies a right-to-left arrangement. The optional argument tight specifies that there is no space between icons.

The *primary_layout* specifies whether icons are placed in a row or a column and the direction of placement. The *secondary_layout* specifies where to place new rows or columns. For example, a value of top right specifies that icons should be placed from top to bottom on the screen and that columns should be added from right to left on the screen.

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 default placement is left bottom (i.e., icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows are added from the bottom of the screen to the top of the screen).

iconPlacementMargin (class IconPlacementMargin)
: Sets the distance from the edge of the screen at which icons are placed. The value should be greater than or equal to 0. A default value is used if an invalid distance is specified. The default value is equal to the space between icons as they are placed on the screen, which is based on maximizing the number of icons in each row and column.

keyBindings (class KeyBindings)
: Identifies the set of key bindings to be used for window management functions; must correspond to a set of key bindings specified in the *mwm* startup file. Note that key bindings specified in the startup file replace the built-in default bindings. The default is DefaultKeyBindings.

limitResize (class LimitResize)
: If True (the default), the user is not allowed to resize a window to greater than the maximum size.

maximumMaximumSize (class MaximumMaximumSize)
: Specifies the maximum size of a client window (as set by the user or client). Takes a value of *widthxheight* (e.g., 1024x1024) where *width* and *height* are in pixels. The default is twice the screen width and height.

moveOpaque (class MoveOpaque)
: If False (the default), when you move a window or icon, its outline is moved before it is redrawn in the new location. If True, the actual (and thus, opaque) window or icon is moved. Available as of *mwm* version 1.2 and later.

resizeBorderWidth (class ResizeBorderWidth)
: Specifies the width in pixels of a window frame border, with resize handles. (The border width includes the three-dimensional shadows.) The default is determined according to screen specifics.

resizeCursors (class ResizeCursors)
: If True (the default), the resize cursors are always displayed when the pointer is in the window resize border.

transientDecoration (class TransientDecoration)
: Specifies the amount of decoration *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 is menu title, which specifies that transient windows have resize borders and a title bar with a window menu button. If the client application also specifies which decorations the window manager should provide, *mwm* uses only those features that both the client and the transientDecoration resource specify.

transientFunctions (class TransientFunctions)
: Specifies 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 is -minimize maximize. If the client application also specifies which window management functions should be applicable, *mwm* provides only those functions that both the client and the transientFunctions resource specify.

useIconBox (class UseIconBox)
: If True, icons are placed in an icon box. By default, the individual icons are placed on the root window.

Client-Specific Resources

Some *mwm* resources can be set to apply to certain client applications or classes of applications. To specify a client-specific resource, use the following syntax:

Mwm**client_name*resource_name: resource_value*

Client-specific specifications take precedence over specifications for all clients. Client- specific resources can be specified for all clients using the following syntax:

Mwm**resource_name: resource_value*

The class name defaults can be used to specify resources for clients that have an unknown name and class.

The following client-specific resources can be specified:

clientDecoration (class ClientDecoration)
: Specifies the amount of window frame decoration. The default frame is composed of several component parts: the title bar, resize handles, border, and the minimize, maximize, and window menu buttons. You can limit the frame decoration for a client using the clientDecoration resource.

  clientDecoration accepts a list of options, each of which corresponds to a part of the client frame. The options are: maximize,

minimize, menu, border, title, resize, all, which encompasses all decorations previously listed, and none, which specifies that no decorations are used.

Some decorations require the presence of others; if you specify such a decoration, any decorations required with it are used automatically. Specifically, if any of the command buttons are specified, a title bar is also used; if resize handles or a title bar is specified, a border is also used.

By default, a client window has all decoration. To specify only certain parts of the default frame, you can use one of two syntaxes: with the first syntax, you disable certain frame features; with the second syntax, you enable only certain features. You supply clientDecoration with a list of options to be enabled or disabled. If the first item is preceded by a minus sign, the features in the list are disabled. If the first item is preceded by a plus sign (or no sign is used), only those features listed are enabled.

clientFunctions (class ClientFunctions)

Specifies whether certain *mwm* functions can be invoked on a client window. The only functions that can be controlled are those that are executable using the pointer on the default window frame. clientFunctions accepts a list of options, each of which corresponds to an *mwm* function. The options are: resize, move, minimize, maximize, close, all, which encompasses all of the previously listed functions, and none, which specifies that no default functions are allowed.

By default, a client recognizes all functions. To limit the functions a client recognizes, you can use one of two syntaxes: with the first syntax, you disallow certain functions; with the second syntax, you allow only certain functions. You supply clientFunctions with a list of options (corresponding to functions) to be allowed or disallowed. If the first item is preceded by a minus sign, the functions in the list are disallowed. If the first option is preceded by a plus sign (or no sign is used), only those functions listed are allowed.
A less than obvious repercussion of disallowing a particular function is that the client window frame is also altered to prevent your invoking that function. For instance, if you disallow the f.resize function for a client, the client's frame does not include resize borders. In addition, the **Size** item on the **Window Menu**, which invokes the f.resize function, no longer appears on the menu.

If the client application also specifies which window management functions should be applicable, *mwm* provides only those functions that both the client and the clientFunctions resource specify.

focusAutoRaise (class FocusAutoRaise)
: If True, a window is raised when it receives the input focus. Otherwise, directing focus to a window does not affect the stacking order. The default depends on the value assigned to the keyboardFocusPolicy resource. If the keyboardFocusPolicy is explicit, the default for focusAutoRaise is True. If the keyboardFocusPolicy is pointer, the default for focusAutoRaise is False.

iconImage (class IconImage)
: Specifies the pathname of a bitmap file to be used as an icon image for a client. The default is to display an icon image supplied by the window manager. If the useClientIcon resource is set to True, an icon image supplied by the client takes precedence over an icon image supplied by the user.

iconImageBackground (class Background)
: Specifies the background color of the icon image. The default is the color specified by Mwm*background or Mwm*icon*background.

iconImageBottomShadowColor (class Foreground)
: Specifies the bottom shadow color of the icon image. The default is the color specified by Mwm*icon*bottomShadowColor.

iconImageBottomShadowPixmap (class BottomShadowPixmap)
: Specifies the bottom shadow pixmap of the icon image. The default is the pixmap specified by Mwm*icon*bottomShadowPixmap.

iconImageForeground (class Foreground)
: Specifies the foreground color of the icon image. The default varies based on the icon background.

iconImageTopShadowColor (class Background)
: Specifies the top shadow color of the icon image. The default is the color specified by Mwm*icon*topShadowColor.

iconImageTopShadowPixmap (class TopShadowPixmap)
: Specifies the top shadow Pixmap of the icon image. The default is the pixmap specified by Mwm*icon*topShadowPixmap.

matteBackground (class Background)
Specifies the background color of the matte. The default is the color specified by Mwm*background or Mwm*client*background. This resource is only relevant if matteWidth is positive.

matteBottomShadowColor (class Foreground)
Specifies the bottom shadow color of the matte. The default is the color specified by Mwm*bottomShadowColor or Mwm*client*bottomShadowColor. This resource is only relevant if matteWidth is positive.

matteBottomShadowPixmap (class BottomShadowPixmap)
Specifies the bottom shadow pixmap of the matte. The default is the pixmap specified by Mwm*bottomShadowPixmap or Mwm*client*bottomShadowPixmap. This resource is only relevant if matteWidth is positive.

matteForeground (class Foreground)
Specifies the foreground color of the matte. The default is the color specified by Mwm*foreground or Mwm*client*foreground. This resource is only relevant if matteWidth is positive.

matteTopShadowColor (class Background)
Specifies the top shadow color of the matte. The default is the color specified by Mwm*topShadowColor or Mwm*client*topShadowColor. This resource is only relevant if matteWidth is positive.

matteTopShadowPixmap (class TopShadowPixmap)
Specifies the top shadow pixmap of the matte. The default is the pixmap specified by Mwm*topShadowPixmap or Mwm*client*topShadowPixmap. This resource is only relevant if matteWidth is positive.

matteWidth (class MatteWidth)
Specifies the width of the matte. The default is 0, which means no matte is used.

maximumClientSize (class MaximumClientSize)
Specifies how a window is to be maximized, either to a specific size (*widthxheight*), or as much as possible in a certain direction (vertical or horizontal). If the value is of the form *widthxheight*, the width and height are interpreted in the units used by the client. For example, *xterm* measures width and height in font characters and lines.

If maximumClientSize is not specified, and the WM_NORMAL_HINTS property is set, the default is obtained

from it. If WM_NORMAL_HINTS is not set, the default is the size (including borders) that fills the screen. *mwm* also uses maximum-MaximumSize to constrain the value in this case.

useClientIcon (class UseClientIcon)
: If True, an icon image supplied by the client takes precedence over an icon image supplied by the user. The default is False.

usePPosition (class UsePPosition)
: Specifies whether *mwm* uses initial coordinates supplied by the client application. If True, *mwm* always uses the program specified position. If False, *mwm* never uses the program specified position. The default is nonzero, which means that *mwm* will use any program specified position except 0,0. Available as of *mwm* version 1.2 and later.

windowMenu (class WindowMenu)
: Specifies a name for the **Window Menu** (which must be defined in the startup file). The default is DefaultWindowMenu.

Functions

mwm supports a number of functions that can be bound to different key and button combinations and assigned to menus in the *mwm* resource description file (*system.mwmrc* or *.mwmrc*). Most window manager functions can be used in key bindings, button bindings, and menus. The function descriptions below note any exceptions to this policy. Most window manager functions can also be specified for three contexts: root, window, and icon. The root context means that the function is applied to the root window, window means that the function is applied to the selected client window, and icon means that the function is applied to the selected icon. The function descriptions below note any functions that cannot be used in all three contexts.

When a function is specified with the context icon | window and you invoke the function from the icon box, the function applies to the icon box itself, rather than to any of the icons it contains.

A function is treated as f.nop if it is not a valid function name, if it is specified inappropriately, or if it is invoked in an invalid way.

mwm recognizes the following functions:

f.beep
: Causes a beep from the keyboard.

f.circle_down [icon | window]

Causes the window or icon on the top of the stack to be lowered to the bottom of the stack. If the icon argument is specified, the function applies only to icons. If the window argument is specified, the function applies only to windows.

f.circle_up [icon | window]

Causes the window or icon on the bottom of the stack to be raised to the top. If the icon argument is specified, the function applies only to icons. If the window argument is specified, the function applies only to windows.

f.exec[*command*]
![*command*]

Executes *command* using the shell specified by the MWMSHELL environment variable. If MWMSHELL is not set, the command is executed using the shell specified by the SHELL environment variable; otherwise, the command is executed using */bin/sh*.

f.focus_color

Sets the colormap focus to a client window. If this function is invoked in the root context, the default colormap (specified by X for the screen where *mwm* is running) is installed and there is no specific client window colormap focus. For the f.focus_color function to work, the colormapFocusPolicy should be specified as explicit; otherwise the function is treated as f.nop.

f.focus_key

Sets the input focus to a window or icon. For the f.focus_key function to work, the keyboardFocusPolicy should be specified as explicit. If keyboardFocusPolicy is not explicit or if the function is invoked in the root context, it is treated as f.nop.

f.kill

Terminates a client. It sends the WM_DELETE_WINDOW message to the selected window if the client application has requested it through the WM_PROTOCOLS property. The application is supposed to respond to the message by removing the indicated window. If the WM_SAVE_YOURSELF protocol is set up and the WM_DELETE_WINDOW protocol is not, the client is sent a message that indicates 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, the f.kill function causes a client's X connection to be terminated.

f.lower [-*client* | within | *freeFamily*]

Without arguments, lowers a window or icon to the bottom of the stack. By default, the context in which the function is invoked indicates to the window or icon to lower. If an application window has one or more transient windows (e.g., dialog boxes), the transient windows are lowered with the parent (within the global stack) and remain on top of it. If the -*client* argument is specified, the function is invoked on the named client. client must be the instance or class name of a program. The within argument is used to lower a transient window within the application's local window hierarchy; all transients remain above the parent window and that window remains in the same position in the global window stack. In practice, this function is only useful when there are two or more transient windows and you want to shuffle them. The freeFamily argument is used to lower a transient below its parent in the application's local window hierarchy. Again, the parent is not moved in the global window stack. However, if you use this function on the parent, the entire family stack is lowered within the global stack.

f.maximize

Causes a window to be redisplayed at its maximum size. This function cannot be invoked in the context root or on a window that is already maximized.

f.menu *menu_name*

Associates a cascading menu with a menu item or associates a menu with a button or key binding. The *menu_name* argument specifies the menu.

f.minimize

Causes a window to be minimized (i.e., iconified). When no icon box is being used, icons are placed on the bottom of the stack, which is generally in the lower-left corner of the screen. If an icon box is being used, icons are placed inside the box. This function cannot be invoked in the context root or on an iconified window.

f.move

Allows you to move a window interactively, using the pointer.

f.next_cmap

Installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [icon | window | transient]

Without any arguments, this function advances the input focus to the next window or icon in the stack. You can specify icon or window to make the function apply only to icons or windows, respectively. Generally, the focus is moved to windows that do not have an associated secondary window that is application modal. If the transient argument is specified, transient windows are also traversed. Otherwise, if only window is specified, focus is moved to the last window in a transient group to have the focus. For this function to work, keyboardFocusPolicy must be explicit; otherwise, the function is treated as f.nop.

f.nop

Specifies no operation.

f.normalize

Causes a client window to be displayed at its normal size. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.normalize_and_raise

Causes the client window to be displayed at its normal size and raised to the top of the stack. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.pack_icons

Rearranges icons in an optimal fashion based on the layout policy being used, either on the root window or in the icon box.

f.pass_keys

Toggles processing of key bindings for window manager functions. When key binding processing is disabled, all keys are passed to the window with the keyboard input focus and no window manager functions are invoked. If the f.pass_keys function is set up to be invoked with a key binding, the binding can be used to toggle key binding processing.

f.post_wmenu

Displays the **Window Menu**. If a key is used to display the menu and a window menu button is present, the upper-left corner of the menu is placed at the lower-left corner of the command button. If no window menu button is present, the menu is placed in the upper-left corner of the 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*]

Without any arguments, this function moves the input focus to the previous window or icon in the stack. You can specify *icon* or *window* to make the function apply only to icons or windows, respectively. Generally, the focus is moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, transient windows are also traversed. Otherwise, if only *window* is specified, focus is moved to the last window in a transient group to have the focus. For this function to work, keyboardFocusPolicy must be explicit; otherwise, the function is treated as f.nop.

f.quit_mwm

Stops the *mwm* window manager. Note that this function does not stop the X server. This function cannot be invoked from a non-root menu.

f.raise *[-client* | *within* | *freeFamily*]

Without arguments, raises a window or icon to the top of the stack. By default, the context in which the function is invoked indicates the window or icon to raise. If an application window has one or more transient windows (e.g., dialog boxes), the transient windows are raised with the parent (within the global stack) and remain on top of it. If the *-client* argument is specified, the function is invoked on the named client. client must be the instance or class name of a program. The *within* argument is used to raise a transient window within the application's local window hierarchy; all transients remain above the parent window and that window remains in the same position in the global window stack. In practice, this function is only useful when there are two or more transient windows and you want to shuffle them.

The *freeFamily* argument raises a transient to the top of the application's local window hierarchy. The parent window is also raised to the top of the global stack.

f.raise_lower [*within* | *freeFamily*]

Raises a primary application window to the top of the stack or lowers a window to the bottom of the stack, as appropriate to the context. The within argument is intended to raise a transient window

within the application's local window hierarchy. All transients remain above the parent window and the parent window should also remain in the same position in the global window stack. If the transient is not obscured by another window in the local stack, the transient window is lowered within the family. The preceding paragraph describes how within *should* work. However, we have found that the parent window does not always remain in the same position in the global window stack. The freeFamily argument raises a transient to the top of the family stack and also raises the parent window to the top of the global stack. If the transient is not obscured by another window, this function lowers the transient to the bottom of the family stack and lowers the family in the global stack.

f.refresh

Redraws all windows.

f.refresh_win

Redraws a single window.

f.resize

Allows you to resize a window interactively, using the pointer.

f.restart

Restarts the *mwm* window manager. The function causes the current *mwm* process to be stopped and a new *mwm* process to be started. It cannot be invoked from a non-root menu.

f.restore

Causes the client window to be displayed at its previous size. If invoked on an icon, f.restore causes the icon to be converted back to a window at its previous size. Thus, if the window was maximized, it is restored to this state. If the window was previously at its normal size, it is restored to this state. If invoked on a maximized window, the window is restored to its normal size. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.restore_and_raise

Causes the client window to be displayed at its previous size and raised to the top of the stack. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.screen [*next* | *prev* | *back* | *screen_number*]

Causes the pointer to be warped to another screen, which is determined by one of four mutually exclusive parameters. The *next* argument means skip to the next managed screen, *prev* means skip back to the previous managed screen, back means skip to the last screen visited, and *screen_number* specifies a particular screen. Screens are numbered beginning at 0.

f.send_msg *message_number*

Sends a message of the type _MOTIF_WM_MESSAGES to a client; the message type is indicated by the *message_number* argument. The message is sent only if the client's _MOTIF_WM_MESSAGES property includes *message_number*. If a menu item is set up to invoke f.send_msg and the *message_number* is not included in the client's _MOTIF_WM_MESSAGES property, the menu item label is greyed out, which indicates that it is not available for selection.

f.separator

Creates a divider line in a menu. Any associated label is ignored.

f.set_behavior

Restarts *mwm*, toggling between the default behavior for the particular system and the user's custom environment. In any case, a dialog box asks the user to confirm or cancel the action. By default this function is invoked using the following key sequence: Shift Ctrl Meta !.

f.title

Specifies the title of a menu. The title string is separated from the menu items by a double divider line.

Event Specification

In order to specify button bindings, key bindings, and menu accelerators, you need to be able to specify events in the *mwm* resource description file. Use the following syntax to specify button events for button bindings:

[*modifier_key...*]<*button_event*>

The acceptable values for *modifier_key* are: Ctrl, Shift, Alt, Meta, Lock, Mod1, Mod2, Mod3, Mod4, and Mod5. *mwm* considers Alt and Meta to be equivalent.

The acceptable values for *button_event* are:

Btn1Down	Btn2Down	Btn3Down	Btn4Down	Btn5Down
Btn1Up	Btn2Up	Btn3Up	Btn4Up	Btn5Up
Btn1Click	Btn2Click	Btn3Click	Btn4Click	Btn5Click
Btn1Click2	Btn2Click2	Btn3Click2	Btn4Click2	Btn5Click2

Use the following syntax to specify key events for key bindings and menu accelerators:

[*modifier_key...*]<Key>*key_name*

Any X11 keysym name is an acceptable value for *key_name*.

Button Bindings

The buttonBindings resource specifies the name of a set of button bindings that control mouse behavior in mwm. You can create your own set of button bindings or use one of the sets defined in *system.mwmrc:* DefaultButtonBindings, ExplicitButtonBindings, or PointerButtonBindings. Use the following syntax to specify a set of button bindings:

```
Buttons button_set_name
{
    button context function
    button context function
    ...
    button context function
}
```

The *context* specifies where the pointer must be located for the button binding to work. The context is also used for window manager functions that are context-sensitive. The valid contexts for button bindings are root, window, icon, title, border, frame, and app. The title context refers to the title area of the frame. border refers to the frame exclusive of the title bar. frame refers to the entire frame. The app context refers to the application window proper. The window context includes the application window and the frame. A context specification can include multiple contexts; use a vertical bar (|) to separate multiple context values.

Key Bindings

The keyBindings resource specifies the name of a set of key bindings that control keyboard behavior in *mwm*. You can create your own set of key bindings or use the default key bindings, DefaultKeyBindings, defined in *system.mwmrc*. Use the following syntax to specify a set of key bindings:

```
Keys key_set_name
{
    key context function
    key context function
    ...
    key context function
}
```

The *context* specifies where the keyboard focus must be for the key binding to work. The context is also used for window manager functions that are context-sensitive. The valid contexts for key bindings are root, window, icon, title, border, frame, and app. The title, border, frame, and app contexts are all equivalent to window. A context specification can include multiple contexts; use a vertical bar (|) to separate multiple context values.

Menus

The window manager functions f.post_wmenu and f.menu post menus. These functions both take the name of a menu to post. You can create your own menus or use the default menus defined in *system.mwmrc*: DefaultRootMenu and DefaultWindowMenu. Use the following syntax to specify a menu:

```
Menu menu_name
{
    label [mnemonic] [accelerator] function
    label [mnemonic] [accelerator] function
    ...
    label [mnemonic] [accelerator] function
}
```

Each line in a menu specification indicates the label for the menu item, optional keyboard mnemonics and accelerators, and the window manager function that is performed. *label* can be a string or a bitmap file. If the string contains multiple words, it must be enclosed in quotation marks. A bitmap file specification is preceded by an at sign (*@*). A mnemonic is specified as _character. An accelerator specification uses the key event specification syntax.

The context of a window manager function that is activated from a menu is based on how the menu is posted. If it is posted from a button binding, the context of the menu is the context of the button binding. If it is posted from a key binding, the context of the menu is based on the location of the keyboard focus.

Environment

mwm uses the following environment variables:

HOME
: The user's home directory.

LANG
: The language to be used for the *mwm* message catalog and the *mwm* startup file.

XBMLANGPATH
: Used to search for bitmap files.

XFILESEARCHPATH
: Used to determine the location of system-wide class resource files. If the LANG variable is set, the *$LANG* subdirectory is also searched.

XUSERFILESEARCHPATH, XAPPLRESDIR
: Used to determine the location of user-specific class resource files. If the LANG variable is set, the *$LANG* subdirectory is also searched.

MWMSHELL, SHELL
: MWMSHELL specifies the shell to use when executing a command supplied as an argument to the f.exec function. If MWMSHELL is not set, SHELL is used.

Files

/usr/lib/X11/$LANG/system.mwmrc
/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/Mwm
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
$HOME/.motifbind

See Also

`XmIsMotifWMRunning`(1), `XmInstallImage`(1), `VendorShell`(2), `xmbind`(3)

Name

uil – the User Interface Language (UIL) compiler.

Syntax

uil [*options*] *file*

Description

The *uil* command invokes the User Interface Language (UIL) compiler. If the file does not contain any errors, the compiler generates a User Interface Description (UID) file that contains a compiled form of the input file. UIL is a specification language that can be used to describe the initial state of a user interface that uses the OSF/Motif widget set, as well as other widgets. The user interface for an application is created at run-time using the Motif Resource Manager (Mrm) library; the interface is based on compiled interface descriptions stored in one or more UID files.

Options

-I*pathname*

Specifies a search path for include files. By default, the current directory and */usr/include* are searched. Path names may be relative or absolute. The paths specified with this option are searched in order after the current directory and before */usr/include*.

-m

When specified with the -v option, the UIL compiler includes machine code in the listing file. The machine code provides binary and text descriptions of the data that is stored in the UID file. This option is useful for determining exactly how the compiler interprets a particular statement and how much storage is used for the variables, declarations, and assignments.

-o *ofile*

Specifies the name of the UID file to output. The default filename is *a.uid*. The customary suffix for UID files is *.uid*.

-s

Specifies that the UIL compiler set the locale before compiling any files. Setting the locale determines the behavior of locale-dependent routines like character string operations. Although setting the locale is an implementation-dependent operation, on ANSI-C-based systems, the locale is set with the call:

```
setlocale (LC_ALL, "")
```

See the setlocale() man page on your system for more information.

-v *lfile*

Directs the UIL compiler to produce a listing of the compilation. The file indicates the name of the output file. If this option is not specified, the compiler does not generate a listing. On UNIX systems, a filename of */dev/tty* usually causes the listing to be output on the terminal where *uil* was invoked.

-w

Directs the compiler to suppress warning and informational messages and to print only error messages. The default behavior is to print error, warning and informational messages.

-wmd *wfile*

Specifies a compiled Widget Meta-Language (WML) description file that is loaded in place of the default WML description. The default WML description file contains a description of all of the Motif widgets. This option is normally used to debug a WML description file without rebuilding the UIL compiler.

Environment

The LANG environment variable affects the way that the UIL compiler parses and generates compound strings, fonts, fontsets, and font tables (font lists). The exact effect is described by the UIL reference pages for these types.

Example

```
% uil -o myfile.uid -v /dev/tty myfile.uil
% uil -I/project/include/uil -o mainui.uid mainui.uil
```

Bugs

If the LANG environment variable is set to an invalid value and the -s option is specified, the UIL compiler crashes.

See Also

`Uil`*(7)*.

Name

xmbind – configure virtual key bindings.

Syntax

xmbind [*options*] [*file*]

Availability

Motif 1.2 and later.

Description

The *xmbind* command configures the virtual key bindings for Motif applications. Since this action is performed by *mwm* on startup, *xmbind* is only needed when *mwm* is not being used or when a user wants to change the bindings without restarting *mwm*.

When a file is specified, its contents are used for the virtual bindings. Otherwise, *xmbind* uses the *.motifbind* file in the user's home directory. A sample specification is shown below:

```
osfBackSpace:   <Key>BackSpace
osfInsert:      <Key>InsertChar
osfDelete:      <Key>DeleteChar
```

If *xmbind* cannot find the *.motifbind* file, it loads the default virtual bindings for the server. *xmbind* searches for a vendor-specific set of bindings located using the file *xmbind.alias*. If this file exists in the user's home directory, it is searched for a pathname associated with the vendor string or the vendor string and vendor release. If the search is unsuccessful, Motif continues looking for *xmbind.alias* in the directory specified by XMBINDDIR or in */usr/lib/Xm/bindings* if the variable is not set. If this file exists, it is searched for a pathname as before. If either search locates a pathname and the file exists, the bindings in that file are used. An *xmbind.alias* file contains lines of the following form:

"*vendor_string*[*vendor_release*]"*bindings_file*

If *xmbind* still has not located any bindings, it loads fixed fallback default bindings.

The Motif toolkit uses a mechanism called *virtual bindings* to map one set of keysyms to another set. This mapping permits widgets and applications to use one set of keysyms in translation tables; applications and users can then customize the keysyms used in the translations based on the particular keyboard that is being used. Keysyms that can be used in this way are called *osf keysyms*. Motif maintains a mapping between the osf keysyms and the actual keysyms that represent keys on a particular keyboard. See the Introduction to Section 2, Motif and Xt Widget Classes, for more information about virtual bindings.

Options

-display[*host*]:*server*[.*screen*]

Specifies the name of the display on which to run *xmbind. host* is the hostname of the physical display, *server* specifies the server number, and *screen* specifies the screen number. Either or both of the *host* and *screen* elements can be omitted. If *host* is omitted, the local display is assumed. If *screen* is omitted, screen 0 is assumed (and the period is unnecessary). The colon and (display) *server* are necessary in all cases.

Environment

The XMBINDDIR environment variable affects the way that *xmbind* searches for vendor-specific default virtual bindings.

See Also

`XmTranslateKey`(1).

Section 5 - UIL File Format

This page describes the format and contents of each reference page in Section 5, which covers the UIL file format.

Name

Section – a brief description of the file section.

Syntax

This section describes the syntax for the section of the UIL file. Anything in constant width type should be typed exactly as shown. Items in italics are expressions that should be replaced by actual names and values when you write a UIL file. Anything enclosed in brackets is optional. An ellipsis (...) means that the previous expression can be repeated multiple times and a vertical bar (|) means to select one of a set of choices.

Description

This section provides an overview of the particular section in the UIL module and it explains the syntax that is expected for the section. A UIL source file, also known as a UIL module, describes the user interface for an application. It consists of a module name, optional module settings, optional include directives, zero or more sections that describe all or part of a user interface, and an end module statement. The module specifies the widgets used in the interface, as well as the resources and callbacks of these widgets. UIL gives you the ability to use variables, procedures, lists, and objects to describe the interface.

A major portion of a UIL module is the sections that describe the user interface. They are the value section, for defining and declaring variables; the procedure section, for declaring callback routines; the identifier section, for declaring values registered by the application at run-time; the list section, for defining lists of procedures, resources, callbacks, and widgets; and the object section, for defining the widgets, their resources, and the widget hier-archy.

In this section, we provide reference pages for each section of a UIL source file, as well as for the overall module structure and the include directive. Figure 5-1 shows an example of a UIL module that contains all of these sections.

UIL Syntax

Symbols and identifiers in a UIL module must be separated by whitespace or punctuation characters in order to be recognized by the UIL compiler. Like C, no other restrictions are placed on the formatting of a UIL module, although the maximum line length accepted by the compiler is 132 characters.

Comments in UIL can take two different forms: single-line and multi-line. A single-line comment begins with a exclamation point (!) and continues to the end of the line. A multi-line comment begins with the characters /* and ends with the characters */. Since the UIL compiler suspends normal parsing within comments, they cannot be nested.

Values, identifiers, procedures, lists, and objects are declared or defined with programmer-assigned names. Names can be composed of upper and lowercase characters from A to Z, the digits from 0 to 9, and the underscore (_) and dollar sign ($) characters. Names may be up to 31 characters in length; they cannot begin with a digit. The names option, which is described on the module reference page, affects the case sensitivity of names. The UIL compiler maintains a single name-space for all UIL keywords and programmer-defined names. This means that the name of each value, identifier, procedure, list, and object must be unique.

UIL Keywords

UIL keywords are categorized into reserved and unreserved keywords. Reserved keywords cannot be redefined by the programmer, while unreserved keywords can be used as programmer-defined names. In general, you should avoid redefining unreserved keywords because it can lead to confusion and programming errors. UIL uses the following reserved and unreserved keywords:

```
module StyleMenu
        character_set = iso_latin1
        objects = { XmSeparator = gadget ; }
include file 'appdefs.uih';
value
        tear_off_mode : XmTEAR_OFF_ENABLED;
        a_normal : imported string;
        a_bold : imported string;
        a_italic : imported string;
identifier
        style_widget_list;
procedure
        reset_all (any);
        register_widget (any);
        bold_changed (any);
        italic_changed (any);
list
        style_menu_entries {
                XmPushButton normal;
                XmSeparator { };
                XmToggleButton bold;
                XmToggleButton italic;
        };
        register : callbacks {
                MrmNCreateCallback = procedure register_widget (style_widget_list);
        };
object style_menu : XmPulldownMenu {
        arguments {
        XmNtearOffModel = tear_off_mode;
};
controls style_menu_entries;
}
object normal : XmPushButton {
arguments {
XmNlabelString = a_normal;
};
callbacks {
XmNactivateCallback = procedure reset_all (style_widget_list);
};
};
...

end module;
```

module header

include direction

value section

identifier section

procedure section

list section

object section

end module statement

Figure 5-1: A Sample UIL source file

Type	Reserved Keywords
General	module, end, widget, gadget
Section and list name	arguments, callbacks, controls, identifier, include, list, object, procedure, procedures, value
Storage classes	exported, private
Boolean constants	on, off, true, false

Type	Unreserved Keywords
Resource names	XmNaccelerators, XmNactivateCallback, et al.
Character set names	iso_latin1, iso_greek, et al.
Enumerated values	XmATTACH_FORM, XmSHADOW_ETCHED_IN, et al.
Widget class names	XmPushButton, XmSeparator, et al.
Option names and values	background, case_insensitive, case_sensitive, file, foreground, imported, managed, names, objects, right_to_left, unmanaged, user_defined
Type names	any, argument, asciz_table, asciz_string_table, boolean, character_set, color, color_table, compound_string, compound_string_table, float, font, font_table, fontset, icon, integer, integer_table, keysym, reason, rgb, single_float, string, string_table, translation_table, wide_character, xbitmapfile

Usage

This section provides less formal information about the section: how you might want to use it in a UIL module and things to watch out for.

Example

This section provides examples of the use of the section in a UIL module.

See Also

This section refers you to related functions, UIL file format sections, and UIL data types. The numbers in parentheses following each reference refer to the sections of this book in which they are found.

Name

identifier – run-time variable declaration section.

Syntax

identifier *identifier_name;*
[...]

Description

The *identifier* section contains variable declarations that are registered at run-time by the application with `MrmRegisterNames`() or `MrmRegisterNamesInHierarchy`(). The section begins with the UIL keyword *identifier*, followed by a list of names separated by semicolons.

Usage

A value declared as an identifier can be assigned to a named variable in a value section, it can be passed as the parameter to a callback procedure, or it can be assigned to a resource in the arguments section of an object definition. An identifier value cannot be used in an expression or as part of a complex literal type definition. An identifier value does not have any type associated with it, so it can be passed as a parameter to any callback that can take an argument or it can be assigned to any resource, regardless of the type of parameter or resource expected.

Example

```
...
identifier
    display_name;
    highlight_color;

value
    alias : display_name;

procedure
    highlight (color);

object label : XmLabel {
    arguments {
        XmNlabelString : display_name;
    }
    callbacks {
        XmNfocusInCallback = procedure highlight (highlight_color);
    };
}
...
```

See Also

`MrmRegisterNames`(3), `MrmRegisterNamesInHierarchy`(3), `procedure`(5), `object`(5).

Name

include – include file directive.

Syntax

include file *'file_name';*

Description

The *include* directive tells the UIL compiler to suspend parsing of the current file and switch to the specified file. Parsing of the original file resumes after the end of the included file has been reached. Include directives may be nested, which means that an included file can contain include directives.

If an include file is specified an absolute pathname, which means that it begins with a slash (/), the compiler looks for the file in that specific location. Otherwise, the compiler tries to locate the file by searching in one or more directories. The directory that contains the UIL source file specified on the command line is searched first. (This directory may or may not be the same as the directory the compiler was invoked from.) If the file is not found there, the compiler searches any directories specified on the command line with the -I option in the order that they were specified. Next, the compiler searches the */usr/include* directory. Finally, if the specified file cannot be found, the compiler generates an error message and exits.

When an *include* directive is encountered, the UIL compiler ends the current section. Therefore, an include file must specifically use one of the section name keywords to begin a new section.

Usage

Include files are used to break up modules into more manageable pieces or to provide a common place for definitions and declarations that are shared by several modules. Include files should not be used for defining strings. Strings should be defined in a separate UIL module and loaded at run-time as part of an Mrm hierarchy. The `MrmOpenHierarchyPerDisplay`() reference page explains how different UID files can be loaded based on the LANG environment variable. String declarations, however, are suitable for placement in an include file.

A UIL module can include a maximum of 99 files. This is not a nesting limit, but a limit on the total number of files that can be included. Because the UIL compiler maintains an open file descriptor for each included file, even after it has been included, the limit may be less than 99 due to operating-system imposed limits. If the UIL compiler tries to include a file and the maximum number of open file descriptors have been used, the compiler prints an error and exits. If this situation occurs, you should reduce the number of files included or increase the maximum number of open file descriptors.

If the string containing the include filename is missing a closing quotation mark, or if extraneous characters precede or follow the string, the UIL compiler may generate many strange errors.

Example

From *callbacks.uih*:

```
procedure
    save();
    save_as (string);
    open (string);
    select (integer_table);
    quit();
```

From *edit_window.uil*:

```
module edit_window
! Include callback definitions
include file 'callbacks.uih';
...
end module;
```

See Also

`MrmOpenHierarchyPerDisplay`(3), `uil`(4), `module`(5).

Name

list - list definition section.

Syntax

```
list
    list_name :
        arguments {
            argument_name = value_expression; | arguments arguments_list_name;
            [...]
        }; |
    list_name :
        callbacks {
            reason_name = procedure procedure_name [ ( [ value_expression ] ) ]; |
            reason_name = procedures {
                procedure_name [ ( [ value_expression ] ) ];
                [...]
            }; |
            reason_name = procedures procedure_list_name; | callbacks callbacks_list_name;
            [...]
        }; |
    list_name :
        controls {
            [ managed | unmanaged ] object_class object_name; |
            [ managed | unmanaged ] object_class [ widget | gadget ] { [ attributes ] }; |
            [ managed | unmanaged ] user_defined procedure creation_procedure {[ attributes ] }; |
            auto_created_object_name { [ attributes ] }; |
            controls controls_list_name;
            [...]
        }; |
    list_name :
        procedures {
            procedure_name [ ( [ value_expression ] ) ]; |  procedures procedures_list_name;
            [...]
        };
    [...]
```

Description

The *list* section is used to define lists of resources, callbacks, procedures, or controls that can be used when setting attributes of a widget defined in an object section. Each list definition consists of a list name followed by a colon, a list type, and a list of items of that type separated by semicolons. Each item can be a single item (resource, callback, procedure, or widget) or a list of that type of item. When a list contains another list, the result is the same as if the items in the included list were specified directly in the including list. The storage class of lists is limited to private. Unlike variables and objects, lists cannot be exported, imported, or retrieved by an application at run-time.

The type of a list determines the type and the format of the items it contains. UIL allows the following types of lists: *arguments*, *callbacks*, *controls*, and *procedures*. The format of the items in arguments, callbacks, and controls lists is the same as the format for the corresponding subsection in an object definition. The exact syntax is described in the *object* section reference page.

The procedures list type exists to allow the specification of a list of procedures for a single callback. Each routine in a procedures list is invoked by the specified callback. A procedures list is specified by the symbol *procedures*, followed by a list of procedures declared elsewhere in the module. An individual procedure is specified with the name of the procedure and an argument specification consistent with the routine's declaration. The order in which routines in a procedures list are invoked is not specified by

the Xt Intrinsics. If you need to have several procedures called in a particular order, you should register a single callback that calls the procedures in that order.

Like many values in UIL, a list can be specified directly in the arguments, callbacks, or controls subsection or as a callback procedures list. An inline list is specified by the type of the list, followed by a list of items of that type.

Usage

A list can be used to group collections of resources, callbacks, and widget children that are common to several object definitions. To specify more than one procedure for a single callback, you must use a list. A simple style/behavior hierarchy can be specified by using nested list definitions, as the example below illustrates. If a resource or callback setting occurs more than once in an arguments or callbacks list, the last occurrence has precedence over earlier occurrences. This feature allows you to define a list that includes settings from another list but overrides some of the settings. The UIL compiler issues an informational message about multiple occurrences, but the messages can be turned off by using the -w compiler option.

Example

```
...
! Declare procedures used below.
procedure
    shift();
    floor_it();
    armed();
    ready();

list
    ! Declare some lists to implement widget styles.
    base_style : arguments {
        ! This list contains individual elements only.
        XmNforeground = default_foreground;
        XmNbackground = default_background;
    };
    button_style : arguments {
        ! Include another list in this list.
        arguments base_style;
        XmNfontList = font ('*helvetica-bold-r-normal-*-120-100-100*');
    };

! Declare a list of procedures to be set on an individual callback.
list
    super_button_activate : procedures {
        shift();
        floor_it();
    };
list
    super_button_callbacks : callbacks {
        XmNactivateCallback = procedures super_button_activate;
        ! Set the arm callback to an inline list of procedures.
        XmNarmCallback = procedures {
            armed();
            ready();
        };
    };

object
    super_button : XmPushButton {
```

```
            arguments {
                ! Use arguments in button_style list and add one of our own.
                arguments button_style;
                XmNarmColor = color ('yellow');
            };
            callbacks super_button_callbacks;
        };
...
```

See Also

`object`(5), `procedure`(5).

Name

module – module structure.

Syntax

```
module module_name
    [ names = [ case_insensitive | case_sensitive ] ]
    [ character_set = character_set ]
    [ objects = { widget_name = gadget | widget; [...] } ]
    [ [ include_directive ] | [ value_section ] | [ procedure_section ] |
        [ identifier_section ] | [ list_section ] | [ object_section ] ]
[...]
end module;
```

Description

A UIL module must begin with the keyword module, followed by the name of the module. You may name a module anything you like, as long as it is a valid UIL identifier. The name of a module is defined as a symbol in the compiler's symbol table, and therefore may not be a UIL reserved keyword. In addition, the name of the module cannot be used as the name of an object, variable, identifier, widget, or procedure elsewhere in the module. Option settings for the module are specified following the module statement. There are three different options that you can set: the case sensitivity of the module, the default character set, and the default object variant.

The *names* option specifies the case sensitivity of keywords and symbols in the UIL module. The syntax of this option is the keyword names, followed by either *case_sensitive* or *case_insensitive*. The default is *case_sensitive*, which means that all keywords must be lowercase and the case of symbols is significant. If *case_insensitive* is specified, keywords may be in upper, lower, or mixed case, and all programmer-defined values, procedures, identifiers, and objects are stored as uppercase in the UID file. For example, the three symbols JellyBean, jellybean, and JELLYBEAN are considered different symbols when names are *case_sensitive*, but are considered the same symbol when names are *case_insensitive*. If this option is specified, it must be the first option after the module name and must be specified in lowercase only.

The *character_set* option specifies the character set used for compound_string, font, and fontset values that are not defined with an explicit character set. The syntax of this option is the keyword *character_set*, followed by the name of a built-in character set. (See the character_set reference page for a list of the built-in character sets.) A user-defined character set cannot be used for this option. If this option is not specified, the default character set is determined from the codeset portion of the LANG environment variable if it is set, or XmFALLBACK_CHARSET otherwise. Setting this option overrides the LANG environment variable and turns off localized string parsing specified by the -s compiler option. When the character_set defaults to XmFALLBACK_CHARSET, the UIL compiler may use ISO8859-1 as the character set, even if the value has been changed by the vendor. Therefore, you should specify a character set explicitly instead of relying on the value of XmFALLBACK_CHARSET.

The *objects* option specifies whether the widget or gadget variant is used by default for CascadeButton, Label, PushButton, Separator, and ToggleButton objects. The syntax of the option is the keyword *objects*, followed by a list of object-specific settings. Like all lists in UIL, each setting is separated by a semicolon and enclosed by curly braces. Each object setting is the name of one of the classes listed above, followed by either widget or gadget. The default value for all of the classes is widget. You can override these settings when you define a specific object by adding widget or gadget after the object class name.

UIL also supports a version option setting, which consists of the string version, followed by a string representing the version of the module. This option is obsolete in Motif 1.2 and is retained for backward compatibility. You may encounter this setting in older UIL source files but you should not use it in new ones. The version string is stored in the UID file, but is not used by Mrm and cannot be accessed by the

application. To make a version identifier that is accessible by the application through Mrm, you can store a version value in an exported UIL variable.

The bulk of a UIL module is the sections that describe the user interface, which occur after the module name and optional module settings. Briefly, the sections are the value section, for defining and declaring variables; the procedure section, for declaring callback routines; the identifier section, for declaring values registered by the application at run-time; the list section, for defining lists of procedures, resources, callbacks, or widgets; and the object section, for defining the widgets and their resources, and the widget hierarchy. Each section is described completely in a separate reference page.

Every UIL module must end with the end module statement, which is simply the string end module followed by a semicolon (;). A final newline is required after the end module statement or the UIL compiler generates an error message stating that the line is too long.

Example

```
module print_panel
    names = case_insensitive
    character_set = iso_latin1
    objects = { XmPushButton = gadget; }
! sections
...
end module;
```

See Also

`identifier`(5), `include`(5), `list`(5), `object`(5), `procedure`(5), `value`(5), `character_set`(6), `compound_string`(6), `font`(6), `fontset`(6), `font_table`(6), `string`(6).

Name

object – widget declaration and definition section.

Syntax

```
object object_name : imported object_type; or

object object_name : [ exported | private ]
                         object_type [ widget | gadget ] |
                         user_defined procedure creation_procedure
{ [
    arguments {
        arguments argument_list_name; |
        argument_name = value_expression;
        [...]
    }; |
    callbacks {
        callbacks callback_list_name; |
        reason_name = procedure procedure_name [ ( [ value_expression ] ) ]; |
        reason_name = procedures {
            procedure_name [ ( [ value_expression ] ) ];
            [...]
        }; |
        reason_name = procedures procedures_list_name;
        [...]
    }; |
    controls {
        controls controls_list_name; |
        [ managed | unmanaged ] object_class object_name; |
        [ managed | unmanaged ] object_class [ widget | gadget ] { [ attributes ] }; |
        [ managed | unmanaged ] user_defined procedure creation_procedure { [ attributes ] }; |
        auto_created_object_name { [ attributes ] };
        [...]
    };
    [...] ]
};
```

Description

The *object* section is used to declare or define the objects that compose the user interface of an application. These objects can be either widgets or gadgets and are created at run-time with the routines `MrmFetchWidget`() and `MrmFetchWidgetOverride`(). Both built-in Motif widgets and user-defined widgets can be defined in an object section. In addition, in Motif 2.0 and later, the section is used to define special pseudo-objects which represent the XmRendition, XmRenderTable, XmTab, and XmTabList resource types.

An object declaration informs the UIL compiler about an object that is defined in another UIL module. A declaration consists of the object name followed by a colon, the keyword *imported*, and the type of the imported widget.

An object definition consists of an object name followed by a colon, an optional storage class, a built-in widget class name or used-defined creation procedure, and a list of attributes. An object's attributes may include resource settings, callbacks, and a list of the object's children. The storage class may be either *private* or *exported*. The default storage class is *exported*. Widgets defined as private are not prevented from being retrieved directly with Mrm, but you can still declare widgets as *private* to indicate that they should not be retrieved directly.

When defining an instance of a built-in widget, the name of a Motif class (such as XmPushButton or XmMessageDialog) follows the optional storage class. A class that has both widget and gadget variants can be followed by *widget* or *gadget* to indicate which variant is used. The default variant is widget, unless gadget is specified in the *objects* setting in the UIL module header. For gadget variants, the UIL compiler also allows the name Gadget to be appended directly to the widget class name (as in XmPushButtonGadget). This syntax is inconsistent, however, so you should avoid using it.

When defining an instance of a user-defined widget, the optional storage class is followed by the string *user_defined procedure* and the name of a widget creation procedure. The procedure must be declared in a *procedure* section elsewhere in the module. It must also be registered by the application at run-time with `MrmRegisterClass`() before the widget is retrieved. The C prototype of a creation procedure is described in the `MrmRegisterClass`() manual page in Section 3, *Mrm Functions*.

The remainder of an object definition consists of three optional subsections that define the widget's resources, callbacks, and children. The subsections are enclosed by curly braces, which must be present, even when none of the subsections are specified. Each subsection consists of the name of the subsection followed by the name of a list defined in a list section or a list of items enclosed by curly braces. The arguments subsection specifies resource settings, the *callbacks* subsection specifies callback procedures, and the *controls* section specifies child widgets. In Motif 2.0 and later, the controls section also specifies XmTab and XmRendition constituents of the XmTabList and XmRenderTable pseudo-objects.

Arguments

The *arguments* subsection, if present, specifies one or more resource settings and/or resource lists. A list is specified with the symbol arguments, followed by the name of an arguments list defined elsewhere in the module. Resource settings are of the form *resourceName = value*. The resource name may be built-in or user-defined. (See the argument reference page in Section 6, *UIL Data Types*, for information about creating user-defined resource names.) If the same resource is set more than once in a widget's arguments section, the last occurrence of the setting is used and the UIL compiler issues an informational message.

If the widget instance being defined is from a built-in Motif widget class, the predefined resources set in the arguments section must be valid for the widget class, but any user-defined resource can be set. It can be useful to set user-defined constraint resources on a built-in widget when it is the child of a user-defined constraint widget. If the widget instance being defined is a user-defined widget, any predefined or user-defined resources can be set in its arguments section. You should take care to set resources that are valid for user-defined widgets, as the UIL compiler is unable to detect invalid resources.

The UIL compiler normally verifies that the type of a value matches the type of the resource to which it is assigned. Type checking is not possible, however, when a value is assigned to a user-defined resource of type any, or when a variable declared in an identifier section is assigned to a resource.

The type of a resource and the value assigned to it do not always have to be an exact match. The UIL compiler automatically converts certain values to the appropriate type for a resource. If a type mismatch occurs and a conversion cannot be performed, the compiler generates an error message and a UID file is not generated. The table below summarizes the supported conversions:

Value Type	Can be Assigned To
string	compound_string
asciz_string_table	compound_string_table
icon	pixmap
xbitmapfile	icon
rgb	color
font	font_list

Value Type	Can be Assigned To
fontset	font_list

When a built-in array resource is specified in the arguments subsection, the UIL compiler automatically sets the associated count resource. All but one of the built-in arrays with associated counts are XmStringTable resources; they are listed in the compound_string_table reference page in Section 6, *UIL Data Type*s. The other resource is the Text and TextField resource XmNselectionArray and its associated count resource, XmNselectionArrayCount.

Callbacks

The *callbacks* subsection, if present, specifies one or more callback settings and/or callback lists. A list is specified with the symbol *callbacks*, followed by the name of a callback list defined elsewhere in the module. A callback setting consists of the callback name, such as XmNactivateCallback, followed by an equal sign (=) and either a single procedure name or the name of a list of procedures defined elsewhere in the module. A single procedure is specified by the symbol *procedure* followed by its name, and an argument specification consistent with the procedure's declaration. A list is specified by the symbol *procedures* followed by the name of the list. If the same callback is set more than once in a widget's callbacks section, the last occurrence of the setting is used and the UIL compiler issues an informational message.

A procedure used in the callbacks section must be declared in a procedure section elsewhere in the module. It must also be registered by the application at run-time with `MrmRegisterNames`() or `MrmRegisterNamesInHierarchy`() before any widgets that reference it are created.

If the widget instance being defined is from a built-in Motif widget class, the predefined callbacks set in the callbacks section must be valid for the widget class, but any user-defined callbacks can be set. There should not be any need to set a user-defined callback on a built-in widget, however. If the widget instance being defined is a user-defined widget, any built-in or user-defined callbacks can be set in the *callbacks* section.

In addition to the standard Motif callbacks, Mrm supports the MrmNcreateCallback, which is called by Mrm when a widget is created. The prototype of an MrmNcreateCallback is the same as any other Xt callback procedure. The *call_data* passed to the callback is an XmAnyCallbackStruct.

Controls

The *controls* subsection, if present, specifies a list of children. Each entry in the list may be a list of children, an object defined elsewhere, an object defined inline, or an automatically-created child. A list is specified with the symbol *controls* followed by the name of a controls lists defined elsewhere in the module. Specify an object defined elsewhere using an optional initial state of managed or unmanaged, followed by *user_defined* or a widget class and the name of the child widget. If the same child widget occurs more than once in a widget's *controls* section, an instance of the child is created for each occurrence.

An inline object definition is similar, but the name of the child widget is replaced by a set of widget attributes. The name of the inline widget is automatically generated by the UIL compiler. Inline definitions can be used to define widget instances that have few or no attributes and that do not need to be referenced by name. You may wish to avoid inline definitions, however, since the widget name is not well-defined, which makes customization via X resources difficult. An automatically-created child is specified by the name of the child followed by an attributes list. Appendix D, *Table of UIL Objects*, lists the automatically-created children of the built-in Motif widgets. The ability to specify attributes for automatically-created children is only available in Motif 1.2 and later.

If the widget instance being defined is from a built-in Motif widget class, the children specified in the *controls* section must be valid for the widget class, but any user-defined children can be specified. If the widget instance being defined is a user-defined widget, any built-in or used-defined children can be specified in the *controls* section. The UIL compiler verifies that the children specified in the controls section are allowable children for the widget being defined. Appendix D, *Table of UIL Objects*, lists the valid children for each built-in widget class. Any children are allowed for user-defined widgets. If an

invalid child is specified in a widget's controls section, the UIL compiler generates an error and no UID file is produced.

From Motif 2.0 and later, the controls section can be used to specify constituent entries in an XmRenderTable or XmTabList pseudo-object. For the XmRenderTable object, the controls section lists a set of further objects of type XmRendition. For the XmTabList object, each listed control is an object of type XmTab. The rendition and tab list objects are associated with one another by specifying an XmTabList object as an XmNtabList value within the controls section of an XmRendition object. The example given below clarifies the relationships.

Usage

A named widget can be specified as a value for a resource of type *widget*, such as the Form constraint resource XmNleftWidget, or as the argument of a callback procedure declared with a parameter of type any or widget. Prior to Motif 1.2.1, UIL does not allow the type *widget* to be used as an argument type in a procedure declaration. You can specify type any to work around this problem. Older versions of UIL may require the widget class name to precede a widget value that is assigned to a resource or used as callback parameter. Since all versions of UIL accept this syntax, you can avoid potential difficulties by always using it.

Mrm places some restrictions on the widgets that can be assigned to a resource or used as a callback parameter. The widget must be a member of the same hierarchy as the widget definition in which it is used. A widget hierarchy includes the widget named in the call to `MrmFetchWidget`() or `MrmFetchWidgetOverride`() and the widgets created in the widget tree below it. If a named widget does not exist when a reference to it is encountered, Mrm waits until all of the widgets in the hierarchy have been created and tries to resolve the name again. If a widget reference still cannot be resolved, Mrm does not set the specified resource or add the specified callback. As of Motif 1.2, Mrm does not generate a warning message when this situation occurs.

The advantage of this functionality is that, unlike in C, you do not have to worry about the creation order of a widget hierarchy when you are specifying a widget as a resource value or callback parameter. UIL also makes the creation of OptionMenus and MenuBars easier by allowing you to specify a PulldownMenu as the child of an OptionMenu or CascadeButton. The XmNsubMenuId resource of the object is automatically set to widget ID of the menu. When specified as the child of a CascadeButton, the menu is created as a child of the MenuBar that contains the button. As a convenience, you can also specify a PopupMenu as the child of any widget (but not gadget).

As of Motif 1.2, the UIL compiler does not support user-defined imported widgets. If you need to import a user-defined widget, declare it with the type of a built-in widget that is a valid child for the context where the imported widget is used.

Example

```
...

object romulus : XmPushButton gadget {
    callbacks {
        XmNactivateCallback = procedure create_Rome();
    };
};

object remus : imported XmPushButton;

object mars : XmForm {
    arguments {
        XmNbackground = color ('orange');
    };
    controls {
        ! Define a couple of children.
        XmPushButton romulus;
```

```
            unmanaged XmPushButton remus;
            ! Define an inline separator.
            XmSeparator { };
        };
};

object thing : user_defined procedure create_thing {
        ...
};

object scale : XmScale {
        controls {
            ! Set the labelString on the automatically created label.
            Xm_Title {
                arguments {
                    XmNlabelString = 'Temperature';
                    XmNrenderTable = rtable_1;
                };
            };
        };
};

! Motif 2.0 and later: pseudo-objects for rendition
! XmRenderTable objects contain rendition objects as controls
object rtable_1 : XmRenderTable {
        controls {
            XmRendition rendition_1;
        };
};

! XmRendition objects contain XmTabList objects as controls
! Note that XmNtag is not a supported argument:
! the tag is implicitly the object name
object rendition_1 : XmRendition {
        arguments {
            XmNfontName = "fixed";
            XmNunderlineType = XmDOUBLE_LINE;
        };
        controls {
            XmTabList tablist_1;
        };
};

! XmTabList objects contain XmTab objects as controls
object tablist_1 : XmTabList {
        controls {
            XmTab tab1;
            XmTab tab2;
        };
};

object tab1 : XmTab {
        arguments {
            XmNtabValue = 1.75;
            XmNunitType = XmCENTIMETERS;
            XmNoffsetModel = XmABSOLUTE;
        };
```

```
};

object tab2 : XmTab {
    arguments {
        XmNtabValue = 2.0;
        XmNunitType = XmCENTIMETERS;
        XmNoffsetModel = XmRELATIVE;
    };
};
...
```

See Also

`MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `MrmRegisterNames`(3), `MrmRegisterNamesInHierarchy`(3), `list`(5), `procedure`(5), `value`(5), `any`(6), `argument`(6), `compound_string_table`(6), `reason`(6), `widget`(6).

Name

procedure – procedure declaration section.

Syntax

procedure *procedure_name* [([*value_type*])];
[...]

Description

The procedure section contains declarations of procedures that can be used as a callback for a widget or as a user-defined widget creation function. Procedures can also be used in a procedure list; procedure lists are used to associate more than one callback procedure with a specific callback. Procedure lists are described on the list reference page.

The procedure section begins with the UIL keyword procedure, followed by list of procedure declarations. Each declaration consists of the procedure name followed by optional parentheses enclosing an optional parameter type. Valid type names are listed in the Introduction to *[cmtr06].

Usage

A procedure declaration can be used to specify whether a procedure expects a parameter, and if so, the type of the parameter. The UIL compiler verifies that a procedure reference conforms to its declaration. If a procedure name is not followed by parentheses, the compiler does not count parameters or perform any type checking when the procedure is used. Zero arguments, or one argument of any type, can be used in the reference.

If the procedure name is followed by an empty pair of parentheses, a reference to the procedure must contain zero arguments. User-defined widget creation functions should be declared as taking no parameters, although the UIL compiler does not enforce this rule.

If the procedure name is followed by a parenthesized type name or widget class, a reference to the procedure must contain exactly one argument of the specified type or class. If the type any is specified, the reference can contain an argument of any type. Prior to Motif 1.2.1, the UIL compiler generates an error if a widget class name is specified as the type in a procedure declaration. If the parameter to a callback procedure is an imported value or an identifier that cannot be resolved at run-time, a segmentation fault may occur when the callback is called.

Because identifiers and procedures are registered in the same name space with MrmRegisterName() and MrmRegisterNamesInHierarchy(), it is possible to declare a value as a procedure in the UIL source, even though the entry that is registered may not be a procedure. An attempt to call a non-procedure value usually causes an application to crash.

Example

```
...
procedure
    exit();
    print (string);
    XawCreateForm();
    popup (XmPopupMenu);

object form : user_defined procedure XawCreateForm { };

object quit : XmPushButton {
    callbacks {
        MrmNcreateCallback = procedure print ('Hello!');
        XmNactivateCallback = procedure exit();
        XmNdestroyCallback = procedure print ('Goodbye!');
    };
};
```

See Also

MrmFetchWidget(3), MrmFetchWidgetOverride(3), MrmRegisterNames(3), MrmRegisterNamesInHierarchy(3), identifier(5), list(5), object(5).

Name

value – variable definition and declaration section.

Syntax

value *value_name :* [exported | private] *value_expression* | imported *value_type;*
[...]

Description

The *value* section contains variable definitions and declarations. A variable is defined by assigning a value to it. A variable declaration is used to inform the UIL compiler of the existence of a variable defined in another module. The value assigned to a variable may be an arithmetic or string expression, a literal value, or another variable or identifier.

A value can be declared with a storage class of *private*, *exported*, or *imported*. Values are private by default. Private and exported values consist of a named variable and the value that is assigned to it. Private values are only accessible within the module in which they are defined. An exported variable definition includes the symbol *exported* before the value assigned. Exported values are accessible in other modules and from the application, in addition to the module in which they are defined.

You can access an exported value in another module by declaring it as an imported value in the module where you want to access it. Imported value declarations consist of a named variable, the symbol *imported*, and the type of the variable. If an imported value is exported from more than one module, the value from the module that occurs first in the array passed to `MrmOpenHierarchyPerDisplay`() is used.

Values of all types can be declared as private; values of most types can be declared as exported and imported. The Introduction to Section 6, *UIL Data Types*, contains a table that summarizes the storage classes that are allowed for each type.

Usage

Variables used in an expression can be forward referenced. However, the specification of some complex literals cannot contain forward-referenced values. The UIL compiler indicates a value cannot be found in these cases. Refer to the reference page for a type to see if its literal representation can contain forward references.

Typically, the value of a variable used in an expression or in the specification of a complex literal must be accessible in the module in which it is used. As a result, in most cases you cannot use an imported variable in an expression or complex value specification. If an imported value is used in an invalid context, the UIL compiler issues an error message.

Example

```
...
! See individual type reference pages for additional examples.
value
    version     : exported 1002;
    Soothsayer  : 'Beware the ides of March.';
    ides        : 15;
    background  : imported color;
    ...
```

See Also

`MrmFetchBitmapLiteral`(3), `MrmFetchColorLiteral`(3), `MrmFetchIconLiteral`(3), `MrmFetchSetValues`(3), `argument`(6), `asciz_string_table`(6), `boolean`(6), `color`(6), `color_table`(6), `compound_string`(6), `compound_string_table`(6), `float`(6), `font`(6), `fontset`(6), `font_table`(6), `icon`(6), `integer`(6), `integer_table`(6), `keysym`(6), `reason`(6), `rgb`(6), `single_float`(6), `string`(6), `translation_table`(6), `wide_character`(6), `xbitmapfile`(6).

Section 6 - UIL Data Types

This page describes the format and contents of each reference page in Section 6, which covers each of the UIL data types.

Name

Type – a brief description of the data type.

Synopsis

Syntax:

The literal syntax for specifying a value of the data type. Anything in constant width type should be typed exactly as shown. Items in italics are expressions that should be replaced by actual values when you specify a value. Anything enclosed in brackets is optional. An ellipsis (...) means that the previous expression can be repeated multiple times and a vertical bar (|) means to select one of a set of choices.

MrmType:

The Mrm value type that corresponds to the data type. These types are returned by MrmFetchLiteral().

Availability

This section appears for data types that were added in Motif 2.0 or later.

Description

This section gives an overview of the data type. It explains the literal syntax that is used to specify a value of the type in a UIL module.

The UIL compiler supports *integer*, *float*, *single_float*, *boolean*, *string*, and *compound_string* expressions in most contexts where a value of one of the types is expected. Expressions can include literal or named values, but any named values that are used must be declared private *or exported* because the result of an expression cannot be computed if it contains an imported *value.*

The UIL compiler allows both string and arithmetic expressions. String expressions contain NULL-terminated strings and compound strings, while arithmetic expressions can contain integer, float, single_float, and boolean values.

A string expression, consists of two or more string or compound_string values concatenated with the string concatenation operator (&). The string and compound_string reference sections contains more details and examples of string concatenation.

An arithmetic expression consists of one or more boolean, integer, single_float, or float values and one or more arithmetic operators. The following operations can be used in arithmetic expressions:

Operator	Type	Operand Types	Operation	Precedence
~	unary	boolean	NOT	1 (highest)
		integer	One's complement	1
-	unary	integer	Negation	1
		float	Negation	1
+	unary	integer	None	1
		float	None	1
*	binary	integer	Multiplication	2
		float	Multiplication	2
/	binary	integer	Division	2
		float	Division	2
+	binary	integer	Addition	3
		float	Addition	3
-	binary	integer	Subtraction	3
		float	Subtraction	3
>>	binary	integer	Shift right	4
<<	binary	integer	Shift left	4
&	binary	boolean	AND	5
		integer	Bitwise AND	5
|	binary	boolean	OR	6
		integer	Bitwise OR	6
^	binary	boolean	XOR	6
		integer	Bitwise XOR	6 (lowest)

When the UIL compiler evaluates an expression, higher precedence operations are performed before those of lower precedence. Binary operations of equal precedence are evaluated from left to right, while unary operations of equal precedence are evaluated from right to left. You can change the default order of evaluation by using parentheses to group subexpressions that should be evaluated first. For example, in the expression 2+4*5, 4*5 is evaluated first, followed by 20+2. If the expression is written (2+4)*5, then 2+4 is evaluated first, followed by 6*5.

The type of an expression is the type of its most complex operand. The UIL compiler converts the value of the less complex type in an operation to a value of the

most complex type. The order of complexity for operands in a string expression is string followed by compound_-string. For operations in an arithmetic expression, the order is boolean, integer, single_float, and float.

For example, if a string expression contains only strings, the type of the concatenated expression is string, but if it contains both strings and compound strings, its type is compound_-string. The result of concatenating two NULL-terminated strings is a NULL-terminated string, unless the two strings have different character sets or writing directions, in which case the result is a compound string. If an arithmetic expression contains only integers, the type of an expression is integer, but if it contains both integers and floats, its type is float.

The table below summarizes the valid uses of the types documented in this section. For each type, the table indicates the supported storage classes. It also specifies whether or not values of the type can be specified literally and whether or not the type can be used for a procedure parameter and as an argument type. The final column lists the Motif Resource Manager (Mrm) routine that can be used to fetch values of the type. If certain information is not relevant for a type, the table entry indicates that it is not applicable (NA).

Type	Supported Storage Classes			Literal Value	Reason/ Parameter	Fetch Function
	Private	Exported	Imported			
any	NA	NA	NA	No	Yes	NA
argument	Yes	No	No	Yes	No	NA
asciz_table	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
boolean	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
character_set	NA	NA	NA	Yes	No	NA
class_rec_name	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
color	Yes	Yes	Yes	Yes	Yes	MrmFetchColorLiteral
color_table	Yes	No	No	Yes	No	NA
compound_string	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
compound_string _component	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
compound_string _table	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
float	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
font	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral

Type	Supported Storage Classes			Literal Value	Reason/ Parameter	Fetch Function
	Private	Exported	Imported			
fontset	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
font_table	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
icon	Yes	Yes	Yes	Yes	Yes	MrmFetchIconLiteral, MrmFetchBitmapLiteral
integer	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
integer_table	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
keysym	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
pixmap	No	No	Yes	No	Yes	NA
reason	Yes	No	No	Yes	No	NA
rgb	Yes	Yes	Yes	Yes	Yes	MrmFetchColorLiteral
single_float	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
string	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
translation_table	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
wide_character	Yes	Yes	Yes	Yes	Yes	MrmFetchLiteral
widget	Yes	Yes	Yes	Yes	Yes	MrmFetchWidget, MrmFetchWidgetOverride
xbitmapfile	Yes	Yes	Yes	Yes	Yes	MrmFetchIconLiteral

The UIL compiler may not generate errors when some of the types are used incorrectly. These cases are documented in the individual type reference pages.

As of Motif version 1.2, the UIL compiler does not support the assignment of a character_set value to a named variable. A built-in or literal character set must be specified in all contexts in which a character set is expected. In addition, prior to Motif 1.2.1, UIL may generate an error if the type widget is used as an argument or reason type. In this case, the type any can be used as a workaround.

Usage

This section provides less formal information about the data type: when and how you might want to use it and things to watch out for.

Example

This section provides examples of the use of the type.

See Also

This section refers you to related functions, UIL file format sections, and UIL data types. The numbers in parentheses following each reference refer to the sections of this book in which they are found.

Name

any – type checking suppression type.

Synopsis

Syntax:

any

MrmType:

MrmRtypeAny

Description

The *any* type is used to suppress type checking for values passed to callback procedures or assigned to user-defined arguments. When a callback parameter or user defined-argument type is specified as *any*, the UIL compiler allows a value of any type to be used. Because the type *any* is only used to specify an expected type in these two cases, it does not have a literal syntax and values of type *any* cannot be defined or declared.

Usage

The *any* type specifier is used when values of more than one type can be passed as a callback parameter or assigned to an argument. It can also be used when a callback or argument expects a type that is not predefined by the UIL compiler.

Since no type checking is performed on callback parameters or arguments declared as type *any*, it is possible to specify a value that is not expected by the callback or widget. You should use caution when specifying the value for a callback or argument that uses the any type.

Example

```
...
! Define activate procedure that takes different arguments depending upon
! usage context. Context must be checked in C code before value is used.
procedure
    activate (any);
! Define a resource that can be set to different types.
! Widget checks type field at run-time to determine value type.
value
    XtNlabelValue    : argument ('labelValue', any);
    XtNlabelType     : argument ('labelType', integer);
```

See Also

`procedure`(5), `argument`(6).

Name

argument – user-defined resource type.

Synopsis

Syntax:

argument (*string_expression* [, *argument_type*])

MrmType:

none

Description

An *argument* value represents a user-defined resource. An *argument* is represented literally by the symbol *argument*, followed by a string expression that evaluates to the name of the resource and an optional resource type. The name of the resource is assigned to the name member of the ArgList structure passed to `XtSetValues`(). The name is typically the name of a resource with the XmN or XtN prefix removed. The type of the argument, if specified, is used by the UIL compiler to perform type checking of assignments to the resource. If omitted, the type defaults to any.

Usage

A user-defined resource can be used in the *arguments* section of a UIL module, for both built-in Motif widgets and user-defined widgets. While user-defined arguments are typically assigned to a named variable in the value section, they can also be specified literally in the arguments section of an *object* definition. If you are defining arguments for a widget or widget set that is not predefined, you should define them as named variables in a separate UIL module that can be included by any module that uses the widget(s).

Arguments must be private values; they cannot be imported or exported. The UIL compiler allows imported and exported declarations, but it generates an error when the user-defined argument is used. Since argument values cannot be exported, they cannot be retrieved by an application.

The argument type can only be used to define non-callback resource types. The *reason* type is used to specify user-defined callback resources.

Some versions of the UIL compiler may not allow the definition of arguments of type widget. If you encounter this problem, use the type any as a workaround. The compiler may allow the definition of arguments of type *argument* or *reason*. If arguments with these types are used, the actual value set as the widget's resource is undefined.

Example

From *Xaw/Tree.uih*:

```
! Resource and definitions for the Athena Tree widget.
value
    XtNautoReconfigure : argument ('autoReconfigure', boolean);
    XtNgravity         : argument ('gravity', integer);
    NorthGravity       : 2;
    WestGravity        : 4;
    EastGravity        : 6;
    SouthGravity       : 8;
    ! Use any type because compiler may not allow widget:
    XtNtreeParent : argument ('treeParent', any);
...
```

From *my_module.uil*:

```
include file 'Xaw/Tree.uih';
object parent : XmPushButton { }
object child : XmPushButton {
    arguments {
        XtNtreeParent = parent;
    };
};

object tree : procedure user_defined XawCreateTreeWidget {
    arguments {
        XtNautoReconfigure = false;
        XtNgravity = NorthGravity;
    };
    controls {
        XmPushButton parent;
        XmPushButton child;
    };
};
```

See Also

`MrmRegisterClass`(3), `include`(5), `object`(5), `reason`(6).

Name

asciz_string_table – array of NULL-terminated strings.

Synopsis

Syntax:

asciz_table (*string_expression* [, ...]) or
asciz_string_table (*string_expression* [, ...])

MrmType:

MrmRtypeChar8Vector

Description

An *asciz_string_table* value represents an array of NULL-terminated strings. An *asciz_string_table* is represented literally by the symbol *asciz_table* or *asciz_string_table*, followed by a list of string expressions separated by commas. String variables in this list can be forward referenced.

Usage

There are no built-in Motif resources of type *asciz_string_table*, so values of this type are usually passed as callback parameters or retrieved with `MrmFetchLiteral`(). The type *asciz_string_table* can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal. An *asciz_string_table* obtained by the application as a callback parameter, a widget resource, or with `MrmFetchLiteral`() is NULL-terminated.

Example

```
...
! Declare a procedure that expects an array of NULL-terminated strings.
procedure
    set_names (asciz_table);

! Define a couple of asciz_tables
value
    dwarfs      : asciz_table (’Dopey’, ’Doc’, ’Sneezy’, ’Sleepy’,  ’Happy’,
    ’Grumpy’, ’Bashful’);
    numbers     : asciz_string_table (one, two);
    one         : ’one’;
    two         : ’two’;
    reindeer    : imported asciz_string;

! Define some asciz_table resources.
value
    XtNniceList         : argument (’niceList’, asciz_table);
```

```
        XtNnaughtyList   : argument ('naughtyList', asciz_table);

    object doit : XmPushButton {
        callbacks {
            XmNactivateCallback = procedure set_names (dwarfs);
        };
    };
    ...
```

See Also

`MrmFetchLiteral`(3), `procedure`(5), `argument`(6), `compound_string`(6), `compound_string_table`(6), `string`(6).

Name

boolean – true/false type.

Synopsis

Syntax:

true | on | false | off

MrmType:

MrmRtypeBoolean

Description

Values of type *boolean* may be either true (on) or false (off). A *boolean* value is represented literally by *true, false, on,* or *off*. A *boolean* variable can be defined in the value section by setting a named variable to one of these literal values or to another boolean variable.

Usage

The type name *boolean* can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an *argument* literal.

A boolean value can be explicitly converted to an *integer, float,* or *single_float* value by specifying the conversion type followed by the *boolean* value in parentheses. *true* and *on* convert to the value 1 or 1.0, while *false* and *off* convert to the value 0 or 0.0.

The storage allocated by Mrm for a *boolean* value is sizeof(int) not sizeof(Boolean). Because sizeof(Boolean) is less than sizeof(int) on many systems, you should use an int pointer rather than a Boolean pointer when retrieving a boolean value with `MrmFetchLiteral`().

Example

```
...
procedure
    set_sleepy_state (boolean);

value
    map_flag     : true;
    one          : integer (true);
    zero         : integer (false);
    debug        : imported boolean;
    XtNtimed     : argument ('timed', boolean);

object sleep : XmPushButton {
    arguments {
        XmNmapWhenManaged = map_flag;
```

```
        XmNtraversalOn = off;
    };
    callbacks {
        XmNactivateCallback = procedure set_sleepy_state (true);
    };
};
...
```

See Also

`MrmFetchLiteral`(3), `procedure`(5), `argument`(6), `float`(6), `integer`(6), `single_float`(6).

Name

character_set – character set type for use with strings and font lists.

Synopsis

Syntax:

character_set (*string_expression*
[, right_to_left = *boolean_expression*]
[, sixteen_bit = *boolean_expression*])

MrmType:
none

Description

The *character_set* type represents a user-defined character set that can be used when defining *strings, compound_strings, fonts, fontsets*, and *font_tables*. A character set specifies the encoding that is used for character values. A *character_set* is represented literally by the symbol *character_set*, followed by a string expression that names the character set and two optional properties.

If the *right_to_left* property of the character set for a string is set to *true*, the string is parsed and stored from right to left and compound strings created from the string have a direction component of XmSTRING_DIRECTION_R_TO_L. The default value of this property is *false*. The direction component used by a compound_string can be specified independently of the parsing direction using the compound_string literal syntax.

If the *sixteen_bit* property of the character set for a string is set to *true*, the string is interpreted as having double-byte characters. Strings with this property set to true must contain an even number of bytes or the UIL compiler generates an error.

Usage

A *character_set* value is used to specify the character set for *string, compound_string, font, fontset*, and *font_table* values. The *right_to_left* and *sixteen_bit* properties only apply to strings and compound strings and have no effect on character sets specified for fonts and fontsets.

Unlike most of the UIL types, the *character_set* type cannot be assigned to a named variable in a *value* section, or used as the type of an imported value, as a parameter type in a *procedure* declaration, or as the type in an *argument* literal. A character set value can only be specified with the *character_set* literal syntax.

If a *font, fontset*, or *font_table* that uses a user-defined character set is exported or used as a resource value, the UIL compiler may exit with a severe internal error. As a result, only the predefined character sets can be used with *font, fontset*, and

font_list values. You can work around this problem by specifying values of these types in an X resource file.

The UIL compiler may allow the use of string variables and the string concatenation operator (&) in a *character_set* name specification. Although no errors are generated, a string using such a character set may be incorrectly converted to a *compound_string* value. To avoid this problem, you should always specify a quoted string as the name in a *character_set* literal.

UIL defines a number of built-in character sets that you can use to define *string*, *compound_string*, *font*, *fontset*, and *font_table* values. The following table summarizes the built-in character sets:

UIL Name	Character Set	Parse Direction	Writing Direction	16 Bit
iso_latin1	ISO8859-1	L to R	L to R	No
iso_latin2	ISO8859-2	L to R	L to R	No
iso_latin3	ISO8859-3	L to R	L to R	No
iso_latin4	ISO8859-4	L to R	L to R	No
iso_latin5	ISO8859-5	L to R	L to R	No
iso_cyrillic	ISO8859-5	L to R	L to R	No
iso_arabic	ISO8859-6	L to R	L to R	No
iso_arabic_lr	ISO8859-6	L to R	R to L	No
iso_greek	ISO8859-7	L to R	L to R	No
iso_hebrew	ISO8859-8	R to L	R to L	No
iso_hebrew_lr	ISO8859-8	L to R	R to L	No
jis_katakana	JISX0201.1976-0	L to R	L to R	No
gb_hanzi	GB2313.1980-0	L to R	L to R	Yes
gb_hanzi_gr	GB2313.1980-1	L to R	L to R	Yes
jis_kanji	JISX0208.1983-0	L to R	L to R	Yes
jis_kanji_gr	JISX0208.1983-1	L to R	L to R	Yes
ksc_hangul	KSC5601.1987-0	L to R	L to R	Yes
ksc_hangul_gr	KSC5601.1987-1	L to R	L to R	Yes

Example

```
...
value
    ! Define font with user-defined character set.
    big: font ('*times-medium-r-normal-*-240-75-75-*',
    character_set = character_set ('body'));
    ! Declare some strings with user-defined character sets.
    player : #character_set (big) "Mookie Wilson";
    hello : exported #iso_hebrew "\355\345\354\371\";
    ...
```

See Also

`compound_string`(6), `font`(6), `fontset`(6), `font_table`(6), `string`(6).

Name

class_rec_name – widget class pointer type.

Synopsis

Syntax:

class_rec_name (*string_expression)*

MrmType:

MrmRtypeClassRecName

Description

The *class_rec_name* type represents a pointer to a widget class record. A class_rec_name value is represented literally by the symbol *class_rec_name*, followed by a string that specifies the class name. The string can either be the name of a class from a widget's class definition or the name of a widget creation function registered with `MrmRegisterClass`(). The string is converted to a widget class pointer at run-time by Mrm when a *class_rec_name* value is referenced. Mrm finds the widget class pointer corresponding to the name by searching the list of widgets registered with `MrmRegisterClass`(). This list includes the built-in Motif widgets and any user-defined widgets that have been registered.

Usage

The type *class_rec_name* can be used as the type of an imported value, as the parameter type in a procedure declaration, or as the type in an argument literal. None of the built-in Motif widgets have a class_rec_name resource, however. If a *class_rec_name* value is specified as a resource value for a widget and the conversion of the class name string to a widget class pointer fails at run-time (inside a call to `MrmFetchWidget`(), `MrmFetchWidgetOverride`(), or `MrmFetchSetValues`()), Mrm does not set the resource. If `MrmFetchLiteral`() is used to retrieve the value and the conversion fails, MrmNOT_FOUND is returned.

Example

```
...
value
    pbclass : class_rec_name ('XmPushButton');
...
```

See Also

`MrmFetchSetValues`(3), `MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `MrmInitialize`(3), `MrmRegisterClass`(3), `procedure`(5), `argument`(6).

Name

color – color specified as color name.

Synopsis

Syntax:

color (*string_expression* [foreground | background])

MrmType:

MrmRtypeColor

Description

A *color* value represents a named color. A color is represented literally by the symbol *color*, followed by a string expression that evaluates to the color name and an optional foreground or background property to indicate how the color is displayed on a monochrome screen. Mrm converts the color name to an X Color at run-time with `XAllocNamedColor`() on a color display, or chooses black or white on a monochrome display. The X server maintains a color name database that is used to map color names to RGB values. The text version of this database is typically in the file */usr/lib/x11/rgb.txt*. See Volume One, *Xlib Programming Manual*, and Volume Two, *Xlib Reference Manual*, for more information on color allocation.

Usage

The *color* type can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal. An *rgb* value can also be specified in any context that a color value is valid. There are several built-in Motif *color* resources, such as XmNforeground and XmNbackground.

The optional *foreground* and *background* properties can be used to specify the mapping of colors on a monochrome display or when a color allocation fails because the colormap is full. Mrm dynamically determines the appropriate foreground or background color based on the context in which a *color* value is used.

When a *color* is used as a resource value for a widget (directly or indirectly in an icon's *color_table*), the background and foreground colors are obtained from the widget. When a color is retrieved for the *color_table* of an icon retrieved with `MrmFetchIconLiteral`(), the background and foreground colors are supplied by the application as arguments to the function.

If the *foreground* or *background* property is not specified, Mrm uses the Color returned by `XAllocNamedColor`() on a monochrome display. When an allocation fails on a color display and neither property is specified, black is used. In addition, black is always used when an allocation on a color display fails in

MrmFetchColorLiteral(); the procedure does not take fallback background and foreground colors arguments.

As of Motif version 1.2.1, the color substitutions described above do not take place. When a color allocation fails for a color specified directly or indirectly as a resource value, the resource is not set. If the allocation fails in a call to MrmFetchColorLiteral() or MrmFetchIconLiteral(), MrmNOT_FOUND is returned.

Example

```
...
value
    background : color ('chocolate mint', background);
    foreground : color ('whipped cream', foreground);

object label: XmLabel {
    arguments {
        XmNbackground = color ('red');
    };
};
...
```

See Also

MrmFetchColorLiteral(3), MrmFetchIconLiteral(3), MrmFetchSetValues(3), MrmFetchWidget(3), MrmFetchWidgetOverride(3), color_table(6), icon(6), rgb(6).

Name

color_table – character-to-color mapping type.

Synopsis

Syntax:

color_table (*color_expression* = *'character'* [, ...])

MrmType:

none

Description

A *color_table* value is used to define a mapping from color names or RGB values to the single characters that are used to represent pixel values in icons. A color_table is represented literally by the symbol *color_table*, followed by a list of mappings. Each mapping associates a previously-defined color value with a single character. A color value can be a variable or a literal of type *color* or *rgb*, the global *background color*, or the symbol *foreground color*.

Usage

The sole purpose of a *color_table* is to define colors that can be used in an icon definition. Because the color mappings are needed at compile-time to construct an icon, a *color_table* value must be private. The UIL compiler may allow an imported or exported *color_table* definition, but it generates an error when the value is used. Unlike most other UIL types, a *color_table* cannot be used as a parameter type in a procedure declaration or as the type in an argument literal.

The *color* values *background color* and *foreground color* can be used to map a character to the background or foreground color. These colors are determined at run-time by Mrm, based on the context in which an *icon* is used. When an *icon* is a resource value for a widget, the foreground and background colors are obtained from the widget. When an *icon* is retrieved by the application with `MrmFetchIconLiteral`(), the foreground and background colors are supplied by the application as arguments to the function.

The colors in a *color_table* are allocated at run-time by Mrm when an icon that uses the color table is retrieved as the value for a widget resource or retrieved by the application with `MrmFetchIconLiteral`(). See the *color* reference page for a description of how Mrm allocates colors and what happens when a color allocation fails.

The UIL compiler may not perform type checking on the color values in a *color_table*. If the compiler allows the use of a value that is not a color, it will crash when the *color_table* is used.

Example

```
...
value
      blue : color (’blue’);
      yellow : rgb (65535,65535,0);
      pallete :  color_table (    background color = ’ ’,
                                  foreground color = ’*’,
                                  color (’red’) = ’r’,
                                  rgb (0,65535,0) = ’g’,
                                  blue = ’b’,
                                  yellow = ’y’);

plus : icon (color_table = pallete,  ’brb’, ’rrr’, ’brb’);
...
```

See Also

`MrmFetchIconLiteral`(3), `color`(6), `icon`(6), `rgb`(6).

Name

compound_string – Motif compound string type.

Synopsis

Syntax:

compound_string (*string_expression*
[, character_set = *character_set*]
[, right_to_left = *boolean_expression*]
[, separate = *boolean_expression*])

MrmType:

MrmRtypeCString

Description

A *compound_string* value represents a Motif XmString. An XmString is the data type for a Motif compound string. The Motif toolkit uses compound strings, rather than character strings, to represent most text values. A compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. The tag specifies the font, and thus the character set, that is used to display the text component.

UIL-generated *compound_strings* can contain up to four components: a single-byte, multi-byte, or wide-character string, a character set, a writing direction, and a separator. Like NULL-terminated strings, *compound_strings* can be concatenated with the concatenation operator (&). A *compound_string* is represented literally by the symbol *compound_string*, followed by a string expression and an optional list of properties. The valid properties are *character_set*, *right_to_left*, and *separate*. They may be specified in any order, but each may occur only once.

The character_set property is used to establish the character set of the *compound_string*. It can be set to one of the UIL built-in character sets or to a user-defined character set. If a character set is specified in the definition of the string using the *#character_set* notation, it takes precedence over the character_set property setting. If the *character_set* property is omitted, the default character set of the module is used.

The *right_to_left* property is used to set the writing direction of the *compound_string*. If the *right_to_left* property is omitted, the writing direction defaults to that of the character set of the *compound_string*.

When the *separate* property is set to true, UIL adds a separator component to the end of the *compound_string*. Separators usually appear as line breaks when a compound string is displayed. If omitted, the separate property defaults to false. Newline characters present in the string expression of a compound string literal are not converted to separators.

Usage

When a *compound_string* literal contains a string expression consisting of two or more concatenated strings, they are combined into a single component if the character set and writing direction of each is the same. If any of the character sets differ, each string is placed in a separate string component with its own character set and direction components. If the separate property is set to true, a separator component is added to the end of the entire compound_string.

A *compound_string* with a character_set that differs from XmFALLBACK_-CHARSET is only displayed correctly in a Motif widget if the XmFontList of the widget includes an XFontStruct or an XFontSet entry for the character_set.

The type *compound_string* can also be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an *argument* literal.

Example

```
...
procedure
    set_label_string (compound_string);

value
    ying  : "Ying";
    yang  : #iso_latin1"Yang";
    left  : compound_string (ying, character_set=iso_latin1, separate=true);
    right : compound_string (yang, right_to_left=true);
    day   : compound_string ('moon' & ' ' & 'sun');
    other : imported compound_string;
lines : exported left & right;

object verse : XmLabel {
    arguments {
        XmNlabelString = lines;
    };
};

value
    XtNgraphicCaption : argument ('graphicCaption', compound_string);
...
```

See Also

`XmStringCreate`(1), `XmStringCreateLocalized`(1), `character_set`(6), `compound_string_component`(6), `compound_string_table`(6), `string`(6).

Name

compound_string_component – Motif compound string component type.

Synopsis

Syntax:

compound_string_component (*component_type* [, { *string* | *enumval* }])

MrmType:
MrmRtypeCString

Availability

Motif 2.0 and later.

Description

A *compound_string_component* value represents a compound string containing a single component. It is the UIL equivalent of the Motif function `XmStringComponentCreate`(). The compound string so produced can be concatenated with other segments to create more complex compound strings. As for the *compound_string* data type, the symbol & is the concatenation operator.

The *component_type* parameter specifies the type of compound string segment to be created. The value is one of the constants defined for the XmStringComponentType enumeration. Depending upon the type of the segment, a second qualifying parameter may be required: the valid component types, together with any extra argument is as follows:

Where *component_type* is XmSTRING_COMPONENT_DIRECTION, *compound_string_component* is equivalent to the Motif function `XmStringDirectionCreate`(), and the XmStringDirection argument is as required for that function: XmSTRING_DIRECTION_L_TO_R, XmSTRING_DIRECTION_R_TO_L, or XmSTRING_DIRECTION_DEFAULT.

Usage

The *compound_string_component* data type can be used in an analogous fashion to the *compound_string* type. The differences lie in the degree of control in constructing the compound strings: tab, separator, and rendition components can be created. The components XmSTRING_COMPONENT_RENDITION_BEGIN and XmSTRING_COMPONENT_RENDITION_END take as argument a string which is matched against a rendition tag within the current render table.

Example

```
...
value
    tab             : compound_string_component
                            (XmSTRING_COMPONENT_TAB);
    separator       : compound_string_component
                            (XmSTRING_COMPONENT_SEPARATOR);
    charset         : compound_string_component
                            (XmSTRING_COMPONENT_CHARSET,
                            iso_latin1);
    l_to_r_text     : compound_string_component
                            (XmSTRING_COMPONENT_TEXT,
                            "left_to_right");
    r_to_l_text     : compound_string_component
                            (XmSTRING_COMPONENT_TEXT,
                            "tfel-ot-thgir");
    r_to_l          : compound_string_component
                            (XmSTRING_COMPONENT_DIRECTION,
                            XmSTRING_DIRECTION_R_TO_L);
    l_to_r          : compound_string_component
                            (XmSTRING_COMPONENT_DIRECTION,
                            XmSTRING_DIRECTION_L_TO_R);
    cstring         : r_to_l & charset & r_to_l_text & separator & l_to_r &
                            l_to_r_text;

object label: XmLabel {
    arguments {
        XmNlabelString = cstring;
    }
};
...
```

See Also

`XmStringComponentCreate`(1), `XmStringDirectionCreate`(1), `compound_string`(6), `compound_string_table`(6), `string`(6).

Name

compound_string_table – array of compound strings.

Synopsis

Syntax:

compound_string_table (*string_expression* [, ...]) or
string_table (*string_expression* [, ...])

MrmType:

MrmRtypeCStringVector

Description

A *compound_string_table* value represents an array of Motif XmStrings. An XmString is the data type for a Motif compound string. The Motif toolkit uses compound strings, rather than character strings, to represent most text values. A compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. The tag specifies the font, and thus the character set, that is used to display the text component.

A *compound_string_table* is represented literally by the symbol *compound_string_table* or *string_table*, followed by a list of *string* or *compound_string* expressions. The UIL compiler automatically converts a string expression to a *compound_string*.

Usage

A common use of *compound_string_table* values is to set resources of the type XmStringTable in a UIL module or in the application with `MrmFetchSetValues`(). When a *compound_string_table* is assigned to a built-in XmStringTable resource, UIL automatically sets the corresponding count resource. The table below lists the XmStringTable resources and their related count resources.

Widget	XmStringTable Resource	Related Resource
XmList	XmNitems	XmNitemCount
XmList	XmNselectedItems	XmNselectedItemCount
XmSelectionBox	XmNlistItems	XmNlistItemCount
XmCommand	XmNhistoryItems	XmNhistoryItemCount
XmFileSelectionBox	XmNdirListItems	XmNdirListItemCount
XmFileSelectionBox	XmNfileListItems	XmNfileListItemCount

The associated count is not automatically set for compound_string_tables that are assigned using `MrmFetchSetValues`().

The type *compound_string_table* can also be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal. A *compound_string_table* that is obtained by the application as a callback parameter, a widget resource, or with `MrmFetchLiteral`() is NULL-terminated.

If a *compound_string_table* contains a forward reference to a *compound_string* value, all items in the list before that entry may be lost by the UIL compiler. To avoid this problem, you should be sure to define all compound_strings used in a *compound_string_table* before they are referenced.

Example

```
...
procedure
    set_items (string_table);

value
    fruit_list : string_table ('apple', 'banana', 'grape');

object list : XmList {
    arguments {
        XmNitems = fruit_list;
    };
};

value
    XtNnameList : argument ('nameList', compound_string_list);
...
```

See Also

`character_set`(6), `compound_string`(6), `compound_string_component`(6), `string`(6).

Name

float – double-precision floating point type.

Synopsis

Syntax:

[+ | -]*integer.integer* [e [+ | -]*integer*]

MrmType:

MrmRtypeFloat

Description

A *float* value represents a negative or positive double-precision floating point number. A *float* is represented literally by an optional sign, one or more consecutive digits which must include a decimal point, and an optional exponent. The UIL compiler uses `atof`() to convert literal float values to the architecture's internal representation.

A float can also be represented literally by the symbol *float* followed by a *boolean*, *integer*, or *single_float* expression. The expression is converted to a float and can be used in any context that a *float* value is valid. A float is formed from a boolean by converting true and on to 1.0 and false and off to 0.0.

Usage

The allowable range of a *float* value is determined by the size of a C double on the machine where the UIL module is compiled. Since a double on most architectures is typically a minimum of four bytes, float values may safely range from 1.4013e-45 to 3.40282e+38 (positive or negative). Although many architectures represent a double using eight bytes, you can ensure greater portability by keeping *float* values within the four-byte range. The UIL compiler generates an error if it encounters a *float* outside of the machine's representable range.

The type *float* can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal.

Example

```
...
! Declare some floating point values.
value
    pi              : 3.14159;
    burn_rate       : imported float;
    one_point_oh    : float (true);
    ten_even        : float (10);
```

```
! Declare a procedure which takes a float parameter.
procedure
    set_temperature (float);

! Declare an argument of type float.
value
    XtNorbitalVelocity : argument ('orbitalVelocity', float);
...
```

See Also

`boolean`(6), `integer`(6), `single_float`(6).

Name

font – XFontStruct type.

Synopsis

Syntax:

font (*string_expression* [, character_set = *character_set*])

MrmType:

MrmRtypeFont

Description

A *font* value represents an XFontStruct, which is an Xlib structure that specifies font metric information. A font is represented literally by the symbol font, followed by a string expression that evaluates to the name of the font and an optional character_set. All parts of the string expression that make up the font name must be private to the UIL module. The *character_set* is associated with the font if it appears in a font_table. If *character_set* is not specified, it is determined from the codeset portion of the LANG environment variable if it is set, or XmFALLBACK_CHARSET otherwise.

The string expression that specifies the font name is an X Logical Font Description (XLFD) string. This string is stored in the UID file and used as a parameter to `XLoadQueryFont`() at run-time to load the font. See Volume One, *Xlib Programming Manual*, and Volume Two, *Xlib Reference Manual*, for more information on fonts.

Usage

You can use a *font* value to specify a font or *font_table* resource. When a font is assigned to a *font_table* resource, at run-time Mrm automatically creates an XmFontList that contains only the specified font. A font value can also be used as an element in font_table, although in this context it must be private to the UIL module.

The font type can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal.

In some versions of UIL, the default character_set is always ISO8859-1, instead of being based on the LANG environment variable or XmFALLBACK_CHARSET.

The UIL compiler may exit with a severe internal error if a user-defined *character_set* is used in a font that is exported or specified as a resource value. If this problem occurs in your version of UIL, only predefined *character_set* values can be used in *font*, *fontset*, and *font_table* values. The workaround is to specify these problematic values in an X resource file.

Example

```
...
procedure
    change_font (font);

value
    title_font      : font ('-*-helvetica-bold-r-nor-
    mal-*-160-100-100-*-iso8859-1');
    family          : 'courier';
    style           : 'medium';;
    body_font       : font ('-*-' & family &'-'& style & '-r-nor-
    mal-*-120-100-100*-iso8859-1');
    kanjiFont       : font ('-*-JISX0208.1983-1', character_set = jis_kanji);
    default_font    : imported font;

value
    XtNheadlineFont : argument ('headlineFont', font);

object label: XmLabel {
    arguments {
        XmNfontList = title_font;
    };
};
...
```

See Also

`character_set`(6), `fontset`(6), `font_table`(6).

Name

font_table – Motif font list type.

Synopsis

Syntax:

font_table ([*character_set* =] *font_expression* [, ...])

MrmType:

MrmRtypeFontList

Description

A *font_table* value represents a Motif XmFontList. An XmFontList is a data type that specifies the fonts that are in use. Each entry in a font list specifies a font or a font set and an associated tag. When a Motif compound string (XmString) is displayed, the font list tag for the string is used to match the string with a font or a font set, so that the compound string is displayed appropriately.

In UIL, a font_table is represented literally by the symbol *font_table*, followed by a list of one or more *font* or *fontset* values. The elements of a *font_table* must be defined as private values. The character_set of an entry in the list can be overridden by preceding it with a predefined or user-defined *character_set* and an equal sign (=).

Usage

The *font_table* type can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an *argument* literal. A *font_table* is converted to an XmFontList at run-time by Mrm.

The UIL compiler may exit with a severe internal error if a user-defined character_set is used in a *font_table* that is exported or specified as a resource value. This situation can occur if *character_set* is specified directly or indirectly in one of the entries. If this problem occurs in your version of UIL, only predefined *character_set* values can be used in *font*, *fontset*, and *font_table* values. The workaround is to specify these problematic values in an X resource file.

Example

```
...
procedure
    switch_styles (font_table);

value
    latin1    : font ('*-iso8859-1', character_set = iso_latin1);
    hebrew    : font ('*-iso8859-8', character_set = iso_hebrew);
    list      : font_table (latin1, hebrew);
```

```
value
    XtNdefaultFonts : argument ('defaultFonts', font_table);

object label: XmLabel {
    arguments {
        XmNfontList = list;
    };
};
...
```

See Also

`XmFontListAppendEntry`(1), `XmFontListEntryCreate`(1), `XmFontListEntryLoad`(1), `character_set`(6), `font`(6), `fontset`(6).

Name

fontset – XFontSet type.

Synopsis

Syntax:

fontset (*string_expression* [, ...] [, character_set = *character_set*])

MrmType:

MrmRtypeFontSet

Description

A *fontset* value represents an XFontSet, which is an Xlib structure that specifies all of the fonts that are needed to display text in a particular locale. A fontset is represented literally by the symbol *fontset*, followed by a list of string expressions that evaluate to font names and an optional *character_set*. All parts of the string expressions that make up the list of font names must be private to the UIL module. The *character_set* is associated with the fontset if it appears in a *font_table*. If *character_set* is not specified, it is determined from the codeset portion of the LANG environment variable if it is set, or XmFALLBACK_CHARSET otherwise.

The string expression that specifies the font name is a list or wildcarded set of X Logical Font Description (XLFD) strings. This list is stored in the UID file and used as a parameter to XCreateFontSet() at run-time to load the font set. See Volume One, *Xlib Programming Manual*, and Volume Two, *Xlib Reference Manual*, for more information on fonts.

Usage

You can use a *fontset* value to specify a *fontset* or *font_table* resource. When a *fontset* is assigned to a *font_table* resource, at run-time Mrm automatically creates an XmFontList that contains only the specified fontset. A fontset value can also be used as an element in font_table, although in this context it must be private to the UIL module.

The *fontset* type can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an *argument* literal.

In some versions of UIL, the default character set is always ISO8859-1, instead of being based on the LANG environment variable or XmFALLBACK_CHARSET.

The UIL compiler may exit with a severe internal error if a user-defined *character_set* is used in a fontset that is exported or specified as a resource value. If this problem occurs in your version of UIL, only predefined *character_set* values can be used in *font*, *fontset*, and *font_table* values. The workaround is to specify these problematic values in an X resource file.

Example

```
procedure
    change_fontset (fontset);

value
    japanese_font : fontset ('-misc-fixed-*-75-75-*');
    default_font : imported font;

value
    XtNbodyFontSet : argument ('bodyFontSet', fontset);

object label: XmLabel {
    arguments {
        XmNfontList = japanese_font;
    };
};
```

See Also

`character_set`(6), `font`(6), `font_table`(6).

Name

icon – multi-color rectangular pixmap type.

Synopsis

Syntax:

icon ([color_table = *color_table_name* ,] *row* [, ...])

MrmType:

MrmRtypeIconImage

Description

An *icon* value represents a multi-color rectangular pixmap, or array of pixel values. An icon is represented literally by the symbol *icon*, followed by an optional *color_table* specification and a list of strings that represent the rows of pixel values in the *icon*.

If a *color_table* is specified, it must be a private value and cannot be forward referenced. If a *color_table* is not specified, the following default color_table is used:

color_table (background_color = ' ', foreground color = '*')

Each *row* in the *icon* is a character expression that represents a row of pixel values. Each character in the *row* represents a single pixel. All of the rows in the *icon* must be the same length and must contain only characters defined in the *color_table* for the *icon*. The UIL compiler generates an error if these rules are violated.

Usage

The type *icon* can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal. An *icon* can be retrieved by an application with `MrmFetchIconLiteral`() or `MrmFetchBitmapLiteral`().

When an *icon* is specified as a resource value for a widget, the depth of the pixmap created by Mrm at run-time is the same as the depth of the widget. When an *icon* is retrieved with `MrmFetchIconLiteral`(), the depth of the resulting pixmap is the value returned from the `DefaultDepthOfScreen`() macro. When an *icon* is retrieved with `MrmFetchBitmapLiteral`(), the depth of the resulting pixmap is always one. The *color_table* of an *icon* retrieved with this function must only contain mappings for background color and foreground color or the function fails and returns MrmNOT_FOUND.

The UIL compiler may not check the type of the value specified as the *color_table* for an *icon*. If the compiler allows the specification of a value that is

not a *color_table*, it generates an error message when the *icon* is referenced. If no reference to the *icon* occurs in the module, the compiler exits with a severe internal error.

If the *row* values in an *icon* literal do not consist entirely of string literals, the UIL compiler may generate an error message or crash with a segmentation violation.

If a named value is declared as an imported *icon* in one UIL module file, but defined with a different type in another, an error is generated at run time when Mrm attempts to retrieve the *icon*. If you attempt to define a named variable with the value of an *icon* variable, the UIL compiler may generate a large number of errors that are seemingly unrelated to the assignment.

Example

```
...
value
    ! Define an icon that uses default color table and can be retrieved
    ! as a resource or with any of the fetch procedures including
    ! MrmFetchBitmapLiteral():
    checker : icon ('* *',  ' * ',  '* *');
    ! Define an icon that uses a custom color table which contains named
    ! colors.  This icon cannot be retrieved with MrmFetchBitmapLiteral().
    red_blue : color_table (color('red') = 'r', color('blue') = 'b');
    plus : icon (color_table = red_blue, 'brb', 'rrr', 'brb');
    ! Declare an argument of type icon.
    XtNwmIcon : argument ('wmIcon', icon);

! Declare a procedure taking an icon parameter.
procedure
    display_icon (icon);

! Use an icon for a resource value
object label: XmLabel {
    arguments {
        XmNlabelType = XmPIXMAP;
        XmNlabelPixmap = plus;
    };
};
...
```

See Also

`MrmFetchBitmapLiteral`(3), `MrmFetchIconLiteral`(3), `MrmFetchSetValues`(3), `MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `color_table`(6), `pixmap`(6), `xbitmapfile`(6).

Name

integer – whole number type.

Synopsis

Syntax:

[+ | -]0-9[...]

MrmType:

MrmRtypeInteger

Description

An *integer* value represents a negative or positive whole number. An *integer* is represented literally by an optional sign followed by one or more consecutive digits.

An integer can also be represented literally by the symbol *integer* followed by a *float*, *single_float*, or *boolean* expression. The expression is converted to an *integer* and can be used in any context that an *integer* value is valid. An *integer* is formed from a *float* or *single_float* by truncating the fractional value. You can add 0.5 to the *float* or *single_float* value if rounding is desired. If a *float* or *single_float* larger (smaller) than MAXINT (-MAXINT) is converted to an *integer*, the resulting value is MAXINT (MININT). An *integer* is formed from a *boolean* by converting *true* and *on* to 1 and *false* and *off* to 0.

Usage

The allowable range of an *integer* value is determined by the size of an integer on the machine where the UIL module is compiled. Since an integer on most architectures is typically a minimum of four bytes, *integer* values may safely range from -2147483647 (-MAXINT) to 2147483647 (MAXINT). You can ensure greater portability by keeping *integer* values within the four-byte range. The UIL compiler generates an error if it encounters an *integer* outside of the machine's representable range.

The type *integer* can be used as the type of an imported value, as a parameter type in a *procedure* declaration, or as the type in an *argument* literal.

Widget resources of type Position (short) and Dimension (unsigned short) are specified as *integers* in UIL. As a result, the UIL compiler does not generate an error if an out-of-range value is assigned to such a resource. If the sizeof(short) is smaller than sizeof(int), part of the out-of-range value is truncated, which produces an undefined result. The part truncated depends on the C compiler and byte-ordering of the machine on which the UIL module is compiled. For maximum portability, Position values should be limited to the range -32768 to 32767 and Dimension values should be limited to the range 0 to 65536.

The UIL compiler uses -MAXINT to MAXINT, not MININT to MAXINT, as the allowable range for integers, which means that on an architecture with four-byte integers, the minimum *integer* value allowed is -2147483647, not -2147483648. The value MININT can be used, however, by converting a float smaller than -MAXINT to an *integer*.

Example

```
...
! Declare a procedure taking an integer value.
procedure
    set_speed (integer);

! Define some integer variables.
value
    meaning_of_life: 41;
    the_question    : imported integer;
    half_life       : meaning_of_life / 2;
    ten             : integer (10.75);
    round_factor    : 0.5;
    eleven          : integer (10.75 + round_factor);
    one             : integer (true);
    ! Generate MININT value by converting large negative float:
    minint          : integer (-3.0e30);

! Define an argument of type integer.
value
    XtNsize : argument ('size', integer);

object pb : XmPushButton {
    arguments {
        XmNleftOffset = -3;
    };
};
...
```

See Also

`boolean`(6), `float`(6), `integer_table`(6), `single_float`(6).

Name

integer_table – array of integers.

Synopsis

Syntax:

integer_table (*integer_expression* [, ...])

MrmType:

MrmRtypeIntegerVector

Description

An *integer_table*[1] value represents an array of integers. An integer_table is represented literally by the symbol *integer_table*, followed by a list of integer expressions.

Usage

The type name *integer_table* can be used as the type of an imported value, as a parameter type in a *procedure* declaration, or as the type in an *argument* literal. In Motif 1.2, the XmNselectionArray resource of the XmText and XmTextField widgets is the only built-in *integer_table* resource. When the resource is set, UIL automatically sets the XmNselectionArrayCount resource to the number of elements in the array. In Motif 2.0 and later, the XmNselectedPositions resource of the List, and the XmNdetailOrder resource of the Container are also built-in *integer_table* types. The XmNselectedPositionCount and XmNdetailOrderCount resources are automatically set by UIL respectively.

Unlike *asciz_string_table* and *compound_string_table* values, an *integer_table* is not NULL-terminated. As a result, you must either use *integer_table* values of a set length, include the length explicitly, or use a value to indicate the end of the array. The application code that uses the values must use the same conventions as the UIL module.

Example

```
...
value
    ! Define table with known number of elements (12).
    days: integer_table (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
    ! Define table with length as first element.
    grades : integer_table (5, 95, 87, 100, 92, 82);
    ! Define table with last element of MININT.
    end_of_table : integer (3.0e-30);
    ages : integer_table (25, 29, 29, 30, 32, end_of_table);
```

1.Erroneously given as *integer_type* in 1st edition.

```
! Declare a procedure taking an integer_table
procedure
    compute_average (integer_table);

! Declare an argument taking an integer table
value
    XtNdaysPerMonth : argument ('daysPerMonth', integer_table);
...
```

See Also

`integer`(6).

Name

keysym – character type.

Synopsis

Syntax:

keysym (*string_literal*)

MrmType:

MrmRtypeKeysym

Description

A *keysym* value is used to represent a single character. A keysym is represented literally by the symbol *keysym*, followed by a string value that contains exactly one character. If the string is a variable, it can be forward referenced and must be private to the UIL module.

Usage

A *keysym* value is typically used to specify a widget mnemonic resource, such as XmNmnemonic. The *keysym* type can be used as the type of an imported value, as a parameter type in a *procedure* declaration, or as the type in an *argument* literal.

When a *keysym* is retrieved by an application with `MrmFetchLiteral`(), the value argument returned is the character value of the *keysym*, not a pointer to the value like many other types.

The UIL compiler may not generate an error if the string expression in a *keysym* literal is more than one character long, but an error will be generated by Mrm at run-time. If an invalid *keysym* is specified as a resource value, the resource is not set. If the application attempts to retrieve an invalid *keysym* with `MrmFetchLiteral`(), MrmNOT_FOUND is returned.

Example

```
...
procedure
    set_keysym (keysym);

value
    d_key : keysym ('d');
    XtNquitKey : argument ('quitKey', keysym);

object the_button : XmPushButton
    arguments {
        XmNmnemonic = keysym ('b');
    };
```

```
};
...
```

See Also

`MrmFetchLiteral`(3).

Name

pixmap – generic icon or xbitmapfile type.

Synopsis

Syntax:

No literal syntax.

MrmType:

MrmRtypeIconImage or MrmRtypeXBitmapFile

Description

A *pixmap* value can be either an *icon* or *xbitmapfile*. In either case, the type specifies an array of pixel values. A *pixmap* does not have its own literal representation; a *pixmap* value is specified with either the *icon* or *xbitmapfile* literal syntax.

Usage

The type *pixmap* can be used as the type of an imported value, as a parameter type in a *procedure* declaration, or as the type in an *argument* literal. The purpose of the *pixmap* type is to allow either an *icon* or an *xbitmapfile* value to be imported, passed as a callback argument, or specified as a resource value.

Example

```
...
value
    ! Declare an imported pixmap that can be defined as an icon or xbitmapfile.
    stop_pixmap : imported pixmap;
    ! Declare an argument to which an icon or xbitmapfile can be assigned.
    XtNstipplePixmap : argument ('stipplePixmap', pixmap);

! Declare a procedure to which an icon or xbitmapfile can be passed.
procedure
    print_pixmap (pixmap);
...
```

See Also

`icon`(6), `xbitmapfile`(6).

Name

reason – user-defined callback type.

Synopsis

Syntax:

reason (*string_expression*)

MrmType:

none

Description

A *reason* value represents a user-defined callback. A reason is represented literally by the symbol *reason*, followed by a string expression that evaluates to the name of a callback. The name of the *reason* is assigned to the name member of the ArgList structure passed to `XtSetValues`(). The name is typically the name of a callback with the XmN or XtN prefix removed.

Usage

A user-defined callback can be used in the callbacks section of a UIL module for both built-in Motif widgets and user-defined widgets. While user-defined callbacks are typically assigned to a named variable in the value section, they can also be specified literally in the *arguments* section of an *object* definition. If you are defining arguments for a widget or widget set which is not predefined, you should define them as named variables in a separate UIL module that can be included by any module that uses the widget(s).

Reasons must be private values; they cannot be imported or exported. The UIL compiler allows imported and exported declarations, but it generates an error when the user-defined *reason* is used. Since *reason* values cannot be exported, they cannot be retrieved by an application.

The *reason* type can only be used to define callback resource types. The argument type is used to specify other user-defined resources.

Example

From *Xaw/Panner.uih*:

```
! Resources and definitions for the Athena Panner widget.
...
! Callback definitions
value
    XtNreportCallback = reason ('XtNreportCallback');
    ...
```

From *my_module.uil*:

```
include file 'Xaw/Panner.uih';

procedure
    panner_report();

object panner : user_defined procedure XawCreatePanner {
    callbacks {
        XtNreportCallback = procedure panner_report();
    };
};
...
```

See Also

`MrmRegisterClass`(3), `include`(5), `object`(5), `argument`(6).

Name

rgb – color specified with the values of red, green, and blue components.

Synopsis

Syntax:

rgb (*red_integer*, *green_integer*, *blue_integer*)

MrmType:

MrmRtypeColor

Description

The type *rgb* represents a color as a mixture of red, green, and blue values. An rgb value is represented literally by the symbol *rgb*, followed by a list of three integers that specify the red, green, and blue components of the color. The amount of each color component can range from 0 (0 percent) to 65,535 (100 percent). Mrm allocates rgb values with `XAllocColor`(). See Volume 1, *Xlib Programming Manual*, and Volume 2, *Xlib Reference Manual*, for more information on color allocation.

Usage

An *rgb* value or literal can be used anywhere a color value is expected: as a callback argument, as a resource value, or in a *color_table*. Unlike color values, it is not possible to specify a foreground or background fallback for *rgb* values. For this reason, and to maximize the number of shareable color cells, you should use named colors defined with the color type whenever possible.

If a color cannot be allocated, Mrm substitutes black, unless the color is specified as the background color or foreground color in a *color_table* and the foreground color or background color is already black. In this situation, white is substituted.

In Motif version 1.2.1, the color substitutions described above do not take place. When a color allocation fails for an rgb value specified directly or indirectly (in the color_table of an icon) the resource is not set. If the allocation fails in a call to `MrmFetchColorLiteral`() or `MrmFetchIconLiteral`(), MrmNOT_FOUND is returned. In Motif 2.1, `XBlackPixelOfScreen`() is used where `XAllocColor`() fails.

Note that the values that specify that red, green, and blue components cannot be integer expressions. The UIL compiler, however, does not generate an error if an integer expression is encountered; it silently replaces the expression with the value 0. In addition, the UIL compiler does not report an error if an integer specified for a color value is less than 0 or greater than 65,535. If any of the three components is out-of-range, the three values stored in the UID file are undefined.

Example

```
value
    white  : rgb (65535, 65535, 65535);
    orange : exported rgb (65535, 32767, 0);
    grape  : imported rgb;
    ctable : color_table (white = 'w, orange = 'o', grape = 'g');
    ....

object label : XmLabel {
    arguments {
        XmNforeground = rgb (0, 0, 32767);
        XmNbackground = orange;
    };
};
```

See Also

`MrmFetchColorLiteral`(3), `MrmFetchIconLiteral`(3), `MrmFetchSetValues`(3), `MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `color`(6), `color_table`(6), `icon`(6).

Name

single_float – single-precision floating point type.

Synopsis

Syntax:

single_float (*numeric_expression*)

MrmType:

MrmRtypeSingleFloat

Description

A *single_float* value represents a negative or positive single-precision floating point number. A single_float is represented literally by the symbol *single_float*, followed by a boolean, float or integer expression. The expression is converted to a *single_float* and can be used in any context in which a *single_float* value is valid. A *single_float* is formed from a boolean by converting true and on to 1.0 and false and off to 0.0. If a float expression is greater than (less than) the largest (smallest) representable float, the resulting *single_float* is +infinity (-infinity).

Usage

The type *single_float* can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal. A *single_float* value is used to save space, as the storage used by a *single_float* is usually less than that used by a float.

The allowable range of a *single_float* value is determined by the size of a C float on the machine where the UIL module is compiled. Since a float on most architectures is typically a minimum of four bytes, *single_float* values may safely range from 1.4013e-45 to 3.40282e+38 (positive or negative). You can ensure greater portability by keeping *single_float* values within the four-byte range.

Example

```
...
! Declare a procedure taking a single_float value.
procedure
    sqrt (single_float);

value
    avogadro   : single_float (6.023e+23);
    prime_rate : imported single_float;

! Define an argument of type single_float.
value
    XtNarea : argument ('area', single_float);
...
```

See Also

`boolean`(6), `float`(6), `integer`(6).

Name

string – NULL-terminated character string type.

Synopsis

Syntax:

[*#character_set*] "*character_expression"* or
'character_expression'

MrmType:

MrmRtypeChar8

Description

A *string* value represents a NULL-terminated single-byte, multi-byte, or wide-character string. A *string* literal is represented by either a double or single-quoted sequence of characters, that may be up to 2000 characters long. Newer versions of UIL may allow even longer strings. The type of quotes used to delimit a *string* literal determines how the string is parsed by the UIL compiler.

Both double and single-quoted strings can directly contain characters with decimal values in the range 32 to 126 and 160 to 255. Characters with values outside of the range can only be entered using the escape sequence \value\, where value represents the character code desired. To allow the easy specification of commonly-used non-printing characters codes, UIL recognizes the following escape sequences:

Character	Meaning
\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quote
\"	Double quote

A double-quoted string consists of an optional *character_set*, followed by a sequence of characters surrounded by a double quotes. Double-quoted strings cannot span multiple lines, but may contain the \n escape sequence. If a

character_set is specified, it precedes the string and is indicated by a pound sign (#). Either a built-in or user-defined *character_set* can be specified. If a *character_set* is not specified, the default character set of the module is used. The default *character_set* can be specified with the *character_set* option in the module header of a UIL module. If this option is not set, the default is determined from the codeset portion of the LANG environment variable if it is set, or XmFALLBACK_CHARSET otherwise.

If the UIL compiler is invoked with the -s option, double-quoted strings are parsed in the current locale. When UIL parses localized strings, escape sequences may be interpreted literally. You can avoid unexpected results by restricting the use of escape sequences to single quoted strings.

A single-quoted string consists of a sequence of characters surrounded by single quotes. Unlike double-quoted strings, single-quoted strings can span multiple lines by using a backslash (\) to indicate that the string is continued on the next line. The newline character following the backslash is not included in the string. The \n escape sequence should be used if an embedded newline is desired. The *character_set* of a single-quoted string defaults to the codeset portion of the LANG environment variable if it is set, or XmFALLBACK_CHARSET otherwise.

The parsing direction of either string variant is determined by the *character_set* of the string. A string that is parsed right-to-left is stored in the UID file in the reverse order that it appeared in the UIL source module. The parsing direction and *character_set* writing direction determine the order of individual characters when a string is printed or displayed. The writing direction of a string is generally the same as the parsing direction, unless explicitly overridden in a compound_string literal. The order of the characters in escape sequences is always the same, regardless of the parsing direction.

Usage

A single or double-quoted string value can be used anywhere a string or string expression is expected. A string expression can be a single string value or two or more string values concatenated with the string concatentation operator (&). A string or string expression can also be used anywhere a *compound_string* is expected, since the UIL compiler automatically converts the string to a compound string, with the character set determined by the rules described above. (When determining the character set, the UIL compiler may use ISO8859-1 as the fallback character set, even if the value has been changed by the vendor. Therefore, you should specify a character set explicitly instead of relying on XmFALLBACK_CHARSET.)

Any newline characters in a NULL-terminated string that is converted into a *compound_string* are not converted into separator components to make a multi-line compound string. If you need a multi-line compound string, it must be specified as a concatenated set of values using the compound_string literal syntax with the separate property set to true.

The type *string* can be used as the type of an imported value, as a parameter type in a *procedure* declaration, or as the type in an *argument* literal. String values used in string expressions or in *compound_string* literals must be private to the module in which they are used.

Example

```
...
procedure
    tie_knot (string);

value
    display         : imported string;
    skit_name       : 'Unfrozen Caveman Lawyer';
    hello           : #iso_hebrew"\237\229\236\249\";
    quote           : exported 'Quoth the Raven, 'Nevermore.\'\n';
    concat          : 'The Cat' & ' in the Hat';
    multi           : 'All that we see or seem\nIs but a dream within a dream.';
    ! Define a resource of type string.
    XtNfilename     : argument ('filename', string);

object play : XmPushButton {
    arguments {
        ! String automatically converted to XmString
        XmNlabelString = skit_name;
    };
};
...
```

See Also

`asciz_string_table`(6), `character_set`(6), `compound_string`(6), `compound_string_table`(6).

Name

translation_table – Xt translation table type.

Synopsis

Syntax:

translation_table (['#override' | '#augment' | '#replace'] *string_expression* [, ...])

MrmType:
MrmRtypeTransTable

Description

A *translation_table* value represents an X Toolkit translation table. A translation table is a list of translations, where each translation maps an event or an event sequence to an action name. In UIL, a translation_table is represented literally by the symbol *translation_table*, followed by an optional directive and list of string expressions that are interpreted as translations. If specified, the directive must be one of #override, #augment, or #replace. The translations are specified as a list of string expressions, one per translation. The individual translations are concatenated and separated with newline characters before they are stored in the UID file.

Usage

The *translation_table* type can be used as the type of an imported value, as a parameter type in a *procedure* declaration, or as the type in an *argument* literal.

The syntax of a *translation_table* is not verified by the UIL compiler. Instead, Mrm converts a translation_table literal to an XtTranslations value with `XtParseTranslationTable`() at run-time. Errors that occur when parsing the *translation_table* are passed to `XtWarning`(). Because `XtParseTranslationTable`() always returns a valid XtTranslations value, even when parsing errors occur, the run-time conversion of a *translation_table* cannot fail. See Volume 4, *X Toolkit Intrinsics Programming Manual*, and Volume 5, *X Toolkit Intrinsics Reference Manual*, for more information about translation tables.

Example

```
...
procedure
    set_translations (translation_table);
    exit();

value
    XtNquickKeys : argument ('translations', translation_table);
```

```
value
    quit_tt : translation_table ('#override', '<Key>q: ArmAndActivate()');
    other_tt : imported translation_table;

object quit : XmPushButton {
    arguments {
        XmNtranslations = quit_tt;
    };
    callbacks {
        XmNactivateCallback = procedure exit();
    };
};
...
```

See Also

`MrmFetchLiteral`(3).

Name

wide_character – wide-character string type.

Synopsis

Syntax:

wide_character (*string_expression*)

MrmType:

MrmRtypeWideCharacter

Description

A *wide_character* value represents a wide-character string. The corresponding C type is wchar_t *. A wide_character literal is represented by the symbol *wide_character,* followed by a string expression.

Usage

A *wide_character* literal is used to make the UIL compiler parse a regular character string as a wide-character string. A *wide_character* string is parsed with the `mbstowcs`() function. The operation of this function depends on the setting of the locale. See the uil reference page for more information regarding the locale setting. The *wide_character* literal syntax may not work in early releases of Motif 1.2. However, you can specify a wide-character string using the normal UIL string syntax. The difference is that the UIL compiler does not verify that a wide-character string specified in this way is properly formed.

The type *wide_character* can also be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal.

Example

```
...
procedure
    print_wcs (wide_character);

value
    wcs : wide_character ('\204\176\224\189\');
    name : imported wide_character;
    XtNwideCharacterString : argument ('wideCharacterString',
    wide_character);

object text : XmText {
    arguments {
        XmNvalueWcs = wcs;
    };
};
...
```

See Also

`uil`(4), `procedure`(5), `argument`(6), `string`(6).

Name

widget – widget type.

Synopsis

Syntax:

See the object section of the UIL file format reference page.

MrmType:

none

Description

Objects that are declared or defined in a UIL object section are of type *widget*. Values of type *widget* are the only UIL values that are not declared or defined in a value section. The literal representation of a *widget* is described in the object section of the UIL file format reference page.

Usage

The type *widget* can be used as a parameter type in a *procedure* declaration or as the type in an *argument* literal. When a widget is used as a callback parameter or resource value in the declaration of another widget, it must be part of the same hierarchy as that widget. A widget hierarchy is defined by the widget passed to `MrmFetchWidget`() or `MrmFetchWidgetOverride`() and it includes all of the descendants of that widget. If you need to specify a widget in a different hierarchy as a callback parameter, you can use the string name of the widget instead and convert it to a widget pointer in the callback with `XtNameToWidget`().

Widgets can be forward referenced. If Mrm encounters a reference to a widget that has not been created in the current hierarchy, it creates the remainder of the hierarchy and makes another attempt to resolve the reference. If the reference cannot be resolved at that point, Mrm does not add the callback or set the resource for which the widget is specified. As of Motif 1.2, Mrm does not generate a warning when a widget reference cannot be resolved.

Prior to Motif 1.2.1, the UIL compiler generates an error when widget is used as a *procedure* parameter or type in an *argument* literal. To work around this problem, you can use the type *any.*

Example

```
...
value
    ! Declare Athena tree widget constraint argument.
    XtNtreeParent : argument ('treeParent', widget);
```

```
procedure
    manage (widget);

object
    button1 : XmPushButton {
        callbacks {
            XmNactivateCallback = manage (button3);
        }
        arguments {
            XmNbottomAttachment = XmATTACH_FORM;
            XmNbottomOffset = 40;
            XmNrightAttachment = XmATTACH_WIDGET;
            XmNrightWidget = button1;
        };
    };
    button2 : XmPushButton { };
    button3 : XmPushButton {
        arguments {
            XmNbottomAttachment = XmATTACH_FORM;
        };
    };
    form : XmForm {
        controls {
            XmPushButton button1;
            XmPushButton button2;
            unmanaged XmPushButton button3;
        };
    };
...
```

See Also

`MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `object`(5), `procedure`(5), `argument`(6).

Name

xbitmapfile – X bitmap file type.

Synopsis

Syntax:

xbitmapfile (*string_expression*)

MrmType:

MrmRtypeXBitmapFile

Description

An *xbitmapfile* value represents a file that contains a bitmap in the standard X bitmap file format. An xbitmapfile literal is represented by the symbol *xbitmapfile*, followed by a string expression that evaluates to the name of the file containing the bitmap. The X bitmap is loaded at run-time by Mrm using `XmGetPixmapByDepth`(). See Volume 1, *Xlib Programming Manual*, for more information about the X bitmap file format.

Usage

The type *xbitmapfile* can be used as the type of an imported value, as a parameter type in a procedure declaration, or as the type in an argument literal. An *xbitmapfile* value can be retrieved by an application with `MrmFetchIconLiteral`(). The `MrmFetchBitmapLiteral`() procedure cannot be used to retrieve values of the *xbitmapfile* type.

When an *xbitmapfile* is specified as a resource value for a widget, the depth of the pixmap created by Mrm at run-time is the same as the depth of the widget. When an *xbitmapfile* is retrieved with `MrmFetchIconLiteral`(), the depth of the resulting pixmap is the value returned from the `DefaultDepthOfScreen`() macro.

The UIL compiler stores the specified file name in the UID output file, not the X bitmap to which the name refers. The compiler does not verify that the specified file exists. If an *xbitmapfile* specified as a resource cannot be loaded, the resource is not set. If MrmFetchIconLiteral fails to load an *xbitmapfile*, MrmNOT_FOUND is returned.

Example

```
...
! Declare a bitmap of the most challenging ski slope in the Northeast.
value
    goat: xbitmapfile ('goat.xbm');
```

```
object scary : XmLabel {
    arguments {
        XmNlabelType = XmPIXMAP;
        XmNlabelPixmap = goat;
    };
};
...
```

See Also

`XmGetPixmapByDepth`(2), `MrmFetchBitmapLiteral`(3), `MrmFetchIconLiteral`(3), `MrmFetchWidget`(3), `MrmFetchWidgetOverride`(3), `icon`(6), `pixmap`(6).

Section 7 - UIL Functions

This page describes the format and contents of each reference page in Section 7, which covers the User Interface Language (UIL) functions.

Name

Function – a brief description of the function.

Synopsis

This section shows the signature of the function: the names and types of the arguments, and the type of the return value. The header file *<uil/UilDef.h>* declares both of the public UIL functions.

Inputs

This subsection describes each of the function arguments that pass information to the function.

Outputs

This subsection describes any of the function arguments that are used to return information from the function. These arguments are always of some pointer type, so you should use the C address-of operator (**&**) to pass the address of the variable in which the function will store the return value. The names of these arguments are sometimes suffixed with *_return* to indicate that values are returned in them. Some arguments both supply and return a value; they will be listed in this section and in the "Inputs" section above. Finally, note that because the list of function arguments is broken into "Input" and "Output" sections, they do not always appear in the same order that they are passed to the function. See the function signature for the actual calling order.

Returns

This subsection explains the return value of the function, if any.

Description

This section explains what the function does and describes its arguments and return value. If you've used the function before and are just looking for a refresher, this section and the synopsis above should be all you need.

Usage

This section appears for most functions and provides less formal information about the function: when and how you might want to use it, things to watch out for, and related functions that you might want to consider.

Example

This section provides an example of the use of the function.

Structures

This section shows the definition of any structures, enumerated types, typedefs, or symbolic constants used by the function.

Procedures

This section shows the syntax of any prototype procedures used by the function.

See Also

This section refers you to related functions, clients, and UIL data types. The numbers in parentheses following each reference refer to the sections of this book in which they are found.

Name

Uil – call the UIL compiler from 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)
```

Inputs

command_desc	Specifies a structure containing the compilation options.
message_cb	Specifies a callback function that is called when error, warning and informational messages are generated by the compiler.
message_data	Specifies data that is passed to the message_cb function.
status_cb	Specifies a callback function that is called periodically during the compilation to indicate progress.
status_data	Specifies data that is passed to the status_cb function.

Outputs

compile_desc	Returns a structure containing the results of the compilation.

Returns

Uil_k_success_status on success and if no problems are detected,
Uil_k_info_status on success and if informational messages are generated,
Uil_k_warning_status on success and if warning messages are generated,
Uil_k_error_status on failure and if error messages are generated, and
Uil_k_severe_status on failure and if the compilation stopped prematurely.

Description

`Uil`() invokes the UIL compiler from within an application. Options for the compiler, including the input, output and listing files, are provided in the *command_desc* argument. The calling application can supply a message handling function in *message_cb* that displays compiler messages in an application-defined manner. The application can also supply a status-monitoring function in *status_cb*. This function is called periodically by the compiler to report progress. Upon completion, the `Uil`() function fills in the *compile_desc* structure with information about the compilation and returns the status of the compilation.

Usage

An application that calls `Uil`() is responsible for allocating the *command_desc* and *compile_desc* arguments. The application must initialize all members of the

command_desc structure. Members of the *compile_desc* structure are set by the compiler. If the *parse_tree_flag* in *command_desc* is set, the compiler returns a pointer to the root of the parse tree in the *parse_tree_root* field of the *compile_desc*. This parse table cannot be freed by the calling application. Therefore, you should not set the *parse_tree_flag* unless you plan to use the parse tree. To limit memory consumption, if you set the *parse_tree_flag*, invoke the `Uil`() routine once and exit soon thereafter.

An application can specify a function for handling compiler generated messages in the *message_cb* argument. You can specify NULL for this argument if you want to use the default message handling routine. This routine prints all messages to *stderr*. If you specify a function, the value of *message_data* is passed to each invocation of the function.

An application can also specify a function for monitoring the status of the compilation in the *status_cb* argument. You can specify NULL to indicate that no status function should be called. If you specify a function, the value of *status_data* is passed to each invocation of the function. In addition to monitoring progress, the function can also be used to process X events in an X application.

The `Uil`() function installs signal handlers for SIGBUS, SIGSYS, and SIGFPE with no regard for application installed handlers. These installed handlers remain set after the function returns, so you may wish to change them.

Applications that call the `Uil`() function must be linked with the UIL library, *libUil.a*, in addition to the Mrm, Motif, Xt, and X libraries.

Structures

The *Uil_command_type* is defined as follows:

```
typedef struct {
    char            *source_file;               /* name of UIL source file */
    char            *resource_file;             /* name of UID output file */
    char            *listing_file;              /* name of listing file */
    unsigned int    include_dir_count;          /* length of include_dir array */
    char            **include_dir;          /* array of include file directories */
    unsigned int    listing_file_flag : 1;      /* write listing file flag */
    unsigned int    resource_file_flag : 1;     /* write UID file flag */
    unsigned int    machine_code_flag : 1;      /* write machine code flag */
    unsigned int    report_info_msg_flag : 1;   /* report informational mes-
    sages */
    unsigned int    report_warn_msg_flag : 1; /* report warning messages */
    unsigned int    parse_tree_flag : 1;        /* generate parse tree flag */
    unsigned int    issue_summary : 1;      /* write diagnostic summary flag */
    unsigned int    status_update_delay;    /* delay between status_cb calls */
```

```
    char            *database;              /* WML database filename */
    unsigned int    database_flag : 1;      /* read WML database flag */
    unsigned int    use_setlocale_flag : 1; /* parse strings in locale flag */
} Uil_command_type;
```

Uil_command_type describes the compilation options for the `Uil`() routine. *source_file* is the name of the UIL module to compile. *resource_file* is the name of the UID file that is output if *resource_file_flag* is set. *listing_file* is the name of the compilation listing file that is output if *listing_file_flag* is set. Setting *machine_code_flag* causes the compiler to output a binary description of the UID file when a listing is generated.

include_dir specifies an array of *include_dir_count* directory names that the compiler searches for UIL include files. If set, *report_info_msg_flag*, *report_warn_msg_flag*, and *issue_summary* cause the compiler to generate informational messages, warning messages, and a summary message, respectively.

If *parse_tree_flag* is set, it instructs the compiler to return a pointer to the parse tree of the module in the *compile_desc* structure. *status_update_delay* specifies how many status check points must be passed before the *status_cb* callback is called. If the field is set to zero, the function is called at every check point.

use_setlocale_flag directs the UIL compiler to parse double-quoted strings in the current locale. (See the UIL string type man page for more information.) *database* specifies the name of a Widget Meta-Language (WML) description file that the compiler loads if *database_flag* is set.

The *Uil_compile_desc_type* is defined as follows:

```
typedef struct _Uil_comp_desc {
    unsigned int    compiler_version;   /* UIL compiler version */
    unsigned int    data_version;       /* UIL structures version */
    char            *parse_tree_root;   /* parse tree for module */
    unsigned int    message_count[];    /* status messages counts */
} Uil_compile_desc_type;
```

Uil_compile_desc_type describes the return data for the `Uil`() routine. *compiler_version* specifies the version of the UIL compiler, while *data_version* specifies the version of the structures used by the compiler. If *parse_tree_flag* is set in the *command_desc* argument, *parse_tree_root* contains a pointer to a compiler-generated parse tree if the compilation succeeds. *message_count* is an array of integers that contains the number of each type of compiler message generated by the routine. Valid indices to the array are Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, and Uil_k_severe_status.

Procedures

A message_cb function has the following syntax:

```
Uil_continue_type *message_cb (char   *message_data,
                               int    message_number,
                               int    severity,
                               char   *message_string,
                               char   *source_text,
                               char   *column_string,
                               char   *location_string,
                               int    message_count[])
```

A *message_cb* function takes eight arguments. The first argument, *message_data*, is the value of the *message_data* argument passed to the `Uil`() function. *message_number* is the internal index of the message, which is used by the UIL compiler. *severity* specifies the severity of the message, which is one of Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, or Uil_k_severe_status.

message_string is a string describing the problem. *source_text* is a copy of the source line to which the message refers, with a tab character prepended. If the source line is not available, *source_text* is the empty string. *column_string* is a string that consists of a leading tab character followed by zero or more spaces and an * (asterisk) in the same column as the problem in the source line. This string is suitable for printing beneath *message_string* to indicate the location of the problem. If the column that contains the error or the source line is not available, *column_string* is the empty string.

location_string describes the location where the problem occurred. The format of this string is "\t\t line: %d file: %s" if both source and column number are available, or if no column number is available. If the column number, but no source line is available, the format is "\t\t line: %d position: %d file: %s". If the location is unavailable, the value of *location_string* is the empty string. If an application does not specify a *message_cb* routine, the compiler prints *source_text*, *column_string*, *message_string*, and *location_string* in that order.

message_count is an array of integers that contains the number of each type of compiler message generated by the routine so far. Valid indices to the array are Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, and Uil_k_severe_status.

A *message_cb* function should return Uil_k_continue if the compilation can continue or Uil_k_terminate if the compilation should be terminated.

A status_cb function has the following format:

```
Uil_continue_type *status_cb (   char   *status_data,
                                 int    percent_complete,
                                 int    lines_processed,
                                 char   *current_file,
                                 int    message_count[])
```

A *status_cb* function takes five parameters. The first argument, *status_data*, is the value of the *status_data* argument passed to the `Uil`() function. *percent_complete* specifies an estimate of the percentage of the compilation that has been completed. The value of this field falls within a fixed range of values for each step of the compilation. The value ranges from 0 to 50 while *source_file* is being parsed, from 60 to 80 while the *resource_file* is written, and from 80 to 100 while the *listing_file* is generated. Some versions of the UIL compiler may only report percent-complete values on the boundaries of these ranges. *lines_processed* indicates the number of lines that have been read from the input file.

When the UIL compiler is invoked, it parses the *source_file*, writes the resource_file, and then generates the *listing_file*, based on the settings of the *command_desc* argument. The *current_file* field changes to reflect the file that the compiler is accessing.

message_count is an array of integers that contains the number of each type of compiler message generated by the routine so far. Valid indices to the array are Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, and Uil_k_severe_status.

A *status_cb* function should return Uil_k_continue if the compilation can continue or Uil_k_terminate if the compilation should be terminated.

The frequency with which the compiler calls the *status_cb* function at check points is based on the value of *status_update_delay* field in *command_desc*. A check point occurs every time a symbol is found during the parsing of *source_file*, every time an element is written to the *resource_file*, and every time a line is written to the *listing_file*.

Example

The following routines illustrate the use of the `Uil`() routine in a very basic way:

```
#include <uil/UilDef.h>
#include <stdio.h>

static char *last_current_file;
static char *status_string_list[Uil_k_max_status] = { NULL };
```

```
Uil_continue_type message_cb ( char    *message_data,
                               int     message_number,
                               int     severity,
                               char *message_string,
                               char    *line_text,
                               char    *error_col_string,
                               char    *line_and_file_string,
                               int     *message_count)
{
    if (*line_text != ' ')
        puts (line_text);
    if (*error_col_string != ' ')
        puts (error_col_string);
    if (*message_string != ' ')
        printf ("%s: %s\n", status_string_list[severity], message_string);
    if (*line_and_file_string != ' ')
        puts (line_and_file_string);

    return (Uil_k_continue);
}

Uil_continue_type status_cb ( char    *status_data,
                              int     percent_complete,
                              int     lines_processed,
                              char    *current_file,
                              int     *message_count)
{
    if (last_current_file == NULL || strcmp (last_current_file, current_file) != 0)
    {
        fprintf (stderr, "Working on file %s...\n", current_file);
        last_current_file = current_file;
    }

    return (Uil_k_continue);
}

Uil_compile_desc_type * compile (char *filename)
{
    Uil_command_type                command_desc;
    static Uil_compile_desc_type    compile_desc;
    Uil_status_type                 status;

    if (status_string_list[Uil_k_success_status] == NULL)  {
        status_string_list[Uil_k_success_status]       = "Success";
```

```
        status_string_list[Uil_k_info_status]       = "Informational";
        status_string_list[Uil_k_warning_status]    = "Warning";
        status_string_list[Uil_k_error_status]      = "Error";
        status_string_list[Uil_k_severe_status]     = "Severe Error";
    }
    command_desc.source_file                = filename;
    command_desc.resource_file              = "a.uid";
    command_desc.listing_file               = "uil.lst";
    command_desc.include_dir_count          = 0;
    command_desc.include_dir                = NULL;
    command_desc.listing_file_flag          = TRUE;
    command_desc.resource_file_flag         = TRUE;
    command_desc.machine_code_flag          = FALSE;
    command_desc.report_info_msg_flag       = TRUE;
    command_desc.report_warn_msg_flag       = TRUE;
    command_desc.parse_tree_flag            = FALSE;
    command_desc.issue_summary              = TRUE;
    command_desc.status_update_delay        = 0;
    command_desc.database                   = NULL;
    command_desc.database_flag              = FALSE;
    command_desc.use_setlocale_flag         = FALSE;

    last_current_file                       = NULL;

    status = Uil (&command_desc, &compile_desc, message_cb, NULL,
    status_cb, NULL);

    if (status == Uil_k_error_status || status == Uil_k_severe_status)
        return (NULL);

    return (&compile_desc);
}

int main (int argc, char **argv)
{
    Uil_compile_desc_type *compile_desc;

    if (argc != 2) {
        printf ("usage: Uil filename\n");
        exit (1);
    }

    compile_desc = compile (argv[1]);

    if (compile_desc != NULL)
        fprintf (stderr, "Compilation Successful.\n");
```

```
    else
        fprintf (stderr, "Compilation Failed.\n");
}
```

See Also

`uil`(4), `string`(6), `UilDumpSymbolTable`(7).

Name

UilDumpSymbolTable – produce a listing of a UIL symbol table.

Synopsis

#include <uil/UilDef.h>

void UilDumpSymbolTable (sym_entry_type **parse_tree_root*)

Inputs

parse_tree_root Specifies a pointer to the root entry of a symbol table.

Description

`UilDumpSymbolTable`() prints a listing of the symbols parsed in a UIL module to stdout. A parse tree is generated by a call to `Uil`(). If the *parse_tree_flag* of the *Uil_command_type* structure passed to the routine is set and the compilation is successful, the `Uil`() routine returns a pointer to the root of the parse tree in the *parse_tree_root* member of the *Uil_compile_desc_type* structure. If the compilation is unsuccessful, the *parse_tree_root* field is set to NULL.

Usage

`UilDumpSymbolTable`() generates a listing of the internal representation of UIL structures and symbols, which is really only useful for people who are quite familiar with the internals of the UIL compiler. The -m option of the `uil` command, or the *machine_code_flag* option of the `Uil`() routine, generates far more useful information for most users of UIL.

Instead of calling `UilDumpSymbolTable`(), an application can examine the parse tree directly. The structures used in the parse tree are defined in the file *<uil/UilSymDef.h>* and definitions of constants used in the structures are in *<uil/UilDBDef.h>*. Both of these files are included by *<uil/UilDef.h>*.

The parse table generated by the `Uil`() routine cannot be freed by the calling application. Therefore, you should not set the *parse_tree_flag* unless you plan to use the parse tree. To limit memory consumption, if you set the *parse_tree_flag*, invoke the `Uil`() routine once and exit soon thereafter.

See Also

`uil`(4).
`Uil`(7).

Appendix A - Function Summaries

This quick reference is intended to help you find and use the right function for a particular task. It organises the Section 1 and Section 3 reference pages into two lists:

- List of functions and macros by groups.
- Alphabetical list of functions and macros

The first column indicates which section to find the routines in; the required Section is given in brackets after the function name.

A.1 Group Listing with Brief Descriptions

Atoms

XmGetAtomName(1)	Get the string representation of an Atom
XmInternAtom(1)	Return an Atom for a given property name string

CascadeButton

XmCascadeButtonGadgetHighlight(1)	Set the highlight state of a CascadeButtonGadget
XmCascadeButtonHighlight(1)	Set the highlight state of a CascadeButton

Clipboard

XmClipboardBeginCopy(1)	Set up storage for a clipboard copy operation
XmClipboardCancelCopy(1)	Cancel a copy operation to the clipboard
XmClipboardCopy(1)	Copy a data item to temporary storage for later copying to the clipboard
XmClipboardCopyByName(1)	Copy as data item passed by name
XmClipboardEndCopy(1)	End a copy operation to the clipboard
XmClipboardEndRetrieve(1)	End a copy operation from the clipboard
XmClipboardInquireCount(1)	Get the number of data item formats available on the clipboard
XmClipboardInquireFormat(1)	Get the specified clipboard data format name
XmClipboardInquireLength(1)	Get the length of the data item on the clipboard

XmClipboardInquirePendingItems(1)	Get a list of pending data ID/private ID pairs
XmClipboardLock(1)	Lock the clipboard
XmClopboardRegisterFormat(1)	Register a new format for clipboard data items
XmClipboardRetrieve(1)	Retrieve a data item from the clipboard
XmClipboardStartCopy(1)	Set up storage for a clipboard copy operation
XmClipboardStartRetrieve(1)	Start a clipboard retrieval operation
XmClipboardUndoCopy(1)	Remove the last item copied to the clipboard
XmClipboardUnlock(1)	Unlock the clipboard
XmClipboardWithdrawFormat(1)	Indicate that an application does not want to supply a data item any longer

Colors

XmChangeColor(1)	Update the colors of a widget
XmGetColorCalculation(1)	Get the procedure that calculates default widget colors
XmGetColors(1)	Get the foreground, select, and shadow colors for a widget
XmSetColorCalculation(1)	Set the procedure that calculates default widget colors

ComboBox

XmComboBoxAddItem(1)	Add a compound string to the ComboBox List
XmComboBoxDeletePos(1)	Delete an item at the specified position from a ComboBox List
XmComboBoxSelectItem(1)	Select an item from a ComboBox List
XmComboBoxSetItem(1)	Select and make visible an item from a ComboBox List
XmComboBoxUpdate(1)	Update the ComboBox List after changes to component widgets

Command

XmCommandAppendValue(1)	Append a compound string to a Command widget
XmCommandError(1)	Display an error message in a Command widget

XmCommandGetChild(1)	Get the specified child of a Command widget
XmCommandSetValue(1)	Replace the Command string

Compound Strings

XmCvtByteStreamToXmString()	Convert a byte stream to a compound string
XmCvtCTToXmString(1)	Convert compound text to a compound string
XmCvtTextPropertyToXmStringTable(1)	Convert an XTextProperty to a compound string table
XmCvtXmStringTableToTextProperty(1)	Convert a compound string table to an XTextProperty
XmCvtXmStringToByteStream(1)	Convert a compound string to byte stream format
XmCvtXmStringToCT(1)	Convert a compound string to compound text
XmMapSegmentEncoding(1)	Get the compound text encoding format for a font list element tag
XmRegisterSegmentEncoding(1)	Register a compound text encoding format for a font list element tag
XmStringBaseline(1)	Get the baseline spacing for a compound string
XmStringByteCompare(1)	Compare two compound strings byte-by-byte
XmStringByteStreamLength(1)	Calculate the length of a byte stream
XmStringCompare(1)	Compare two compound strings
XmStringComponentCreate(1)	Create a compound string component
XmStringConcat(1)	Concatenate two compound strings
XmStringConcatAndFree(1)	Create a new compound string formed by concatenating two compound strings, freeing the two strings afterwards
XmStringCopy(1)	Copy a compound string
XmStringCreate(1)	Create a compound string
XmStringCreateLocalized(1)	Create a compound string in the current locale
XmStringCreateLtoR(1)	Create a compound string
XmStringCreateSimple(1)	Create a compound string in the current language environment
XmStringDirectionCreate(1)	Create a compound string containing a direction component

XmStringDirectionToDirection(1)	Convert a string direction to a direction
XmStringDraw(1)	Draw a compound string
XmStringDrawImage(1)	Draw a compound string
XmStringDrawUnderline(1)	Draw a compound string with an underlined sub-string
XmStringEmpty(1)	Determine whether there are any text components in a compound string
XmStringExtent(1)	Get the smallest rectangle enclosing a compound string
XmStringFree(1)	Free the memory used by a compound string
XmStringFreeContext(1)	Free a compound string context
XmStringGenerate(1)	Generate a compound string
XmStringGetLtoR(1)	Get a text segment from a compound string
XmStringGetNextComponent(1)	Retrieves information about the next compound string component
XmStringGetNextSegment(1)	Retrieves information about the next compound string component
XmStringGetNextTriple(1)	Retrieve information about the next compound string segment
XmStringHasSubstring(1)	Determine whether a compound string contains a sub-string
XmStringHeight(1)	Get the line height of a compound string
XmStringInitContext(1)	Create a compound string context
XmStringIsVoid(1)	Determine whether there are valid segments in a compound string
XmStringLength(1)	Get the length of a compound string
XmStringLineCount(1)	Get the number of lines in a compound string
XmStringNConcat(1)	Concatenate a specified portion of a compound string to another compound string
XmStringNCopy(1)	Copy a specified portion of a compound string
XmStringParseText(1)	Convert a string to a compound string
XmStringPeekNextComponent(1)	Returns the type of the next compound string component
XmStringPutRendition(1)	Add rendition components to a compound string
XmStringSegmentCreate(1)	Create a compound string segment

XmStringSeparatorCreate(1)	Create a compound string containing a separator component
XmStringTableParseStringArray(1)	Convert an array of strings into a compound string table
XmStringTableProposeTabList(1)	Create a Tab list for a compound string
XmStringTableToXmString(1)	Convert a compound string table to a compound string
XmStringTableUnparse(1)	Convert a compound string table to a string
XmStringToXmStringTable(1)	Convert a compound string to a compound string table
XmStringUnparse(1)	Convert a compound string to a string
XmStringWidth(1)	Return the width of a compound string

Container

XmContainerCopy(1)	Copy the Container primary selection to the clipboard
XmContainerCopyLink(1)	Copy links to the Container primary selection into the clipboard
XmContainerCut(1)	Cuts the Container primary selection onto the clipboard
XmContainerGetItemChild(1)	Find the children of a Container item
XmContainerPaste(1)	Pastes the clipboard selection into a Container
XmContainerPasteLink(1)	Copies links from the clipboard selection into a Container
XmContainerRelayout(1)	Force relayout of a Container widget
XmContainerReorder(1)	Reorder children of a Container

Cursors

XmGetMenuCursor(1)	Get the current menu cursor
XmSetMenuCursor(1)	Set the current menu cursor

Direction

XmDirectionMatch(1)	Compare two directions
XmDirectionMatchPartial(1)	Partially compare two directions
XmDirectionToStringDirection(1)	Convert a direction to a string direction
XmStringDirectionToDirection(1)	Convert a string direction to a direction

Display

XmGetXmDisplay(1)	Get the XmDisplay object for the Display
XmUpdateDisplay(1)	Update the display

Drag and Drop

XmDragCancel(1)	Cancel a drag operation
XmDragStart(1)	Start a drag operation
XmDropSiteConfigureStackingOrder(1)	Change the stacking order of a drop site
XmDropSideEndUpdate(1)	End an update of multiple drop site operations
XmDropSiteQueryStackingOrder(1)	Get the stacking order of a drop site
XmDropSiteRegister(1)	Register a widget as a drop site
XmDropSiteRetrieve(1)	Get the resource values of a drop site
XmDropSiteStartUpdate(1)	Start an update of multiple drop site operations
XmDropSiteUnregister(1)	Remove a drop site
XmDropSiteUpdate(1)	Change the resource values for a drop site
XmDropTransferAdd(1)	Add drop transfer entries to a drop operation
XmDropTransferStart(1)	Start a drop operation
XmGetDragContext(1)	Get information about a drag-and-drop operation
XmTargetsAreCompatible(1)	Determine whether or not the target types of a drag source and a drop site match

FileSelectionBox

XmFileSelectionBoxGetChild(1)	Get a specified child of a FileSelectionBox widget
XmFileSelectionDoSearch(1)	Start a directory search

Font Lists

XmFontListAdd(1)	Create a new font list
XmFontListAppendEntry(1)	Append a font entry to a font list
XmFontListCopy(1)	Copy a font list
XmFontListCreate(1)	Create a new font list
XmFontListCreate_r(1)	Create a new font list in a thread-safe manner

XmFontListEntryCreate(1)	Create a new font list entry
XmFontListEntryCreate_r(1)	Create a new font list entry in a thread-safe manner
XmFontListEntryFree(1)	Free the memory used by a font list entry
XmFontListEntryGetFont(1)	Get the font information from a font list entry
XmFontListEntryGetTag(1)	Get the tag of a font list entry
XmFontListEntryLoad(1)	Load a font or create a font set, and then create a font list entry
XmFontListFree(1)	Free the memory used by a font list
XmFontListFreeFontContext(1)	Free a font context
XmFontListGetNextFont(1)	Retrieve information about the next font list element
XmFontListInitFontContext(1)	Create a font context
XmFontListNextEntry(1)	Retrieve the next font list entry in a font list
XmFontListRemoveEntry(1)	Remove a font list entry from a font list
XmSetFontUnit(1)	Set the font unit values
XmSetFontUnits(1)	Set the font unit values

Keyboard Handling

XmTranslateKey(1)	Convert a KeyCode to a KeySym using the default translator

Keyboard Traversal

XmAddTabGroup(1)	Add a widget to a tab group list
XmGetDestination(1)	Get the current destination widget
XmGetFocusWidget(1)	Get the widget that has the keyboard focus
XmGetTabGroup(1)	Get the tab group for a widget
XmGetVisibility(1)	Determine whether or not a widget is visible
XmIsTraversable(1)	Determine whether or not a widget can receive the keyboard focus
XmProcessTraversal(1)	Set the widget which has the keyboard focus
XmRemoveTabGroup(1)	Remove a widget from a tab group list

Input Methods

XmImCloseXIM(1)	Close all input contexts associated with the input method
XmImFreeXIC(1)	Free an input context
XmImGetXIC(1)	Create an input context for a widget
XmImGetXIM(1)	Retrieve the input method for a widget
XmImMbLookupString(1)	Retrieve a compound string from an input method
XmImMbResetIC(1)	Reset an input context
XmImRegister(1)	Register an input context for a widget
XmImSetFocusValues(1)	Set the values and focus for an input context
XmImSetValues(1)	Set the values for an input context
XmImSetXIC(1)	Register a widget with an existing input context
XmImUnregister(1)	Unregister the input context for a widget
XmImUnsetFocus(1)	Unset focus for an input context
XmImVaSetFocusValues(1)	Set the values and focus for an input context
XmImVaSetValues(1)	Set the values for an input context

List

XmListAddItem(1)	Add an item to a List
XmListAddItems(1)	Add items to a List
XmListAddItemUnselected(1)	Add an item to a List, without selecting the item
XmListAddItemsUnselected(1)	Add items to a List, without selecting the items
XmListDeleteAllItems(1)	Delete all of the items from a List
XmListDeleteItem(1)	Delete an item from a List
XmListDeleteItems(1)	Delete items from a List
XmListDeleteItemsPos(1)	Delete items starting at a specified position from a List
XmListDeletePos(1)	Delete an item at a specified position from a List
XmListDeletePositions(1)	Delete items at specified positions from a List
XmListDeselectAllItems(1)	Deselect all items in a List
XmListDeselectPos(1)	Deselect an item at a specified position in a List

XmListGetKbdItemPos(1)	Get the position of the item that has the location cursor in a List
XmListGetMatchPos(1)	Get all occurrences of an item in a List
XmListItemExists(1)	Determine if a specified item is in a List
XmListItemPos(1)	Determine the position of an item in a List
XmListPosSelected(1)	Check if the item at the specified position in the List is selected
XmListPosToBounds(1)	Return the bounding box of the item at the specified position in a List
XmListReplaceItems(1)	Replace specified items in a List
XmListReplaceItemsPos(1)	Replace items at specified positions in a List
XmListReplaceItemsPosUnselected(1)	Replace items at specified positions in a List, without selecting the replacement items
XmListReplaceItemsUnselected(1)	Replace items in the List, without selecting the replacement items
XmListReplacePositions(1)	Replace items at the specified positions in a List
XmListSelectedPos(1)	Get the positions of the selected items in a List
XmListSelectItem(1)	Select a specified item in a List
XmListSelectPos(1)	Select the item at the specified position in a List
XmListSetAddMode(1)	Set add mode in a List
XmListSetBottomItem(1)	Set the last visible item in a List
XmListSetBottomPos(1)	Set the last visible position in a List
XmListSetHorizPos(1)	Set the horizontal position of a List
XmListSetItem(1)	Set the first visible item in a List
XmListSetKbdItemPos(1)	Set the position of the location cursor in a List
XmListSetPos(1)	Set the first visible position in a List
XmListUpdateSelectedList(1)	Update the set of selected items in a List
XmListYToPos(1)	Get the position of the item at the specified y-coordinate in a List

MainWindow

XmMainWindowSep1(1)	Get the widget ID of the first MainWindow Separator

XmMainWindowSep2(1)	Get the widget ID of the second MainWindow Separator
XmMainWindowSep3(1)	Get the widget ID of the third MainWindow Separator
XmMainWindowSetAreas(1)	Specify the children for a MainWindow

Menus

XmGetPostedFromWidget(1)	Get the widget that posted a menu
XmGetTearOffControl(1)	Get the tear-off control for a Menu
XmMenuPosition(1)	Position a popup menu
XmOptionButtonGadget(1)	Get the CascadeButtonGadget in an OptionMenu
XmOptionLabelGadget(1)	Get the LabelGadget in an OptionMenu
XmRemoveFromPostFromList(1)	Make a menu inaccessible from a widget

MessageBox

XmMessageBoxGetChild(1)	Retrieve a specified child of a MessageBox widget

Mrm Functions

MrmCloseHierarchy(3)	Close an Mrm hierarchy
MrmFetchBitmapLiteral(3)	Retrieve an exported bitmap from an Mrm hierarchy
MrmFetchColorLiteral(3)	Retrieve an exported color value from an Mrm hierarchy
MrmFetchIconLiteral(3)	Retrieve an exported icon from an Mrm hierarchy
MrmFetchLiteral(3)	Retrieve an exported value from an Mrm hierarchy
MrmFetchSetValues(3)	Set widget resources to values retrieved from an Mrm hierarchy
MrmFetchWidget(3)	Create the widget tree rooted at a named widget
MrmFetchWidgetOverride(3)	Create the widget tree rooted at a named widget, and override the resources set in the UID file
MrmInitialize(3)	Prepare the Mrm library for use
MrmOpenHierarchy(3)	Open an Mrm hierarchy
MrmOpenHierarchyFromBuffer(3)	Open an Mrm hierarchy

MrmOpenHierarchyPerDisplay(3)	Open an Mrm hierarchy
MrmRegisterClass(3)	Register a widget creation routine for a non-Motif component
MrmRegisterNames(3)	Register application-defined values and procedures
MrmRegisterNamesInHierarchy(3)	Register application-defined values and procedures for use in a specific UIL hierarchy

Notebook

XmNotebookGetPageInfo(1)	Return information about a Notebook page

Parse Mapping

XmParseMappingCreate(1)	Create a parse mapping
XmParseMappingFree(1)	Free the memory used by a parse mapping
XmParseMappingGetValues(1)	Fetch resources from a parse mapping object
XmParseMappingSetValues(1)	Set resources for a parse mapping object
XmParseTableFree(1)	Free the memory used by a parse table

Pixmaps

XmDestroyPixmap(1)	Remove a pixmap from the pixmap cache
XmGetPixmap(1)	Create and return a pixmap
XmGetPixmapByDepth(1)	Create and return a pixmap of the specified depth
XmGetScaledPixmap(1)	Create and return a scaled pixmap
XmInsiallImage(1)	Install an image in the image cache
XmUninstallImage(1)	Remove an image from the image cache

Print

XmPrintPopupPDM(1)	Notify the Print Display Manager
XmPrintSetup(1)	Create a PrintShell widget
XmPrintToFile(1)	Save X print Server data to file
XmRedisplayWidget(1)	Force widget exposure for printing

Protocols

XmActivateProtocol(1)	Activate a protocol
XmActivateWMProtocol(1)	Activate the XA_WM_PROTOCOLS protocol
XmAddProtocolCallback(1)	Add client callbacks to a protocol
XmAddProtocols(1)	Add protocols to the protocol manager
XmAddWMProtocolCallback(1)	Add client callbacks to an XA_WM_PROTOCOLS protocol
XmAddWMProtocols(1)	Add the XA_WM_PROTOCOLS protocol to the protocol manager
XmDeactivateProtocol(1)	Deactivate a protocol
XmDeactivateWMProtocol(1)	Deactivate the XA_WM_PROTOCOLS protocol
XmRemoveProtocolCallback(1)	Remove client callback from a protocol
XmRemoveProtocols(1)	Remove protocols from the protocol manager
XmRemoveWMProtocolCallback(1)	Remove client callbacks from the XA_WM_PROTOCOLS protocol
XmRemoveWMProtocols(1)	Remove the XA_WM_PROTOCOLS protocol from the protocol manager
XmSetProtocolHooks(1)	Set prehooks and posthooks for a protocol
XmSetWMProtocolHooks(1)	Set prehooks and posthooks for the XA_WM_PROTOCOLS protocol

Render Tables and Renditions

XmRenderTableAddRenditions(1)	Add renditions to a render table
XmRenderTableCopy(1)	Copy a render table
XmRenderTableCvtFromProp(1)	Convert from a string representation into a render table
XmRenderTableCvtToProp(1)	Convert a render table into a string representation
XmRenderTableFree(1)	Free the memory used by a render table
XmRenderTableGetRendition(1)	Search a render table for a matching rendition
XmRenderTableGetRenditions(1)	Search a render table for a matching renditions
XmRenderTableGetTags(1)	Fetch the list of rendition tags from a render table
XmRenderTableRemoveRenditions(1)	Copy a render table, excluding specified renditions

XmRenditionCreate(1)	Create a rendition object
XmRenditionFree(1)	Free the memory used by a rendition object
XmRenditionRetrieve(1)	Fetch rendition object resources
XmRenditionUpdate(1)	Set rendition object resources

Resolution Independence

XmConvertUnits(1)	Convert a value to a specified unit type

Resource Conversion

XmConvertStringToUnits(1)	Convert a string to an integer, optionally translating the units
XmCvtStringToUnitType(1)	Convert a string to a unit-type value
XmRepTypeAddReverse(1)	Install the reverse converter for a representation type
XmRepTypeGetId(1)	Get the ID number of a representation type
XmRepTypeGetNameList(1)	Get the list of value names for a representation type
XmRepTypeGetRecord(1)	Get the information about a representation type
XmRepTypeGetRegistered(1)	Get the registered representation types
XmRepTypeInstallTearOffModelConverter(1)	Install the resource converter for XmNtearOffModel
XmRepTypeRegister(1)	Register a representation type resource
XmRepTypeValidValue(1)	Determine the validity of a numerical value for a representation type

Scale

XmScaleGetValue(1)	Get the slider value for a Scale widget
XmScaleSetTicks(1)	Set tick marks for a Scale widget
XmScaleSetValue(1)	Set the slider value for a Scale widget

Screen

XmGetXmScreen(1)	Get the XmScreen object for a screen

ScrollBar

XmScrollBarGetValues(1)	Get information about the current state of a ScrollBar
XmScrollBarSetValues(1)	Set the current state of a ScrollBar widget

ScrolledWindow

XmScrolledWindowSetAreas(1)	Set the children for a ScrolledWindow
XmScrollVisible(1)	Make an obscured child of a ScrolledWindow visible

SelectionBox

XmSelectionBoxGetChild(1)	Get the specified child of a SelectionBox widget

SpinBox

XmSimpleSpinBoxAddItem(1)	Add an item to a SimpleSpinBox
XmSimpleSpinBoxDeletePos(1)	Delete an item at a specified position from a SimpleSpinBox
XmSimpleSpinBoxSetItem(1)	Set an item in a SimpleSpinBox
XmSpinBoxValidatePosition(1)	Validate the current value of a SpinBox

Tab Lists

XmTabCreate(1)	Create a tab (XmTab) object
XmTabFree(1)	Free the memory used by an XmTab object
XmTabGetValues(1)	Fetch the value of an XmTab object
XmTabListCopy(1)[1]	Copy a tab list object
XmTabListFree(1)	Free the memory used by a tab list
XmTabListGetTab(1)	Retrieve a tab from a tab list
XmTabListInsertTabs(1)	Insert tabs into a tab list
XmTabListRemoveTabs(1)	Copy a tab list, excluding specified tabs
XmTabListReplacePositions(1)	Copy a tab list, replacing tabs at specified positions
XmTabListTabCount(1)	Count the number of tabs in a tab list
XmTabSetValue(1)	Set the value of a tab object

1.Erroneously given as XmTabListTabCopy() in 2nd edition

Text

XmTextClearSelection(1)	Clear the primary selection
XmTextCopy(1)	Copy the primary selection to the clipboard
XmTextCopyLink(1)	Copy links from the primary selection to the clipboard
XmTextCut(1)	Copy the primary selection to the clipboard, and remove the selected text
XmTextDisableRedisplay(1)	Prevent visual update
XmTextEnableRedisplay(1)	Allow visual update
XmTextFindString(1)	Find the beginning position of a text string
XmTextFindStringWcs(1)	Find the beginning position of a wide-character text string
XmTextGetBaseline(1)	Get the position of the baseline
XmTextGetCenterline(1)	Get the height of vertical text
XmTextGetCursorPosition(1)	Get the position of the location cursor
XmTextGetEditable(1)	Get the edit permission state
XmTextGetInsertionPosition(1)	Get the position of the insertion cursor
XmTextGetLastPosition(1)	Get the position of the last character
XmTextGetMaxLength(1)	Get the maximum possible length of a text string
XmTextGetSelection(1)	Get the value of the primary selection
XmTextGetSelectionPosition(1)	Get the position of the primary selection
XmTextGetSelectionWcs(1)	Get the wide-character value of the primary selection
XmTextGetSource(1)	Get the text source
XmTextGetString(1)	Get the text string
XmTextGetStringWcs(1)	Get the wide-character text string
XmTextGetSubstring(1)	Get a copy of part of the text string
XmTextGetSubstringWcs(1)	Get a copy of part of the wide-character text string
XmTextGetTopCharacter(1)	Get the position of the first visible character
XmTextInsert(1)	Insert a string into the text string
XmTextInsertWcs(1)	Insert a wide-character string into the text string
XmTextPaste(1)	Insert the clipboard selection
XmTextPasteLink(1)	Insert links from the clipboard selection
XmTextPosToXY(1)	Get the x, y coordinates of a character position

XmTextRermove(1)	Delete the primary selection
XmTextReplace(1)	Replace part of the text string
XmTextReplaceWcs(1)	Replace part of the wide-character text string
XmTextScroll(1)	Scroll the text
XmTextSetAddMode(1)	Set the add mode state
XmTextSetCursorPosition(1)	Set the position of the location cursor
XmTextSetEditable(1)	Set the edit permission state
XmTextSetHighlight(1)	Highlight part of the text string
XmTextSetInsertionPosition(1)	Set the position of the insertion cursor
XmTextSetMaxLength(1)	Set the maximum possible length of a text string
XmTextSetSelection(1)	Set the value of the primary selection
XmTextSetSource(1)	Set the text source
XmTextSetString(1)	Set the text string
XmTextSetStringWcs(1)	Set the wide-character text string
XmTextSetTopCharacter(1)	Set the position of the first visible character
XmTextShowPosition(1)	Scroll the text so that a specified position is visible
XmTextXYToPos(1)	Get the character position for a given x, y coordinate

TextField

XmTextFieldClearSelection(1)	Clear the primary selection
XmTextFieldCopy(1)	Copy the primary selection to the clipboard
XmTextFieldCopyLink(1)	Copy links from the primary selection to the clipboard
XmTextFieldCut(1)	Copy the primary selection to the clipboard, and remove the selected text
XmTextFieldGetBaseline(1)	Get the position of the baseline
XmTextFieldGetCursorPosition(1)	Get the position of the location cursor
XmTextFieldGetEditable(1)	Get the edit permission state
XmTextFieldGetInsertionPosition(1)	Get the position of the insertion cursor
XmTextFieldGetLastPosition(1)	Get the position of the last character
XmTextFieldGetMaxLength(1)	Get the maximum possible length of a text string
XmTextFieldGetSelection(1)	Get the value of the primary selection
XmTextFieldGetSelectionPosition(1)	Get the position of the primary selection

XmTextFieldGetSelectionWcs(1)	Get the wide-character value of the primary selection
XmTextFieldGetString(1)	Get the text string
XmTextFieldGetStringWcs(1)	Get the wide-character text string
XmTextFieldGetSubstring(1)	Get a copy of part of the text string
XmTextFieldGetSubstringWcs(1)	Get a copy of part of the wide-character text string
XmTextFieldGetTopCharacter(1)	Get the position of the first visible character
XmTextFieldInsert(1)	Insert a string into the text string
XmTextFieldInsertWcs(1)	Insert a wide-character string into the text string
XmTextFieldPaste(1)	Insert the clipboard selection
XmTextFieldPasteLink(1)	Insert links from the clipboard selection
XmTextFieldPosToXY(1)	Get the x, y coordinates of a character position
XmTextFieldRermove(1)	Delete the primary selection
XmTextFieldReplace(1)	Replace part of the text string
XmTextFieldReplaceWcs(1)	Replace part of the wide-character text string
XmTextFieldSetAddMode(1)	Set the add mode state
XmTextFieldSetCursorPosition(1)	Set the position of the location cursor
XmTextFieldSetEditable(1)	Set the edit permission state
XmTextFieldSetHighlight(1)	Highlight part of the text string
XmTextFieldSetInsertionPosition(1)	Set the position of the insertion cursor
XmTextFieldSetMaxLength(1)	Set the maximum possible length of a text string
XmTextFieldSetSelection(1)	Set the value of the primary selection
XmTextFieldSetSource(1)	Set the text source
XmTextFieldSetString(1)	Set the text string
XmTextFieldSetStringWcs(1)	Set the wide-character text string
XmTextFieldShowPosition(1)	Scroll the text so that a specified position is visible
XmTextFieldXYToPos(1)	Get the character position for a given x, y coordinate

ToggleButton

XmToggleButtonGadgetGetState(1)	Get the state of a ToggleButtonGadget
XmToggleButtonGadgetSetState(1)	Set the state of a ToggleButtonGadget
XmToggleButtonGadgetSetValue(1)	Set the value of a ToggleButtonGadget
XmToggleButtonGetState(1)	Get the state of a ToggleButton

XmToggleButtonSetState(1)	Set the state of a ToggleButton
XmToggleButtonSetValue(1)	Set the value of a ToggleButton

Uniform Transfer Model

XmTransferDone(1)	Complete a data transfer operation
XmTransferSendRequest(1)	Send a multiple transfer request
XmTransferSetParameters(1)	Set parameters for the next transfer operation
XmTransferStartRequest(1)	Initiate a multiple transfer batch operation
XmTransferValue(1)	Transfer data to a destination

Widget Class

XmIs*Object*(1)	Determine whether a widget is a subclass of a given class

Widget Creation

XmCreate*Object*(1)	Create an instance of a particular widget class or compound object
XmPrintSetup(1)	Create a PrintShell widget
XmVaCreateSimpleCheckBox(1)	Create a CheckBox compound object
XmVaCreateSimpleMenuBar(1)	Create a MenuBar compound object
XmVaCreateSimpleOptionMenu(1)	Create an OptionMenu compound object
XmVaCreateSimplePopupMenu(1)	Create a PopupMenu compound object as the child of a MenuShell
XmVaCreateSimplePulldownMenu(1)	Create a PulldownMenu compound object as the child of a MenuShell
XmVaCreateSimpleRadioBox(1)	Create a RadioBox compound object

Widget Internals

XmGetSecondaryResourceData(1)	Retrieve secondary widget resource data
XmResolveAllPartOffset(1)	Ensure upward-compatible widgets and applications
XmResolvePartOffsets(1)	Ensure upward-compatible widgets and applications

Widget Layout

XmObjectAtPoint(1)	Determine the child nearest to a point coordinate
XmWidgetGetBaselines(1)	Get the positions of the baselines in a widget
XmWidgetGetDisplayRect(1)	Get the display rectangle for a widget

Widget Selection

XmTrackingEvent(1)	Allow for modal selection of a component
XmTrackingLocate(1)	Allow for modal selection of a component

Window Manager

XmIsMotifWMRunning(1)	Check whether the Motif Window Manager (mwm) is running

A.2 Alphabetical Listing

MrmCloseHierarchy(3)	Close an Mrm hierarchy
MrmFetchBitmapLiteral(3)	Retrieve an exported bitmap from an Mrm hierarchy
MrmFetchColorLiteral(3)	Retrieve an exported color value from an Mrm hierarchy
MrmFetchIconLiteral(3)	Retrieve an exported icon from an Mrm hierarchy
MrmFetchLiteral(3)	Retrieve an exported value from an Mrm hierarchy
MrmFetchSetValues(3)	Set widget resources to values retrieved from an Mrm hierarchy
MrmFetchWidget(3)	Create the widget tree rooted at a named widget
MrmFetchWidgetOverride(3)	Create the widget tree rooted at a named widget, and override the resources set in the UID file
MrmInitialize(3)	Prepare the Mrm library for use
MrmOpenHierarchy(3)	Open an Mrm hierarchy
MrmOpenHierarchyFromBuffer(3)	Open an Mrm hierarchy
MrmOpenHierarchyPerDisplay(3)	Open an Mrm hierarchy

MrmRegisterClass(3)	Register a widget creation routine for a non-Motif component
MrmRegisterNames(3)	Register application-defined values and procedures
MrmRegisterNamesInHierarchy(3)	Register application-defined values and procedures for use in a specific UIL hierarchy
XmActivateProtocol(1)	Activate a protocol
XmActivateWMProtocol(1)	Activate the XA_WM_PROTOCOLS protocol
XmAddProtocolCallback(1)	Add client callbacks to a protocol
XmAddProtocols(1)	Add protocols to the protocol manager
XmAddTabGroup(1)	Add a widget to a tab group list
XmAddWMProtocolCallback(1)	Add client callbacks to an XA_WM_PROTOCOLS protocol
XmAddWMProtocols(1)	Add the XA_WM_PROTOCOLS protocol to the protocol manager
XmCascadeButtonGadgetHighlight(1)	Set the highlight state of a CascadeButtonGadget
XmCascadeButtonHighlight(1)	Set the highlight state of a CascadeButton
XmChangeColor(1)	Update the colors of a widget
XmClipboardBeginCopy(1)	Set up storage for a clipboard copy operation
XmClipboardCancelCopy(1)	Cancel a copy operation to the clipboard
XmClipboardCopy(1)	Copy a data item to temporary storage for later copying to the clipboard
XmClipboardCopyByName(1)	Copy as data item passed by name
XmClipboardEndCopy(1)	End a copy operation to the clipboard
XmClipboardEndRetrieve(1)	End a copy operation from the clipboard
XmClipboardInquireCount(1)	Get the number of data item formats available on the clipboard
XmClipboardInquireFormat(1)	Get the specified clipboard data format name
XmClipboardInquireLength(1)	Get the length of the data item on the clipboard
XmClipboardInquirePendingItems(1)	Get a list of pending data ID/private ID pairs
XmClipboardLock(1)	Lock the clipboard
XmClipboardRetrieve(1)	Retrieve a data item from the clipboard
XmClipboardStartCopy(1)	Set up storage for a clipboard copy operation

XmClipboardStartRetrieve(1)	Start a clipboard retrieval operation
XmClipboardUndoCopy(1)	Remove the last item copied to the clipboard
XmClipboardUnlock(1)	Unlock the clipboard
XmClipboardWithdrawFormat(1)	Indicate that an application does not want to supply a data item any longer
XmClopboardRegisterFormat(1)	Register a new format for clipboard data items
XmComboBoxAddItem(1)	Add a compound string to the ComboBox List
XmComboBoxDeletePos(1)	Delete an item at the specified position from a ComboBox List
XmComboBoxSelectItem(1)	Select an item from a ComboBox List
XmComboBoxSetItem(1)	Select and make visible an item from a ComboBox List
XmComboBoxUpdate(1)	Update the ComboBox List after changes to component widgets
XmCommandAppendValue(1)	Append a compound string to a Command widget
XmCommandError(1)	Display an error message in a Command widget
XmCommandGetChild(1)	Get the specified child of a Command widget
XmCommandSetValue(1)	Replace the Command string
XmContainerCopy(1)	Copy the Container primary selection to the clipboard
XmContainerCopyLink(1)	Copy links to the Container primary selection into the clipboard
XmContainerCut(1)	Cuts the Container primary selection onto the clipboard
XmContainerGetItemChild(1)	Find the children of a Container item
XmContainerPaste(1)	Pastes the clipboard selection into a Container
XmContainerPasteLink(1)	Copies links from the clipboard selection into a Container
XmContainerRelayout(1)	Force relayout of a Container widget
XmContainerReorder(1)	Reorder children of a Container
XmConvertStringToUnits(1)	Convert a string to an integer, optionally translating the units
XmConvertUnits(1)	Convert a value to a specified unit type
XmCreateObject(1)	Create an instance of a particular widget class or compound object

XmCvtByteStreamToXmString()	Convert a byte stream to a compound string
XmCvtCTToXmString(1)	Convert compound text to a compound string
XmCvtStringToUnitType(1)	Convert a string to a unit-type value
XmCvtTextPropertyToXmStringTable(1)	Convert an XTextProperty to a compound string table
XmCvtXmStringTableToTextProperty(1)	Convert a compound string table to an XTextProperty
XmCvtXmStringToByteStream(1)	Convert a compound string to byte stream format
XmCvtXmStringToCT(1)	Convert a compound string to compound text
XmDeactivateProtocol(1)	Deactivate a protocol
XmDeactivateWMProtocol(1)	Deactivate the XA_WM_PROTOCOLS protocol
XmDestroyPixmap(1)	Remove a pixmap from the pixmap cache
XmDirectionMatch(1)	Compare two directions
XmDirectionMatchPartial(1)	Partially compare two directions
XmDirectionToStringDirection(1)	Convert a direction to a string direction
XmDragCancel(1)	Cancel a drag operation
XmDragStart(1)	Start a drag operation
XmDropSideEndUpdate(1)	End an update of multiple drop site operations
XmDropSiteConfigureStackingOrder(1)	Change the stacking order of a drop site
XmDropSiteQueryStackingOrder(1)	Get the stacking order of a drop site
XmDropSiteRegister(1)	Register a widget as a drop site
XmDropSiteRetrieve(1)	Get the resource values of a drop site
XmDropSiteStartUpdate(1)	Start an update of multiple drop site operations
XmDropSiteUnregister(1)	Remove a drop site
XmDropSiteUpdate(1)	Change the resource values for a drop site
XmDropTransferAdd(1)	Add drop transfer entries to a drop operation
XmDropTransferStart(1)	Start a drop operation
XmFileSelectionBoxGetChild(1)	Get a specified child of a FileSelectionBox widget
XmFileSelectionDoSearch(1)	Start a directory search
XmFontListAdd(1)	Create a new font list
XmFontListAppendEntry(1)	Append a font entry to a font list

XmFontListCopy(1)	Copy a font list
XmFontListCreate(1)	Create a new font list
XmFontListCreate_r(1)	Create a new font list in a thread-safe manner
XmFontListEntryCreate(1)	Create a new font list entry
XmFontListEntryCreate_r(1)	Create a new font list entry in a thread-safe manner
XmFontListEntryFree(1)	Free the memory used by a font list entry
XmFontListEntryGetFont(1)	Get the font information from a font list entry
XmFontListEntryGetTag(1)	Get the tag of a font list entry
XmFontListEntryLoad(1)	Load a font or create a font set, and then create a font list entry
XmFontListFree(1)	Free the memory used by a font list
XmFontListFreeFontContext(1)	Free a font context
XmFontListGetNextFont(1)	Retrieve information about the next font list element
XmFontListInitFontContext(1)	Create a font context
XmFontListNextEntry(1)	Retrieve the next font list entry in a font list
XmFontListRemoveEntry(1)	Remove a font list entry from a font list
XmGetAtomName(1)	Get the string representation of an Atom
XmGetColorCalculation(1)	Get the procedure that calculates default widget colors
XmGetColors(1)	Get the foreground, select, and shadow colors for a widget
XmGetDestination(1)	Get the current destination widget
XmGetDragContext(1)	Get information about a drag-and-drop operation
XmGetFocusWidget(1)	Get the widget that has the keyboard focus
XmGetMenuCursor(1)	Get the current menu cursor
XmGetPixmap(1)	Create and return a pixmap
XmGetPixmapByDepth(1)	Create and return a pixmap of the specified depth
XmGetPostedFromWidget(1)	Get the widget that posted a menu
XmGetScaledPixmap(1)	Create and return a scaled pixmap
XmGetSecondaryResourceData(1)	Retrieve secondary widget resource data
XmGetTabGroup(1)	Get the tab group for a widget
XmGetTearOffControl(1)	Get the tear-off control for a Menu

XmGetVisibility(1)	Determine whether or not a widget is visible
XmGetXmDisplay(1)	Get the XmDisplay object for the Display
XmGetXmScreen(1)	Get the XmScreen object for a screen
XmImCloseXIM(1)	Close all input contexts associated with the input method
XmImFreeXIC(1)	Free an input context
XmImGetXIC(1)	Create an input context for a widget
XmImGetXIM(1)	Retrieve the input method for a widget
XmImMbLookupString(1)	Retrieve a compound string from an input method
XmImMbResetIC(1)	Reset an input context
XmImRegister(1)	Register an input context for a widget
XmImSetFocusValues(1)	Set the values and focus for an input context
XmImSetValues(1)	Set the values for an input context
XmImSetXIC(1)	Register a widget with an existing input context
XmImUnregister(1)	Unregister the input context for a widget
XmImUnsetFocus(1)	Unset focus for an input context
XmImVaSetFocusValues(1)	Set the values and focus for an input context
XmImVaSetValues(1)	Set the values for an input context
XmInsiallImage(1)	Install an image in the image cache
XmInternAtom(1)	Return an Atom for a given property name string
XmIsMotifWMRunning(1)	Check whether the Motif Window Manager (mwm) is run- ning
XmIsObject(1)	Determine whether a widget is a subclass of a given class
XmIsTraversable(1)	Determine whether or not a widget can receive the keyboard focus
XmListAddItem(1)	Add an item to a List
XmListAddItemUnselected(1)	Add an item to a List, without selecting the item
XmListAddItems(1)	Add items to a List
XmListAddItemsUnselected(1)	Add items to a List, without selecting the items
XmListDeleteAllItems(1)	Delete all of the items from a List
XmListDeleteItem(1)	Delete an item from a List
XmListDeleteItems(1)	Delete items from a List

XmListDeleteItemsPos(1)	Delete items starting at a specified position from a List
XmListDeletePos(1)	Delete an item at a specified position from a List
XmListDeletePositions(1)	Delete items at specified positions from a List
XmListDeselectAllItems(1)	Deselect all items in a List
XmListDeselectPos(1)	Deselect an item at a specified position in a List
XmListGetKbdItemPos(1)	Get the position of the item that has the location cursor in a List
XmListGetMatchPos(1)	Get all occurrences of an item in a List
XmListItemExists(1)	Determine if a specified item is in a List
XmListItemPos(1)	Determine the position of an item in a List
XmListPosSelected(1)	Check if the item at the specified position in the List is selected
XmListPosToBounds(1)	Return the bounding box of the item at the specified position in a List
XmListReplaceItems(1)	Replace specified items in a List
XmListReplaceItemsPos(1)	Replace items at specified positions in a List
XmListReplaceItemsPosUnselected(1)	Replace items at specified positions in a List, without selecting the replacement items
XmListReplaceItemsUnselected(1)	Replace items in the List, without selecting the replacement items
XmListReplacePositions(1)	Replace items at the specified positions in a List
XmListSelectItem(1)	Select a specified item in a List
XmListSelectPos(1)	Select the item at the specified position in a List
XmListSelectedPos(1)	Get the positions of the selected items in a List
XmListSetAddMode(1)	Set add mode in a List
XmListSetBottomItem(1)	Set the last visible item in a List
XmListSetBottomPos(1)	Set the last visible position in a List
XmListSetHorizPos(1)	Set the horizontal position of a List
XmListSetItem(1)	Set the first visible item in a List
XmListSetKbdItemPos(1)	Set the position of the location cursor in a List
XmListSetPos(1)	Set the first visible position in a List

XmListUpdateSelectedList(1)	Update the set of selected items in a List
XmListYToPos(1)	Get the position of the item at the specified y-coordinate in a List
XmMainWindowSep1(1)	Get the widget ID of the first MainWindow Separator
XmMainWindowSep2(1)	Get the widget ID of the second MainWindow Separator
XmMainWindowSep3(1)	Get the widget ID of the third MainWindow Separator
XmMainWindowSetAreas(1)	Specify the children for a MainWindow
XmMapSegmentEncoding(1)	Get the compound text encoding format for a font list element tag
XmMenuPosition(1)	Position a popup menu
XmMessageBoxGetChild(1)	Retrieve a specified child of a MessageBox widget
XmNotebookGetPageInfo(1)	Return information about a Notebook page
XmObjectAtPoint(1)	Determine the child nearest to a point coordinate
XmOptionButtonGadget(1)	Get the CascadeButtonGadget in an OptionMenu
XmOptionLabelGadget(1)	Get the LabelGadget in an OptionMenu
XmParseMappingCreate(1)	Create a parse mapping
XmParseMappingFree(1)	Free the memory used by a parse mapping
XmParseMappingGetValues(1)	Fetch resources from a parse mapping object
XmParseMappingSetValues(1)	Set resources for a parse mapping object
XmParseTableFree(1)	Free the memory used by a parse table
XmPrintPopupPDM(1)	Notify the Print Display Manager
XmPrintSetup(1)	Create a PrintShell widget
XmPrintSetup(1)	Create a PrintShell widget
XmPrintToFile(1)	Save X print Server data to file
XmProcessTraversal(1)	Set the widget which has the keyboard focus
XmRedisplayWidget(1)	Force widget exposure for printing
XmRegisterSegmentEncoding(1)	Register a compound text encoding format for a font list element tag
XmRemoveFromPostFromList(1)	Make a menu inaccessible from a widget
XmRemoveProtocolCallback(1)	Remove client callback from a protocol

XmRemoveProtocols(1)	Remove protocols from the protocol manager
XmRemoveTabGroup(1)	Remove a widget from a tab group list
XmRemoveWMProtocolCallback(1)	Remove client callbacks from the XA_WM_PROTOCOLS protocol
XmRemoveWMProtocols(1)	Remove the XA_WM_PROTOCOLS protocol from the protocol manager
XmRenderTableAddRenditions(1)	Add renditions to a render table
XmRenderTableCopy(1)	Copy a render table
XmRenderTableCvtFromProp(1)	Convert from a string representation into a render table
XmRenderTableCvtToProp(1)	Convert a render table into a string representation
XmRenderTableFree(1)	Free the memory used by a render table
XmRenderTableGetRendition(1)	Search a render table for a matching rendition
XmRenderTableGetRenditions(1)	Search a render table for a matching renditions
XmRenderTableGetTags(1)	Fetch the list of rendition tags from a render table
XmRenderTableRemoveRenditions(1)	Copy a render table, excluding specified renditions
XmRenditionCreate(1)	Create a rendition object
XmRenditionFree(1)	Free the memory used by a rendition object
XmRenditionRetrieve(1)	Fetch rendition object resources
XmRenditionUpdate(1)	Set rendition object resources
XmRepTypeAddReverse(1)	Install the reverse converter for a representation type
XmRepTypeGetId(1)	Get the ID number of a representation type
XmRepTypeGetNameList(1)	Get the list of value names for a representation type
XmRepTypeGetRecord(1)	Get the information about a representation type
XmRepTypeGetRegistered(1)	Get the registered representation types
XmRepTypeInstallTearOffModelConverter(1)	Install the resource converter for XmNtearOffModel
XmRepTypeRegister(1)	Register a representation type resource
XmRepTypeValidValue(1)	Determine the validity of a numerical value for a representation type

XmResolveAllPartOffset(1)	Ensure upward-compatible widgets and applications
XmResolvePartOffsets(1)	Ensure upward-compatible widgets and applications
XmScaleGetValue(1)	Get the slider value for a Scale widget
XmScaleSetTicks(1)	Set tick marks for a Scale widget
XmScaleSetValue(1)	Set the slider value for a Scale widget
XmScrollBarGetValues(1)	Get information about the current state of a ScrollBar
XmScrollBarSetValues(1)	Set the current state of a ScrollBar widget
XmScrollVisible(1)	Make an obscured child of a ScrolledWindow visible
XmScrolledWindowSetAreas(1)	Set the children for a ScrolledWindow
XmSelectionBoxGetChild(1)	Get the specified child of a SelectionBox widget
XmSetColorCalculation(1)	Set the procedure that calculates default widget colors
XmSetFontUnit(1)	Set the font unit values
XmSetFontUnits(1)	Set the font unit values
XmSetMenuCursor(1)	Set the current menu cursor
XmSetProtocolHooks(1)	Set prehooks and posthooks for a protocol
XmSetWMProtocolHooks(1)	Set prehooks and posthooks for the XA_WM_PROTOCOLS protocol
XmSimpleSpinBoxAddItem(1)	Add an item to a SimpleSpinBox
XmSimpleSpinBoxDeletePos(1)	Delete an item at a specified position from a SimpleSpinBox
XmSimpleSpinBoxSetItem(1)	Set an item in a SimpleSpinBox
XmSpinBoxValidatePosition(1)	Validate the current value of a SpinBox
XmStringBaseline(1)	Get the baseline spacing for a compound string
XmStringByteCompare(1)	Compare two compound strings byte-by-byte
XmStringByteStreamLength(1)	Calculate the length of a byte stream
XmStringCompare(1)	Compare two compound strings
XmStringComponentCreate(1)	Create a compound string component
XmStringConcat(1)	Concatenate two compound strings
XmStringConcatAndFree(1)	Create a new compound string formed by concatenating two compound strings, freeing the two strings afterwards
XmStringCopy(1)	Copy a compound string

XmStringCreate(1)	Create a compound string
XmStringCreateLocalized(1)	Create a compound string in the current locale
XmStringCreateLtoR(1)	Create a compound string
XmStringCreateSimple(1)	Create a compound string in the current language environment
XmStringDirectionCreate(1)	Create a compound string containing a direction component
XmStringDirectionToDirection(1)	Convert a string direction to a direction
XmStringDirectionToDirection(1)	Convert a string direction to a direction
XmStringDraw(1)	Draw a compound string
XmStringDrawImage(1)	Draw a compound string
XmStringDrawUnderline(1)	Draw a compound string with an underlined sub-string
XmStringEmpty(1)	Determine whether there are any text components in a compound string
XmStringExtent(1)	Get the smallest rectangle enclosing a compound string
XmStringFree(1)	Free the memory used by a compound string
XmStringFreeContext(1)	Free a compound string context
XmStringGenerate(1)	Generate a compound string
XmStringGetLtoR(1)	Get a text segment from a compound string
XmStringGetNextComponent(1)	Retrieves information about the next compound string component
XmStringGetNextSegment(1)	Retrieves information about the next compound string component
XmStringGetNextTriple(1)	Retrieve information about the next compound string segment
XmStringHasSubstring(1)	Determine whether a compound string contains a sub-string
XmStringHeight(1)	Get the line height of a compound string
XmStringInitContext(1)	Create a compound string context
XmStringIsVoid(1)	Determine whether there are valid segments in a compound string
XmStringLength(1)	Get the length of a compound string
XmStringLineCount(1)	Get the number of lines in a compound string
XmStringNConcat(1)	Concatenate a specified portion of a compound string to another compound string

XmStringNCopy(1)	Copy a specified portion of a compound string
XmStringParseText(1)	Convert a string to a compound string
XmStringPeekNextComponent(1)	Returns the type of the next compound string component
XmStringPutRendition(1)	Add rendition components to a compound string
XmStringSegmentCreate(1)	Create a compound string segment
XmStringSeparatorCreate(1)	Create a compound string containing a separator component
XmStringTableParseStringArray(1)	Convert an array of strings into a compound string table
XmStringTableProposeTabList(1)	Create a Tab list for a compound string
XmStringTableToXmString(1)	Convert a compound string table to a compound string
XmStringTableUnparse(1)	Convert a compound string table to a string
XmStringToXmStringTable(1)	Convert a compound string to a compound string table
XmStringUnparse(1)	Convert a compound string to a string
XmStringWidth(1)	Return the width of a compound string
XmTabCreate(1)	Create a tab (XmTab) object
XmTabFree(1)	Free the memory used by an XmTab object
XmTabGetValues(1)	Fetch the value of an XmTab object
XmTabListCopy(1)	Copy a tab list object
XmTabListFree(1)	Free the memory used by a tab list
XmTabListGetTab(1)	Retrieve a tab from a tab list
XmTabListInsertTabs(1)	Insert tabs into a tab list
XmTabListRemoveTabs(1)	Copy a tab list, excluding specified tabs
XmTabListReplacePositions(1)	Copy a tab list, replacing tabs at specified positions
XmTabListTabCount(1)	Count the number of tabs in a tab list
XmTabSetValue(1)	Set the value of a tab object
XmTargetsAreCompatible(1)	Determine whether or not the target types of a drag source and a drop site match
XmTextClearSelection(1)	Clear the primary selection
XmTextCopy(1)	Copy the primary selection to the clipboard
XmTextCopyLink(1)	Copy links from the primary selection to the clipboard

XmTextCut(1)	Copy the primary selection to the clipboard, and remove the selected text
XmTextDisableRedisplay(1)	Prevent visual update
XmTextEnableRedisplay(1)	Allow visual update
XmTextFieldClearSelection(1)	Clear the primary selection
XmTextFieldCopy(1)	Copy the primary selection to the clipboard
XmTextFieldCopyLink(1)	Copy links from the primary selection to the clipboard
XmTextFieldCut(1)	Copy the primary selection to the clipboard, and remove the selected text
XmTextFieldGetBaseline(1)	Get the position of the baseline
XmTextFieldGetCursorPosition(1)	Get the position of the location cursor
XmTextFieldGetEditable(1)	Get the edit permission state
XmTextFieldGetInsertionPosition(1)	Get the position of the insertion cursor
XmTextFieldGetLastPosition(1)	Get the position of the last character
XmTextFieldGetMaxLength(1)	Get the maximum possible length of a text string
XmTextFieldGetSelection(1)	Get the value of the primary selection
XmTextFieldGetSelectionPosition(1)	Get the position of the primary selection
XmTextFieldGetSelectionWcs(1)	Get the wide-character value of the primary selection
XmTextFieldGetString(1)	Get the text string
XmTextFieldGetStringWcs(1)	Get the wide-character text string
XmTextFieldGetSubstring(1)	Get a copy of part of the text string
XmTextFieldGetSubstringWcs(1)	Get a copy of part of the wide-character text string
XmTextFieldGetTopCharacter(1)	Get the position of the first visible character
XmTextFieldInsert(1)	Insert a string into the text string
XmTextFieldInsertWcs(1)	Insert a wide-character string into the text string
XmTextFieldPaste(1)	Insert the clipboard selection
XmTextFieldPasteLink(1)	Insert links from the clipboard selection
XmTextFieldPosToXY(1)	Get the x, y coordinates of a character position
XmTextFieldReplace(1)	Replace part of the text string
XmTextFieldReplaceWcs(1)	Replace part of the wide-character text string
XmTextFieldRemove(1)	Delete the primary selection
XmTextFieldSetAddMode(1)	Set the add mode state
XmTextFieldSetCursorPosition(1)	Set the position of the location cursor

XmTextFieldSetEditable(1)	Set the edit permission state
XmTextFieldSetHighlight(1)	Highlight part of the text string
XmTextFieldSetInsertionPosition(1)	Set the position of the insertion cursor
XmTextFieldSetMaxLength(1)	Set the maximum possible length of a text string
XmTextFieldSetSelection(1)	Set the value of the primary selection
XmTextFieldSetSource(1)	Set the text source
XmTextFieldSetString(1)	Set the text string
XmTextFieldSetStringWcs(1)	Set the wide-character text string
XmTextFieldShowPosition(1)	Scroll the text so that a specified position is visible
XmTextFieldXYToPos(1)	Get the character position for a given x, y coordinate
XmTextFindString(1)	Find the beginning position of a text string
XmTextFindStringWcs(1)	Find the beginning position of a wide-character text string
XmTextGetBaseline(1)	Get the position of the baseline
XmTextGetCenterline(1)	Get the height of vertical text
XmTextGetCursorPosition(1)	Get the position of the location cursor
XmTextGetEditable(1)	Get the edit permission state
XmTextGetInsertionPosition(1)	Get the position of the insertion cursor
XmTextGetLastPosition(1)	Get the position of the last character
XmTextGetMaxLength(1)	Get the maximum possible length of a text string
XmTextGetSelection(1)	Get the value of the primary selection
XmTextGetSelectionPosition(1)	Get the position of the primary selection
XmTextGetSelectionWcs(1)	Get the wide-character value of the primary selection
XmTextGetSource(1)	Get the text source
XmTextGetString(1)	Get the text string
XmTextGetStringWcs(1)	Get the wide-character text string
XmTextGetSubstring(1)	Get a copy of part of the text string
XmTextGetSubstringWcs(1)	Get a copy of part of the wide-character text string
XmTextGetTopCharacter(1)	Get the position of the first visible character
XmTextInsert(1)	Insert a string into the text string
XmTextInsertWcs(1)	Insert a wide-character string into the text string
XmTextPaste(1)	Insert the clipboard selection
XmTextPasteLink(1)	Insert links from the clipboard selection

XmTextPosToXY(1)	Get the x, y coordinates of a character position
XmTextReplace(1)	Replace part of the text string
XmTextReplaceWcs(1)	Replace part of the wide-character text string
XmTextRermove(1)	Delete the primary selection
XmTextScroll(1)	Scroll the text
XmTextSetAddMode(1)	Set the add mode state
XmTextSetCursorPosition(1)	Set the position of the location cursor
XmTextSetEditable(1)	Set the edit permission state
XmTextSetHighlight(1)	Highlight part of the text string
XmTextSetInsertionPosition(1)	Set the position of the insertion cursor
XmTextSetMaxLength(1)	Set the maximum possible length of a text string
XmTextSetSelection(1)	Set the value of the primary selection
XmTextSetSource(1)	Set the text source
XmTextSetString(1)	Set the text string
XmTextSetStringWcs(1)	Set the wide-character text string
XmTextSetTopCharacter(1)	Set the position of the first visible character
XmTextShowPosition(1)	Scroll the text so that a specified position is visible
XmTextXYToPos(1)	Get the character position for a given x, y coordinate
XmToggleButtonGadgetGetState(1)	Get the state of a ToggleButtonGadget
XmToggleButtonGadgetSetState(1)	Set the state of a ToggleButtonGadget
XmToggleButtonGadgetSetValue(1)	Set the value of a ToggleButtonGadget
XmToggleButtonGetState(1)	Get the state of a ToggleButton
XmToggleButtonSetState(1)	Set the state of a ToggleButton
XmToggleButtonSetValue(1)	Set the value of a ToggleButton
XmTrackingEvent(1)	Allow for modal selection of a component
XmTrackingLocate(1)	Allow for modal selection of a component
XmTransferDone(1)	Complete a data transfer operation
XmTransferSendRequest(1)	Send a multiple transfer request
XmTransferSetParameters(1)	Set parameters for the next transfer operation
XmTransferStartRequest(1)	Initiate a multiple transfer batch operation
XmTransferValue(1)	Transfer data to a destination

XmTranslateKey(1)	Convert a KeyCode to a KeySym using the default translator
XmUninstallImage(1)	Remove an image from the image cache
XmUpdateDisplay(1)	Update the display
XmVaCreateSimpleCheckBox(1)	Create a CheckBox compound object
XmVaCreateSimpleMenuBar(1)	Create a MenuBar compound object
XmVaCreateSimpleOptionMenu(1)	Create an OptionMenu compound object
XmVaCreateSimplePopupMenu(1)	Create a PopupMenu compound object as the child of a MenuShell
XmVaCreateSimplePulldownMenu(1)	Create a PulldownMenu compound object as the child of a MenuShell
XmVaCreateSimpleRadioBox(1)	Create a RadioBox compound object
XmWidgetGetBaselines(1)	Get the positions of the baselines in a widget
XmWidgetGetDisplayRect(1)	Get the display rectangle for a widget

Appendix B - Data Types

This appendix summarizes the data types used as arguments or return values in Motif toolkit and Motif Resource Manager functions. Xt and Xlib data types used by the routines are included. For each data type, the description states the header file that defines the type. Data types (which include simple typedefs as well as structures and enums) are listed alphabetically. Defined symbols (for example, constants used to specify the value of a mask or a field in a structure) or other data types used only to set structure members are listed with the data type in which they are used.

ArgList

An ArgList is used for setting resources in calls to widget creation routines. It is defined as follows in <*X11/Intrinsic.h*>:

```
typedef struct {
    String      name;
    XtArgVal    value;
} Arg, *ArgList;
```

The name *field is* typically a defined constant of the form *XtNresourcename* from either <*X11/Stringdefs.h*> or a widget public header file. It identifies the name of the argument to be set. The value *field is* an *XtArgVal*, a system-dependent typedef chosen to be large enough to hold a pointer to a function. It is often not large enough to hold a float or double.

Atom

To optimize communication with the server, a property is referenced by string name only once, and subsequently by a unique integer ID called an Atom. Predefined atoms are defined in <*X11/Xatom.h*> using defined symbols beginning with XA_; other atoms can be obtained from the server by calling the Xlib function `XInternAtom`(). The Motif toolkit supports an atom-caching mechanism with `XmInternAtom`(). Atoms are used by the Motif protocol routines.

Boolean

A typedef from <*X11/Intrinsic.h*> used to indicate True (1) or False (0). Use either the symbols TRUE or FALSE, defined in <*X11/Intrinsic.h*> or True or False, defined in <*X11/Xlib.h*>.

Cardinal

A typedef from <*X11/Intrinsic.h*> used to specify any unsigned integer value.

Colormap

An XID (server resource ID) from <*X11/X.h*> that identifies a Colormap resource maintained by the server. `XmGetColors`() *and* `MrmFetchColorLiteral`*()* use Colormap values.

Cursor
: A typedef in <*X11/X.h*> for an XID *(*server resource ID) that identifies a cursor resource maintained by the server. A Cursor is used to set the menu cursor in Motif. `XmTrackingEvent`() *and* `XmTrackingLocate`*()* also have a Cursor parameter.

Dimension
: A typedef from <*X11/Intrinsic.h*> used to specify an unsigned short quantity, typically used for window sizes.

Display
: A structure defined in <*X11/Xlib.h*> that contains information about the display the program is running on. Display structure fields should not be accessed directly; Xlib provides a number of macros to return essential values. In Xt, a pointer to the current Display is returned by a call to `XtDisplay`(). The Motif clipboard routines and string drawing routines, among others, use Display parameters. This data type should not be confused with the XmDisplay widget in Motif 1.2 and later.

GC
: A graphics context, which is defined in <*X11/Xlib.h*>. A GC is a pointer to a structure that contains a copy of the settings in a server resource. The server resource, in turn, contains information about how to interpret a graphics primitive. A pointer to a structure of this type is returned by the Xlib call `XCreateGC`() or the Xt call `XtGetGC`(). Motif string drawing routines use GC parameters. The members of this structure should not be accessed directly.

KeyCode
: *A* server-dependent code that describes a key that has been pressed. A *KeyCode* is defined as an unsigned character in <*X11/X.h*>. `XmTranslateKey`() takes a *KeyCode* argument.

KeySym
: A portable representation of the symbol on the cap of a key. The Motif toolkit use both virtual keysyms (osfkeysyms) and actual keysyms. The toolkit maps osfkeysyms to actual keysyms. Individual *KeySyms* are symbols defined in <*X11/keysymdef.h*>. The keycode-to-keysym lookup tables are maintained by the server, and hence a *KeySym* is actually an *XID*. `XmVaCreateSimpleOptionMenu`() and `XmTranslateKey`*()* take *KeySym* arguments.

Modifiers
: Any bitmask that describes modifier keys. The Modifiers type and its values are defined as follows in <*X11/Intrinsic.h*> and <*X11/X.h*>:

  typedef unsigned int Modifiers;

```
#define ShiftMask       (1<<0)
#define LockMask        (1<<1)
#define ControlMask     (1<<2)
#define Mod1Mask        (1<<3)
#define Mod2Mask        (1<<4)
#define Mod3Mask        (1<<5)
#define Mod4Mask        (1<<6)
#define Mod5Mask        (1<<7)
```

`XmTranslateKey`() takes an argument of type *Modifiers*.

MrmCode
: Indicates the type of a value returned by `MrmFetchLiteral`*()*. Codes are prefixed with MrmRtype and are defined in <*Mrm/MrmPublic.h*>.

MrmCount
: A typedef in <*Mrm/MrmPublic.h*> for specifying a count of items.

MrmHierarchy
: A pointer to an Mrm hierarchy opened with `MrmOpenHierarchy`*()* or `MrmOpenHierarchyPerDisplay`(). The type is defined in <*Mrm/MrmPublic.h*>. The functions associate one or more UID files with the hierarchy. An MrmHierarchy is a required argument of most of the Mrm functions.

MrmOsOpenParamPtr
: A structure of operating system-dependent settings used as an argument to `MrmOpenHierarchy`*()* and `MrmOpenHierarchyPerDisplay`() and defined in <*Mrm/MrmPublic.h*>. As of Motif 1.2, the settings are only useful to the UIL compiler.

MrmRegisterArg
: See MrmRegisterArgList.

MrmRegisterArgList
: A type used for registering application-defined procedures and identifiers with `MrmRegisterNames`*()* and `MrmRegisterNamesInHierarchy`(). It is defined as follows in <*Mrm/MrmPublic.h*>*:*

```
typedef struct {
    String          name;   /* case-sensitive name                      */
    XtPointer       value; /* value/procedure to associate with name    */
} MrmRegisterArg, *MrmRegisterArglist;
```

MrmType
: Indicates the class of a widget created with `MrmFetchWidget`*()* or `MrmFetchWidgetOverride`(). As of Motif 1.2, the types are not defined in any of the Mrm include files, although the OSF documentation states that they are defined in <*Mrm/Mrm.h*>.

Pixel
: An unsigned long integer (defined in <X11/Intrinsic.h>) that serves as a lookup key to a Colormap. If the visual type is PseudoColor, it is implemented as an index to a Colormap; for DirectColor, the RGB values are directly coded into the Pixel value. The Motif pixmap and color routines, as well as some Mrm functions, use Pixel values.

Pixmap
: *An XID* (server resource ID) that represents a two-dimensional array of pixels, used as an offscreen drawable. that is, a drawable with a specified width, height, and depth (number of planes), but no screen coordinates. The Motif pixmap routines, as well as some Mrm functions, use Pixmap parameters.

Position
: A typedef from <*X11/Intrinsic.h*> used to specify a short quantity used for x- and y-coordinates. The Motif string drawing routines, among others, use Position values.

Screen
: A structure that describes the characteristics of a screen (one or more of which make up a display). A pointer to a list of these structures is a member of the Display structure. A pointer to a structure of this type is returned by `XtScreen`() and `XGetWindowAttributes`(). The Motif pixmap routines, among others, as well as some of the Mrm functions, use Screen values. This data type should not be confused with the Screen object in Motif 1.2.

```
typedef struct {
    XExtData        *ext_data;      /* hook for extension to hang data   */
    struct _XDisplay *display;      /* back pointer to display structure */
    Window          root;           /* root window ID                    */
    int             width;          /* width of screen                   */
    int             height;         /* height of screen                  */
    int             mwidth;         /* width in millimeters              */
    int             mheight;        /* height in millimeters             */
    int             ndepths;        /* number of depths possible         */
    Depth           *depths;        /* list of allowable depths on screen*/
    int             root_depth;     /* bits per pixel                    */
    Visual          *root_visual;   /* root visual                       */
    GC              default_gc;     /* GC for the root visual            */
    Colormap        cmap;           /* default Colormap                  */
    unsigned long   white_pixel;
    unsigned long   black_pixel;    /* white and black pixel values      */
    int             max_maps;       /* max  Colormaps                    */
    int             min_maps;       /* min Colormaps                     */
    int             backing_store;  /* Never, WhenMapped, Always         */
    Bool            save_unders;
    long            root_input_mask; /* initial root input mask          */
} Screen;
```

String
: A typedef for char *.

StringTable
: A pointer to a list of Strings.

Time
: A typedef for an unsigned long value (defined in <*X11/X.h*>) that contains a time value in milliseconds. The constant CurrentTime is interpreted as the time in milliseconds since the server was started. The Time data type is used in event structures and as an argument to some Motif clipboard, drag and drop, and text selection routines.

Visual
: A structure that defines a way of using color resources on a particular screen.

VoidProc
: The prototype for the procedure that copies data passed by name to the clipboard. `XmClipboardBeginCopy`() specifies a procedure of this type. It is defined as follows in <*Xm/CutPaste.h*>:

```
typedef void (*VoidProc) (Widget widget, int *data_id, int *private_id,
                                         int   *reason)
```

VoidProc takes four arguments. The first argument, *widget*, is the widget passed to the callback routine, which is the same widget as passed to XmClipboardBeginCopy(). *The data_id* argument is the ID of the data item that is returned by XmClipboardCopy() *and private_id* is the private data passed to XmClipboardCopy(). *The reason* argument takes the value XmCR_CLIPBOARD_DATA_REQUEST, which indicates that the data must be copied to the clipboard, or XmCR_CLIPBOARD_DATA_DELETE, which indicates that the client can delete the data from the clipboard. Although the last three parameters are pointers to integers, the values are read-only and changing them has no effect.

Widget
: A structure returned by calls to create a widget, such as XtAppInitialize*()*, XtCreateWidget(), and XtCreateManagedWidget*()*, as well as the Motif widget creation routines. The members of this structure should not be accessed directly from applications; they should regard it as an opaque pointer. Type Widget *is* actually a pointer to a widget instance structure. Widget code accesses instance variables from this structure.

WidgetClass
: A pointer to the widget class structure, used to identify the widget class in various routines that create widgets or that return information about widgets. Widget class names have the form nameWidgetClass, with the exception of the widget-precursor classes, Object and RectObj, which have the class pointers objectClass and rectObjClass, respectively.

WidgetList
: A pointer to a list of Widgets.

Window
: A resource maintained by the server, and known on the client side only by an integer ID. In Xt, a widget's window can be returned by the XtWindow() macro. Given the window, the corresponding widget can be returned by XtWindowToWidget(). Conversely, given a widget, the window can be deduced through XtWindowOfObject(). The Motif clipboard and string drawing routines use Window values.

XEvent
: A union of all thirty event structures. The first member is always the type, so it is possible to branch on the type, and do event-specific processing in each branch. Both XmDragStart() and XmTrackingEvent() take XEvent parameters. An XButtonPressedEvent, which is one of the event structures in the union, is used by XmMenuPosition().

XFontSet
: Specifies all of the fonts needed to display text in a particular locale. The Motif font list entry routines can use XFontSet values.

XFontStruct
: Specifies metric information (in pixels) for an entire font. This structure (defined in *<X11/Xlib.h>*) is filled by means of the Xlib routines `XLoadQueryFont`() and `XQueryFont`*()*. `XListFontsWithInfo`() also fills it, but with metric information for the entire font only, not for each character. Some of the Motif font list routines use XFontStructs.

```
typedef struct {
    XExtData    *ext_data;          /* hook for extension to hang data      */
    Font        fid;                /* font ID for this font                */
    unsigned    direction;          /* direction the font is painted        */
    unsigned    min_char_or_byte2;  /* first character                      */
    unsigned    max_char_or_byte2;  /* last character                       */
    unsigned    min_byte1;          /* first row that exists                */
    unsigned    max_byte1;          /* last row that exists                 */
    Bool        all_chars_exist;    /* flag if all characters have          */
                                    /* nonzero size                         */
    unsigned    default_char;       /* char to print for undefined character*/
    int         n_properties;       /* how many properties there are        */
    XFontProp   *properties;        /* pointer to array of additional       *\
                                    /* properties                           */
    XCharStruct  min_bounds;        /* minimum bounds over all              */
                                    /* existing char                        */
    XCharStruct  max_bounds;        /* maximum bounds over all              */
                                    /* existing char                        */
    XCharStruct *per_char;          /* first_char to last_char information  */
    int         ascent;             /* logical extent of largest character  */
                                    /* above baseline                       */
    int         descent;            /* logical descent of largest character */
                                    /* below baseline                       */
} XFontStruct;
```

The *direction* member is specified by one of the following constants from *<X11/X.h>:*

FontLeftToRight FontRightToLeft FontChange

XImage

Describes an area of the screen. This structure (defined in <X11/Xlib.h>) is used by `XmInstallImage`() and `XmUninstallImage`().

```
typedef struct _XImage {
    int                 width, height;      /* size of image in pixels          */
    int                 xoffset;            /* number of pixels offset in X direction
    */
    int                 format;             /* XYBitmap, XYPixmap, ZPixmap*/
    char                *data;              /* pointer to image data            */
    int                 byte_order;         /* data byte order: LSBFirst, MSBFirst
    */
    int                 bitmap_unit;        /* quant. of scan line 8, 16, 32    */
    int                 bitmap_bit_order; /* LSBFirst, MSBFirst                 */
    int                 bitmap_pad;         /* 8, 16, 32                      */
    int                 depth;              /* depth of image                 */
    int                 bytes_per_line;     /* accelerator to next line         */
    int                 bits_per_pixel;     /* bits per pixel (ZPixmap only)    */
    unsigned long       red_mask;           /* bits in z arrangement            */
    unsigned long       green_mask;
    unsigned long       blue_mask;
    char                *obdata;            /* hook for object routines to hang on
    */
    struct funcs {                          /* image manipulation routines
    */
        struct _XImage      *(*create_image)(void);
        int                 (*destroy_image)(struct XImage *);
        unsigned long       (*get_pixel)(struct XImage *, int, int);
        int                 (*put_pixel)(struct XImage *, int, int, unsigned int,
                                    unsigned int);
        struct _XImage      *(*sub_image)(struct XImage *, int, int, unsigned
                                    int, unsigned int);
        int                 (*add_pixel)(struct XImage *, long);
    } f;
} XImage;
```

The format *member is specified by one of the following constants defined in <X11/X.h>:*

```
XYBitmap    /* depth 1, XYFormat */
XYPixmap    /* pixmap viewed as stack of planes; depth == drawable depth
*/
ZPixmap     /* pixels in scan-line order; depth == drawable depth       */
```

byte_order and bitmap_bit_order are specified by either LSBFirst *or MSBFirst*, which are defined in *<X11/X.h>*.

XRectangle
Specifies a rectangle. This structure (defined in *<X11/Xlib.h>)* is used by the Motif string drawing routines and `XmGetDisplayRect`().

```
typedef struct {
    short           x, y;
    unsigned short  width, height;
} XRectangle;
```

XmAllocColorProc
The prototype for the per-screen color allocation procedure which is specified through the XmScreen resource XmNcolorAllocationProc. It is defined as follows in <Xm/Screen.h>:

```
typedef void (*XmAllocColorProc)(  Display     *display;
                                   /* connection to the X server              */
                                   Colormap  colormap;
                                   /* Colormap in which to allocate color */
                                   XColor     *color)
                                   /* color to allocate                       */
```

An XmAllocColorProc takes three arguments. The first *display* argument is the connection to the X server. The second argument is the Colormap in which to allocate the required color. The third color argument is where the required color is specified and returned.

XmAnyCallbackStruct
The generic Motif callback structure. It is defined as follows in *<Xm/Xm.h*>:

```
typedef struct {
    int         reason;     /* the reason that the callback was called   */
    XEvent      *event;     /* event structure that triggered callback   */
} XmAnyCallbackStruct;
```

XmArrowButtonCallbackStruct
The callback structure passed to ArrowButton callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;         /* the reason that the callback was called     */
    XEvent      *event;         /* event structure that triggered callback     */
    int         click_count;    /* number of clicks in multi-click sequence    */
} XmArrowButtonCallbackStruct;
```

XmButtonType
An enumerated type that specifies the type of button used in a simple menu creation routine. The valid values for the type are:

XmPUSHBUTTON	XmTOGGLEBUTTON
XmRADIOBUTTON	XmCHECKBUTTON
XmCASCADEBUTTON	XmTITLE
XmSEPARATOR	XmDOUBLE_SEPARATOR

XmButtonTypeTable
A pointer to a list of XmButtonType values.

XmClipboardPendingList
A structure used in calls to `XmClipboardInquirePendingItems`() to specify a data_id/private_id pair. It is defined as follows in <*Xm/CutPaste.h*>:

```
typedef struct {
    long DataId;
    long PrivateId;
} XmClipboardPendingRec, *XmClipboardPendingList;
```

XmColorProc
The prototype for the color calculation procedure used by `XmGetColorCalculation`() and `XmSetColorCalculation`(). It is defined as follows in <*Xm/Xm.h*>:

```
typedef void (*XmColorProc)(
                    XColor    *bg_color,   /* specifies the background color
*/
                    XColor    *fg_color,   /* returns the foreground color
*/
                    XColor    *sel_color, /* returns the select color
*/
                    XColor    *ts_color,   /* returns the top shadow color
*/
                    XColor    *bs_color)   /* returns the bottom shadow color
*/
```

An XmColorProc takes five arguments. The first argument, *bg_color*, is a pointer to an XColor structure that specifies the background color. The red, green, blue, and pixel fields in the structure contain valid values. The rest of the arguments are pointers to XColor structures for the colors that are to be calculated. The procedure fills in the red, green, and blue fields in these structures.

XmComboBoxCallbackStruct

The callback structure passed to ComboBox callback routines. It is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int         reason;         /* reason that the callback was called    */
    XEvent      *event;         /* event that triggered callback          */
    XmString    item_or_text;   /* the selected item                      */
    int         item_position;  /* the index of the item in the list      */
} XmComboBoxCallbackStruct;
```

XmCommandCallbackStruct

The callback structure passed to Command widget callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;     /* the reason that the callback was called    */
    XEvent      *event;     /* event structure that triggered callback    */
    XmString    value;      /* the string contained in the command area   */
    int         length;     /* the size of this string                    */
} XmCommandCallbackStruct;
```

XmContainerOutlineCallbackStruct

The callback structure passed to Container Outline callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int             reason;             /* reason that callback was called */
    XEvent          *event;             /* event that triggered callback   */
    Widget          item;               /* container item associated       */
                                        /* with event                      */
    unsigned char   new_outline_state;  /* the requested state             */
} XmContainerOutlineCallbackStruct;
```

XmContainerSelectCallbackStruct

The callback structure passed to Container Select callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int             reason;                 /* reason that callback was called */
    XEvent          *event;                 /* event that triggered callback   */
    WidgetList      selected_items;         /* the list of selected items      */
    int             selected_item_count;    /* the number of selected items    */
    unsigned char   auto_selection_type;    /* type of selection event         */
} XmContainerSelectCallbackStruct;
```

XmConvertCallbackStruct

The callback structure passed to the XmNconvertCallback routines of widgets when they are asked to convert a selection. It is defined as follows in *<Xm/Transfer.h>*:

```
typedef struct {
    int             reason;         /* reason that callback is invoked          */
    XEvent          *event;         /* event that triggered callback            */
    Atom            selection;      /* requested conversion selection           */
    Atom            target;         /* the conversion target                    */
    XtPointer       source_data;    /* selection source information             */
    XtPointer       location_data;  /* information on data to be transferred    */
    int             flags;          /* input status of the conversion           */
    XtPointer       parm;           /* parameter data for the target            */
    int             parm_format;    /* format of parameter data                 */
    unsigned long   parm_length;    /* number of elements in parameter data     */
    Atom            parm_type;      /* the type of the parameter data           */
    int             status;         /* output status of the conversion          */
    XtPointer       value;          /* returned conversion data                 */
    Atom            type;           /* type of conversion data returned         */
    int             format;         /* format of the conversion data            */
    unsigned long   length;         /* number of elements in conversion data    */
} XmConvertCallbackStruct;
```

XmCutPasteProc

The prototype for the procedure that copies data passed by name to the clipboard. `XmClipboardStartCopy()` specifies a procedure of this type. It is defined as follows in <*Xm/CutPaste.h*>:

```
typedef void (*XmCutPasteProc) (Widget widget, long *data_id, long
*private_id, int *reason)
```

An XmCutPasteProc takes four arguments. The first argument, *widget*, is the widget passed to the callback routine, which is the same widget as passed to `XmClipboardBeginCopy`(). *The data_id* argument is the ID of the data item that is returned by `XmClipboardCopy`() and *private_id* is the private data passed to `XmClipboardCopy`(). *The reason* argument takes the value XmCR_CLIPBOARD_DATA_REQUEST, which indicates that the data must be copied to the clipboard, or XmCR_CLIPBOARD_DATA_DELETE, which indicates that the client can delete the data from the clipboard. Although the last three parameters are pointers, the values are read-only and changing them has no effect.

XmDestinationCallbackStruct

The callback structure passed to the XmNdestinationCallback routines of widgets when they are the destination of a data transfer. It is defined as follows in <*Xm/Transfer.h*>:

```
typedef struct {
    int         reason;           /* reason that callback is invoked          */
    XEvent      *event;           /* event that triggered callback            */
    Atom        selection;        /* requested selection type, as an Atom     */
    XtEnum      operation;        /* the type of transfer requested           */
    int         flags;            /* whether destination and source are same*/
    XtPointer   transfer_id;      /* unique identifier for the request        */
    XtPointer   destination_data;  /* information about the destination       */
    XtPointer   location_data;    /* information about the data               */
    Time        time;             /* time when transfer operation started     */
} XmDestinationCallbackStruct;
```

XmDirection

An enumerated type that specifies a direction, used either in laying out components of a widget, or in rendering compound strings. The valid values for the type are:

XmRIGHT_TO_LEFT_TOP_TO_BOTTOM
XmLEFT_TO_RIGHT_TOP_TO_BOTTOM
XmRIGHT_TO_LEFT_BOTTOM_TO_TOP
XmLEFT_TO_RIGHT_BOTTOM_TO_TOP
XmRIGHT_TO_LEFT
XmLEFT_TO_RIGHT
XmTOP_TO_BOTTOM_RIGHT_TO_LEFT
XmBOTTOM_TO_TOP_RIGHT_TO_LEFT
XmTOP_TO_BOTTOM_LEFT_TO_RIGHT
XmBOTTOM_TO_TOP_LEFT_TO_RIGHT
XmTOP_TO_BOTTOM
XmBOTTOM_TO_TOP
XmDEFAULT_DIRECTION

XmDisplayCallbackStruct

The callback structure passed to Display XmNnoFontCallback and XmNnoRenditionCallback callback routines. It is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int             reason;         /* reason that the callback was called
    */
    XEvent          *event;         /* event that triggered callback      */
    XmRendition     rendition;      /* rendition with a missing font      */
    char            *font_name;     /* font which is not loadable         */
    XmRenderTable   render_table;   /* render table with missing rendition
    */
    XmString        tag;            /* tag of the missing rendition       */
} XmDisplayCallbackStruct;
```

XmDragDropFininshCallbackStruct

The callback structure passed to the XmNdragDropFinishCallback of a DragContext object. It is defined as follows in <*Xm/DragC.h*>:

```
typedef struct {
    int         reason;         /* the reason the callback was called     */
    XEvent      *event;         /* event that triggered callback          */
    Time        timeStamp;      /* time at which operation completed      */
} XmDragDropFinishCallbackStruct, *XmDragDropFinishCallback;
```

XmDragMotionCallbackStruct

The callback structure passed to the XmNdragMotionCallback of a DragContext object. It is defined as follows in <*Xm/DragC.h*>:

```
typedef struct {
    int             reason;         /* reason the callback was called       */
    XEvent          *event;         /* event that triggered callback        */
    Time            timeStamp;      /* timestamp of logical event           */
    unsigned char   operation;      /* current operation                    */
    unsigned char   operations;     /* supported operations                 */
    unsigned char   dropSiteStatus; /* valid, invalid, or none              */
    Position        x;              /* x-coordinate of pointer              */
    Position        y;              /* y-coordinate of pointer              */
} XmDragMotionCallbackStruct, *XmDragMotionCallback;
```

XmDragProcCallbackStruct

The callback structure passed to the XmNdragProc of a drop site. It is defined as follows in <*Xm/DropSMgr.h*>:

```
typedef struct {
    int             reason;         /* reason the callback was called       */
    XEvent          *event;         /* event that triggered callback        */
    Time            timeStamp;      /* timestamp of logical event           */
    Widget          dragContext;    /* DragContext widget associated        */
                                    /* with operation                       */
    Position        x;              /* x-coordinate of pointer              */
    Position        y;              /* y-coordinate of pointer              */
    unsigned char   dropSiteStatus; /* valid or invalid                     */
    unsigned char   operation;      /* current operation                    */
    unsigned char   operations;     /* supported operations                 */
    Boolean         animate;        /* toolkit or receiver does animation   */
} XmDragProcCallbackStruct, *XmDragProcCallback;
```

XmDrawingAreaCallbackStruct

The callback structure passed to DrawingArea callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;     /* reason that the callback was called          */
    XEvent      *event;     /* event that triggered callback;               */
                            /* for XmNresizeCallback, this is NULL          */
    Window      window;     /* the widget's window                          */
} XmDrawingAreaCallbackStruct;
```

XmDrawnButtonCallbackStruct

The callback structure passed to DrawnButton callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;       /* reason that the callback was called      */
    XEvent      *event;       /* event that triggered callback            */
    Window      window;       /* ID of window in which event occurred     */
    int         click_count;  /* number of multi-clicks                   */
} XmDrawnButtonCallbackStruct;
```

XmDropFinishCallbackStruct

The callback structure passed to the XmNdropFinishCallback of a DragContext object. It is defined as follows in *<Xm/DragC.h>*:

```
typedef struct {
    int             reason;           /* reason the callback was called   */
    XEvent          *event;           /* event that triggered callback    */
    Time            timeStamp;        /* time at which drop completed     */
    unsigned char   operation;        /* current operation                */
    unsigned char   operations;       /* supported operations             */
    unsigned char   dropSiteStatus;   /* valid, invalid, or none          */
    unsigned char   dropAction;       /* drop, cancel, help, or interrupt */
    unsigned char   completionStatus; /* success or failure               */
} XmDropFinishCallbackStruct, *XmDropFinishCallback;
```

XmDropProcCallbackStruct

The callback structure passed to the XmNdropProc of a drop site. It is defined as follows in *<Xm/DropSMgr.h>*:

```
typedef struct {
    int             reason;           /* reason callback was called       */
    XEvent          *event;           /* event that triggered callback    */
    Time            timeStamp;        /* timestamp of logical event       */
    Widget          dragContext;      /* DragContext widget associated    */
                                      /* with operation                   */
    Position        x;                /* x-coordinate of pointer          */
    Position        y;                /* y-coordinate of pointer          */
    unsigned char   dropSiteStatus;   /* valid or invalid                 */
    unsigned char   operation;        /* current operation                */
    unsigned char   operations;       /* supported operations             */
    unsigned char   dropAction;       /* drop or help                     */
} XmDropProcCallbackStruct, *XmDropProcCallback;
```

XmDropSiteEnterCallbackStruct

The callback structure passed to the XmNdropSiteEnterCallback of a DragContext object. It is defined as follows in *<Xm/DragC.h>*:

```
typedef struct {
    int             reason;         /* reason the callback was called   */
    XEvent          *event;         /* event that triggered callback    */
    Time            timeStamp;      /* time of crossing event           */
    unsigned char   operation;      /* current operation                */
    unsigned char   operations;     /* supported operations             */
    unsigned char   dropSiteStatus; /* valid, invalid, or none          */
    Position        x;              /* x-coordinate of pointer          */
    Position        y;              /* y-coordinate of pointer          */
} XmDropSiteEnterCallbackStruct, *XmDropSiteEnterCallback;
```

XmDropSiteLeaveCallbackStruct

The callback structure passed to the XmNdropSiteLeaveCallback of a DragContext object. It is defined as follows in *<Xm/DragC.h>*:

```
typedef struct {
    int             reason;         /* reason callback was called       */
    XEvent          *event;         /* event that triggered callback    */
    Time            timeStamp;      /* time of crossing event           */
} XmDropSiteLeaveCallbackStruct, *XmDropSiteLeaveCallback;
```

XmDropStartCallbackStruct

The callback structure passed to the XmNdropStartCallback of a DragContext object. It is defined as follows in *<Xm/DragC.h>*:

```
typedef struct {
    int             reason;         /* reason callback was called       */
    XEvent          *event;         /* event that triggered callback    */
    Time            timeStamp;      /* time at which drag completed     */
    unsigned char   operation;      /* current operation                */
    unsigned char   operations;     /* supported operations             */
    unsigned char   dropSiteStatus; /* valid, invalid, or none          */
    unsigned char   dropAction;     /* drop, cancel, help, or interrupt */
    Position        x;              /* x-coordinate of pointer          */
    Position        y;              /* y-coordinate of pointer          */
    Window          window;         /* internal: not documented         */
    Atom            icchandle;      /* internal: not documented         */
} XmDropStartCallbackStruct, *XmDropStartCallback;
```

XmDropTransferEntryRec
A structure that specifies the targets of a drop operation for a Drop Transfer object. It is defined as follows in *<Xm/DropTrans.h>:*

```
typedef struct {
    XtPointer    client_data;    /* data passed to the transfer proc    */
    Atom         target;         /* target format of the transfer       */
} XmDropTransferEntryRec, *XmDropTransferEntry;
```

XmDropTransferEntry
See XmDropTransferEntryRec.

XmFileSelectionBoxCallbackStruct
The callback structure passed to FileSelectionBox callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int          reason;         /* reason that callback was called     */
    XEvent       *event;         /* event that triggered callback       */
    XmString     value;          /* current value of XmNdirSpec resource */
    int          length;         /* number of bytes in value member     */
    XmString     mask;           /* current value of XmNdirMask resource */
    int          mask_length;    /* number of bytes in mask member      */
    XmString     dir;            /* current base directory              */
    int          dir_length;     /* number of bytes in dir member       */
    XmString     pattern;        /* current search pattern              */
    int          pattern_length; /* number of bytes in pattern member   */
} XmFileSelectionBoxCallbackStruct;
```

XmFontContext
A typedef for a font list context that lets an application access the font list entries and font list tags in a font list. This data type is an opaque structure returned by a call to `XmFontListInitFontContext`(), and is used in subsequent calls to `XmFontListGetNextEntry`(), `XmFontListGetNextFont`() and `XmFontListFreeFontContext`().

XmFontList
A font list contains entries that describe the fonts that are in use. In Motif 1.1, each entry associates a font and a character set. In Motif 1.2, each entry consists of a XmFontListEntry and an associated tag, where the XmFontListEntry specifies a font or a font set. XmFontList is an opaque data type used in calls to font list routines and string manipulation routines. When a Motif compound string is displayed, the font list tag is used to match the string with a font or font set, so that the compound string is displayed appropriately. The font list tag XmFONTLIST_DEFAULT_TAG causes compound strings to be displayed using the font for the current locale.

To specify a font list in a resource file, use the following syntax:

resource_spec: font_entry [, font_entry] ...

The value specification consists of at least one font list entry, with multiple entries separated by commas. Each font_entry specifies a font or a font set and an optional font list entry tag. Use the following syntax to specify a single font:

font_name [= font_list_entry_tag]

To specify the optional tag for a single font, separate the font_name and the font_list_entry_tag by an equal sign (=). Use the following syntax to specify a font set:

font_name [; font_name] ... : [font_list_entry_tag]

Separate multiple font_names with semicolons and end the specification with a colon, followed by the optional tag. A font_name is an X Logical Font Description (XLFD) string. If a font_list_entry_tag is not specified for an entry, XmFONTLIST_DEFAULT_TAG is used.

In Motif 2.0 and later, the XmFontList is considered obsolete, and is replaced by the XmRenderTable. The XmFontList type is maintained for backwards compatibility, and is implemented through a render table.

XmFontListEntry

In Motif 1.2, a font list entry is an element of an XmFontList that specifies a font or a font set. Each XmFontListEntry is associated with a font list entry tag. XmFontListEntry is an opaque type.

In Motif 2.0 and later, the XmFontList and XmFontListEntry are considered obsolete, and are replaced by the XmRenderTable and XmRendition object respectively. The XmFontList and XmFontListEntry types are maintained for backwards compatibility, and are implemented directly through the render table and rendition object.

XmFontType

An enumerated type that specifies the type of entry in a XmFontListEntry. It is defined as follows in <*Xm/Xm.h*>:

```
typedef enum {
    XmFONT_IS_FONT,          /* specifies a font       */
    XmFONT_IS_FONTSET        /* specifies a font set   */
} XmFontType;
```

XmHighlightMode

An enumerated type that defines the kind of text highlighting that results from calls to `XmTextSetHighlight`() and `XmTextFieldSetHighlight`(). It is defined as follows in *<Xm/Xm.h>:*

```
typedef enum {
    XmHIGHLIGHT_NORMAL,          /* no highlighting                */
    XmHIGHLIGHT_SELECTED,        /* highlight in reverse video     */
    XmHIGHLIGHT_SECONDARY_SELECTED
                                 /* highlight by underlining       */
    XmSEE_DETAIL                 /* unused except by abortive      */
                                 /* Motif 2.0 CSText widget        */
} XmHighlightMode;
```

XmICCEncodingStyle

An enumerated type which specifies the way in which compound string tables are converted to and from a text property. It is defined as follows in *<Xm/Xm.h>:*

```
typedef enum {
    XmSTYLE_STRING               = XStringStyle,
    XmSTYLE_COMPOUND_TEXT        = XCompoundTextStyle,
    XmSTYLE_TEXT                 = XTextStyle,
    XmSTYLE_STANDARD_ICC_TEXT    = XStdICCTextStyle,
    XmSTYLE_LOCALE               = 32,
    XmSTYLE_COMPOUND_STRING
} XmICCEncodingStyle;
```

XmIncludeStatus

A typedef for unsigned char that is used to define the way in which compound strings are parsed when a ParseMapping object is applied to an input stream. Variables of this type can have the following values:

```
XmINSERT        /* concatenate XmNsubstitute value to output */
                /* parsing is continued */
XmINVOKE        /* result determined by XmNinvokeParseProc   */
XmTERMINATE     /* concatenate XmNsubstitute value to output */
                /* parsing is termi nated                    */
```

XmKeySymTable

A pointer to a list of KeySyms.

XmListCallbackStruct

The callback structure passed to List widget callback routines. It is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int         reason;                 /* reason that callback was called */
    XEvent      *event;                 /* event that triggered callback   */
    XmString    item;                   /* item most recently selected at  */
                                        /* the time event occurred         */
    int         item_length;            /* number of bytes in item member*/
    int         item_position;          /* item position in XmNitems array*/
    XmString    *selected_items;        /* list of items selected at the   */
                                        /* time event occurred             */
    int         selected_item_count;    /* number of items in              */
                                        /* selected_items                  */
    int         *selected_item_positions; /* array of integers marking     */
                                        /* selected items                  */
    char        selection_type;         /* type of the most recent         */
                                        /* selection                       */
    char        auto_selection_type;    /* 2.0 or later: automatic         */
                                        /* selection type                  */
} XmListCallbackStruct;
```

The structure members event, item, item_length, and item_position are valid for any value of reason. The structure members selected_items, selected_item_count, and selected_item_ positions are valid when the reason field has a value of XmCR_MULTIPLE_SELECT or XmCR_EXTENDED_SELECT. The structure member selection_type is valid only when the reason field is XmCR_EXTENDED_SELECT.

For the strings pointed to by item and selected_items, as well as for the integers pointed to by selected_item_positions, storage is overwritten each time the callback is invoked. Applications that need to save this data should make their own copies of it.

selected_item_positions is an integer array. The elements of the array indicate the positions of each selected item within the List widget's XmNitems array.

selection_type specifies what kind of extended selection was most recently made. One of three values is possible, defined in <*Xm/List.h*>:

```
XmINITIAL       /* selection was the initial selection              */
XmMODIFICATION  /* selection changed an existing selection          */
XmADDITION      /* selection added non-adjacent items to an        */
                /* existing selection                               */
```

auto_selection_type specifies at what point within the selection the user is. Possible values, defined in *<Xm/Xm.h>:*

XmAUTO_UNSET	XmAUTO_BEGIN
XmAUTO_MOTION	XmAUTO_CANCEL
XmAUTO_NO_CHANGE	XmAUTO_CHANGE

XmMergeMode

An enumerated type that specifies the way in which renditions are merged into a render table. The valid values for the type are:

XmSKIP	XmMERGE_REPLACE
XmMERGE_OLD	XmMERGE_NEW
XmDUPLICATE	

XmDUPLICATE is an internal value used in mapping XmFontList and XmFontListEntry types to the render table types of Motif 2.0 and later.

XmNavigationType

An enumerated type that specifies the type of keyboard navigation associated with a widget. The valid values for the type are:

XmNONE	XmTAB_GROUP
XmSTICKY_TAB_GROUP	XmEXCLUSIVE_TAB_GROUP

XmNotebookCallbackStruct

The callback structure passed to Notebook selection callback routines. It is defined as follows in *<Xm/Xm.h>*:

```
typedef struct {
    int         reason;             /* reason that callback was called  */
    XEvent      *event;             /* points to event structure        */
                                    /* that triggered callback          */
    int         page_number;        /* current logical page number      */
    Widget      page_widget;        /* widget associated with           */
                                    /* current logical page number      */
    int         prev_page_number;   /* previous logical page number     */
    Widget      prev_page_widget;   /* widget associated with           */
                                    /* previous logical page number     */
} XmNotebookCallbackStruct;
```

XmNotebookPageInfo
Specifies a structure passed to the function `XmNotebookGetPageInfo`() in order to retrieve information about a Notebook page. It is defined as follows in <*Xm/Notebook.h*>:

```
typedef struct {
    int         page_number;        /* logical page number          */
    Widget      page_widget;        /* widget ID of a page child    */
    Widget      status_area_widget; /* widget ID of a status area child */
    Widget      major_tab_widget;   /* widget ID of a major tab child */
    Widget      minor_tab_widget;   /* widget ID of a minor tab child */
} XmNotebookPageInfo;
```

XmOffset
A long integer that represents the units used in calculating the offsets into a widget's instance data. The type is used internally to Motif. See also XmOffsetPtr.

XmOffsetModel
An enumerated type that specifies whether tabs are calculated at absolute offsets, or relative to the previous tab. The valid values for the type are:

```
XmABSOLUTE    XmRELATIVE
```

XmOffsetPtr
A pointer to an XmOffset value, which is returned by a calls to `XmResolveAllPartOffsets`() and `XmResolvePartOffsets`().

XmOperationChangedCallbackStruct
The callback structure passed to the XmNoperationChangedCallback of a DragContext object. It is defined as follows in <*Xm/DragC.h*>:

```
typedef struct {
    int             reason;         /* reason callback was called   */
    XEvent          *event;         /* event that triggered callback */
    Time            timeStamp;      /* timestamp of logical event   */
    unsigned char   operation;      /* current operation            */
    unsigned char   operations;     /* supported operations         */
    unsigned char   dropSiteStatus; /* valid, invalid, or none      */
} XmOperationChangedCallbackStruct, *XmOperationChangedCallback;
```

XmParseMapping
A typedef for a parse mapping object that lets an application control the way in which an input stream of bytes is converted into the components or segments within a compound string. This data type is an opaque structure returned by a call to `XmParseMappingCreate`(), and is placed into an XmParseTable and used in subsequent calls to the string manipulation routines: `XmStringParse-`

`Text`(), `XmStringTableParseStringArray`(), and `XmStringTable-Unparse`(), and `XmStringUnparse`().

XmParseModel
An enumerated type which specifies how non-text components of a compound string are unparsed. It is defined as follows in <*Xm/Xm.h*>:

```
typedef enum {
    XmOUTPUT_ALL,
    XmOUTPUT_BETWEEN,
    XmOUTPUT_BEGINNING,
    XmOUTPUT_END,
    XmOUTPUT_BOTH
} XmParseModel;
```

This data type is used in calls to the following compound string routines: `XmStringTableUnparse`(), and `XmStringUnparse`().

XmParseProc
A procedure within an XmParseMapping object for controlling the way in which an input stream is parsed into a compound string. It is defined as follows in <*Xm/Xm.h*>:

```
typedef XmIncludeStatus (*XmParseProc) (    XtPointer           *in_out,
                                            XtPointer           text_end,
                                            XmTextType          type,
                                            XmStringTag         locale_tag,
                                            XmParseMapping      entry,
                                            int             pattern_length,
                                            XmString          *str_include,
                                            XtPointer           call_data);
```

XmParseTable
A typedef for an array of parse mapping objects, used for parsing an input stream into a compound strings.

```
typedef XmParseMapping *XmParseTable;
```

XmPushButtonCallbackStruct
The callback structure passed to PushButton callback routines. It is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int         reason;         /* reason that callback was called      */
    XEvent      *event;         /* event that triggered callback        */
    int         click_count;    /* number of multi-clicks               */
} XmPushButtonCallbackStruct;
```

XmPopupHandlerCallbackStruct

The callback structure passed to Popup Handler callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;         /* the reason the callback is invoked      */
    XEvent      *event;         /* event that triggered callback           */
    Widget       menuToPost;    /* the menu to post                        */
    Boolean     postIt;         /* whether to continue posting             */
    Widget      target;         /* manager descendant issuing request      */
} XmPopupHandlerCallbackStruct;
```

XmPrintShellCallbackStruct

The callback structure passed to PrintShell callback routines. It is defined as follows in <Xm/Xm.h>:

```
typedef struct {
    int         reason;         /* reason that the callback is invoked     */
    XEvent      *event;         /* event that triggered the callback       */
    XPContext   context;        /* X Print Context                         */
    Boolean     last_page;      /* whether this is the last page           */
    XtPointer   detail;         /* PDM selection                           */
} XmPrintShellCallbackStruct;
```

XmQualifyProc

The prototype for the qualification procedure that produces a qualified directory mask, base directory, and search pattern for the directory and file search procedures in a FileSelectionBox. The XmNqualifySearchDataProc resource specifies a procedure of this type, which is defined as follows in *<Xm/FileSB.h>*:

typedef void (*XmQualifyProc) (Widget widget, XtPointer input_data, XtPointer output_data)

An XmQualifyProc takes three arguments. The first argument, widget, is the FileSelectionBox widget. The input_data argument is a pointer to an XmFileSelectionBoxCallbackStruct that contains input data to be qualified. The output_data argument is a pointer to an XmFileSelectionBoxCallbackStruct that is to be filled in by the qualification procedure.

XmRendition

An opaque data structure, implemented as a pseudo-widget, which encapsulates the resources required to render a compound string. This data type is returned by a call to `XmRenditionCreate`(), and is used in subsequent calls to the following routines: `XmRenderTableAddRenditions`(), `XmRenditionFree`(), `XmRenditionRetrieve`(), `XmRenditionUpdate`().

XmRenderTable
An opaque data structure, representing a list of XmRendition objects, used to render compound strings. Typically used as the XmNrenderTable resource of a widget, the type us used in calls to the following routines: `XmRenderTableCopy`(), `XmRenderTableFree`(), `XmRenderTableGetRendition`(), `XmRenderTableGetRenditions`(), `XmRenderTableGetTags`(), `XmRenderTableRemoveRenditions`().

XmRepTypeEntry
A pointer to a representation type entry structure which contains information about the value names and values for an enumerated type. The Motif representation type manager routines use values of this type, which is defined as follows in <*Xm/RepType.h*>:

```
typedef struct {
    String          rep_type_name;      /* name of representation type      */
    String          *value_names;       /* array of value names             */
    unsigned char   *values;            /* array of numeric values          */
    unsigned char   num_values;         /* number of values                 */
    Boolean         reverse_installed;  /* reverse converter installed flag */
    XmRepTypeId     rep_type_id;        /* representation type ID           */
} XmRepTypeEntryRec, *XmRepTypeEntry, XmRepTypeListRec,
                                                *XmRepTypeList;
```

XmRepTypeId
An unsigned short that specifies the identification number of a representation type registered with the representation type manager. The representation type manager routines use values of this type.

XmRepTypeList
See XmRepTypeEntry.

XmRowColumnCallbackStruct
The callback structure passed to RowColumn callback routines. It is only used by map and unmap callbacks, and is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int         reason;         /* reason that callback was called      */
    XEvent      *event;         /* event that triggered callback        */
    Widget      widget;         /* ID of activated RowColumn item       */
    char        *data;          /* value of application's client data   */
    char        *callbackstruct; /* created when item is activated      */
} XmRowColumnCallbackStruct;
```

widget, *data*, and callbackstruct *are set to NULL*.

XmScaleCallbackStruct

The callback structure passed to Scale widget callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;     /* reason that callback was called     */
    XEvent      *event;     /* event that triggered callback       */
    int         value;      /* new value of the slider             */
} XmScaleCallbackStruct;
```

XmScreenColorProc

The prototype for the per-screen color calculation procedure which is specified through the XmScreen resource XmNcolorCalculationProc. It is defined as follows in *<Xm/Screen.h>*:

```
typedef void (*XmScreenColorProc)(
                Screen      *screen,    /* screen of top-level window          */
                XColor      *bg_color,  /* specifies the background color      */
                XColor      *fg_color,  /* returns the foreground color        */
                XColor      *sel_color, /* returns the select color            */
                XColor      *ts_color,  /* returns the top shadow color        */
                XColor      *bs_color)  /* returns the bottom shadow color     */
```

An XmScreenColorProc takes six arguments. The first argument is a pointer to a screen object. The second argument, bg_color, is a pointer to an XColor structure that specifies the background color. The red, green, blue, and pixel fields in the structure contain valid values. The rest of the arguments are pointers to XColor structures for the colors that are to be calculated. The procedure fills in the red, green, and blue fields in these structures.

XmScrollBarCallbackStruct

The callback structure passed to ScrollBar callback routines. It is defined as follows in *<Xm/Xm.h>*:

```
typedef struct {
    int         reason;     /* reason that callback was called     */
    XEvent      *event;     /* event that triggered callback       */
    int         value;      /* value of the slider's new location  */
    int         pixel;      /* coordinate where selection occurred */
} XmScrollBarCallbackStruct;
```

XmSearchProc

The prototype for a search procedure that searches the directories or files in a FileSelectionBox. The XmNdirSearchProc and XmNfileSearchProc resources specify procedures of this type, which is defined as follows in *<Xm/FileSB.h>:*

```
typedef void (*XmSearchProc) (Widget widget, XtPointer search_data)
```

An XmSearchProc takes two arguments. The first argument, widget, is the FileSelectionBox widget. The search_data argument is a pointer to an XmFileSelectionBoxCallbackStruct that contains the information for performing a search.

XmSecondaryResourceData

A structure that specifies information about secondary resources associated with a widget class. `XmGetSecondaryResourceData`() returns an array of these values. The type is defined as follows in *<Xm/Xm.h>*:

```
typedef struct {
    XmResourceBaseProc  base_proc;
    XtPointer           client_data;
    String              name;
    String              res_class;
    XtResourceList      resources;
    Cardinal            num_resources;
} XmSecondaryResourceDataRec, *XmSecondaryResourceData;
```

XmSelectionBoxCallbackStruct

The callback structure passed to SelectionBox callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;     /* reason that callback was called              */
    XEvent      *event;     /* event that triggered callback                */
    XmString    value;      /* selection string that was either chosen      */
                            /* from the SelectionBox list or typed in       */
    int         length;     /* number of bytes of value                     */
} XmSelectionBoxCallbackStruct;
```

XmSelectionCallbackStruct

The callback structure passed to routines which are responsible for data transfer from the primary selection. The function `XmTransferValue`() takes as its third parameter a procedure which is responsible for inserting data into the desti-

nation. The procedure receives a pointer to an XmSelectionCallbackStruct as callback data when invoked. It is defined as follows in *<Xm/Transfer.h>:*

```
typedef struct {
    int             reason;       /* reason the callback was invoked          */
    XEvent          *event;       /* event which triggered callback           */
    Atom            selection;    /* selection that has been converted        *
    Atom            target;       /* target for which conversion requested    */
    Atom            type;         /* type of the selection value              */
    XtPointer       transfer_id;  /* unique identifier for transfer operation */
    int             flags;        /* unused: pass constant                    */
                                  /*  XmSELECTION_DEFAULT                     */
    int             remaining;    /* number of transfers remaining in         */
                                  /* operation                                */
    XtPointer       value;        /* the data transferred in this request     */
    unsigned long   length;       /* the number of elements in the value      */
    int             format;       /* size of each element in the value        */
} XmSelectionCallbackStruct;
```

XmSpinBoxCallbackStruct

The callback structure passed to SpinBox callback routines. It is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    int         reason;           /* the reason that the callback was called  */
    XEvent      *event;           /* points to event that triggered callback  */
    Widget      widget;           /* the textual child affected by callback   */
    Boolean     doit;             /* whether to perform the changes           */
    int         position;         /* specifies the index of the next value    */
    XmString    value;            /* specifies the next value                 */
    Boolean     crossed_boundary; /* whether the SpinBox has wrapped
    */
} XmSpinBoxCallbackStruct;
```

XmString

The data type for Motif compound strings. In Motif 1.2, a compound string is composed of one or more segments, where each segment can contain a font list element tag, a string direction, and a text component. The font list element tag XmFONTLIST_DEFAULT_TAG specifies a text segment encoded in the current locale. In Motif 1.1, compound strings use character set identifiers rather than font list element tags. The character set identifier for a compound string can have the value XmSTRING_DEFAULT_CHARSET, which takes the character set from the current language environment, but this value may be removed from future versions of Motif.

XmStringCharSet
A typedef for char * that is used to define the character set of a compound string in Motif 1.1. Variables of this type can have the following values, among others:

XmSTRING_ISO8859_1
XmSTRING_OS_CHARSET
XmSTRING_DEFAULT_CHARSET

XmSTRING_DEFAULT_CHARSET specifies the character set from the current language environment, but this value may be removed from future versions of Motif.

XmStringCharSetTable
A pointer to a list of XmStringCharSets.

XmStringComponentType
An unsigned char value that specifies the type of component in a compound string segment. Values of this type are returned by calls to `XmStringGetNextComponent`() and `XmStringPeekNextComponent`(). The valid values for the type are:

```
XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG
                                /* font list element tag component     */
                                /* obsolete in Motif 2.0               */
XmSTRING_COMPONENT_CHARSET
                                /* character set identifier component; */
                                /* obsolete in Motif 1.2               */
XmSTRING_COMPONENT_TEXT         /* text component                      */
XmSTRING_COMPONENT_LOCALE_TEXT
                                /* locale-encoded text component       */
XmSTRING_COMPONENT_DIRECTION
                                /* direction component                 */
XmSTRING_COMPONENT_SEPARATOR
                                /* separator component                 */
XmSTRING_COMPONENT_END          /* last component in string            */
XmSTRING_COMPONENT_UNKNOWN
                                /* unknown component                   */
XmSTRING_COMPONENT_LOCALE
                                /* the locale specifier                */
XmSTRING_COMPONENT_WIDECHAR_TEXT
                                /* widechar text component             */
XmSTRING_COMPONENT_LAYOUT_PUSH
                                /* stacked layout direction            */
XmSTRING_COMPONENT_LAYOUT_POP
                                /* unstacked layout component          */
```

```
    XmSTRING_COMPONENT_RENDITION_BEGIN
                                    /* beginning of rendition          */
    XmSTRING_COMPONENT_RENDITION_END
                                    /* end of rendition                */
    XmSTRING_COMPONENT_TAG          /* charset/font list tag component */
    XmSTRING_COMPONENT_TAB          /* tab component                   */
```

XmStringContext
: A typedef for a string context that lets an application access the components or segments within a compound string. This data type is an opaque structure returned by a call to `XmStringInitContext`(), and is used in subsequent calls to the four other string context routines: `XmStringFreeContext`(), `XmStringGetNextSegment`(), `XmStringGetNextComponent`(), and `XmStringPeekNextComponent`().

`XmStringDirection`
: An unsigned char used for determining the direction in which a compound string is displayed. The type is used in calls to `XmStringDirectionCreate`() and `XmStringSegmentCreate`(). The valid values for the type are:

    XmSTRING_DIRECTION_L_TO_R
    XmSTRING_DIRECTION_R_TO_L
    XmSTRING_DIRECTION_DEFAULT

XmStringTable
: An opaque typedef for XmString * that is used for arrays of compound strings.

XmStringTag
: A typedef for char * that is used to specify the tag which identifies components or segments within a compound string. This data type is used in calls to the following compound string routines: `XmRenderTableCopy`(), `XmRenderTableGetRendition`(), `XmRenderTableGetRenditions`(), `XmRenderTableGetTags`(), `XmRenderTableRemoveRenditions`(), `XmRenditionCreate`(), `XmRenditionRetrieve`(), `XmStringGenerate`(), `XmStringParseText`(), `XmStringPutRendition`(), `XmStringTableParseStringArray`(), `XmStringTableUnparse`(), and `XmStringUnparse`().

XmTab
: Specifies a tab stop, which is used to lay out compound strings within a columns. This data type is an opaque structure returned by a call to `XmTabCreate`(), and is used in calls to the following tab routines: `XmTabGetValues`(), `XmTabFree`(), `XmTabListInsertTabs`()

XmTabList
Specifies a list of tab stops, which are used to lay out compound strings within a columns. This data type is an opaque structure returned by a call to XmTabListInsertTabs(), and is used in calls to the following tab routines: XmTabListReplacePositions(), XmTabListRemoveTabs(), XmTabListGetTab(), XmTabListTabCount(), XmTabListCopy(), XmTabListFree(), and XmTabListInsertTabs().

XmTextBlockRec
A structure that specifies information about a block of text in a Text or TextField widget. The text field in an XmTextVerifyCallbackStruct points to a structure of this type, which is defined as follows in *<Xm/Xm.h>*:

```
typedef struct {
    char            *ptr;     /* pointer to the text to insert      */
    int             length;   /* length of this text                */
    XmTextFormat    format;   /* text format (e.g., FMT8BIT, FMT16BIT) */
} XmTextBlockRec, *XmTextBlock;
```

XmTextBlockRecWcs
A structure that specifies information about a block of text in wide-character format in a Text or TextField widget. The text field in an XmTextVerifyCallbackStructWcs points to a structure of this type, which is defined as follows in *<Xm/Xm.h>:*

```
typedef struct {
    wchar_t    *wcsptr;   /* pointer to text to insert      */
    int        length;    /* length of this text            */
} XmTextBlockRecWcs, *XmTextBlockWcs;
```

XmTextDirection
An enumerated type that specifies the search direction in calls to XmTextFindString() and XmTextFindStringWcs(). It is defined as follows in *<Xm/Xm.h>:*

```
typedef enum {
    XmTEXT_FORWARD,        /* search forward     */
    XmTEXT_BACKWARD        /* search backward    */
} XmTextDirection;
```

XmTextPosition
A long integer, used by Text and TextField routines for determining the position of a character inside the text string.

XmTextSource
A pointer to an opaque structure that specifies a text source. The type is used in calls to XmTextGetSource() and XmTextSetSource().

XmTextType

An enumerated type which specifies the type of data contained within an input stream. It is defined as follows in <*Xm/Xm.h*>:

```
typedef enum {
    XmCHARSET_TEXT,
    XmMULTIBYTE_TEXT,
    XmWIDECHAR_TEXT,
    XmNO_TEXT
} XmTextType;
```

This data type is used in calls to the following compound string routines: `XmParseMappingGetValues`(), `XmParseMappingSetValues`(), `XmStringGenerate`(), `XmStringParseText`(), `XmStringTableParseStringArray`(), `XmStringTableUnparse`(), and `XmStringUnparse`().

XmTextVerifyCallbackStruct

The callback structure passed to the XmNlosingFocusCallback, XmNmodifyVerifyCallback, and XmNmotionVerifyCallback callback routines of Text and TextField widgets. It is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int             reason;     /* reason that callback was called       */
    XEvent          *event;     /* event that triggered callback         */
    Boolean         doit;       /* do the action (True) or undo it (False) */
    long            currInsert; /* the insert cursor's current position  */
    long            newInsert;  /* desired new position of insert cursor */
    long            startPos;   /* start of text to change               */
    long            endPos;     /* end of text to change                 */
    XmTextBlock     text;       /* describes the text to insert          */
} XmTextVerifyCallbackStruct, *XmTextVerifyPtr;
```

start_pos *specifies the location at which to start modifying text. start_pos* is unused if the callback resource is XmNmotionVerifyCallback, *and is the same as the current_insert* member if the callback resource is XmNlosingFocusCallback.

end_pos specifies the location at which to stop modifying text (however, if no text was modified, end_pos has the same value as start_pos). end_pos is unused if the callback resource is XmNmotionVerifyCallback, and is the same as the current_insert member if the callback resource is XmNlosingFocusCallback.

XmTextVerifyCallbackStructWcs

The callback structure passed to the XmNmodifyVerifyCallbackWcs of Text and TextField widgets. It is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int             reason;     /* reason that callback was called          */
    XEvent          *event;     /* event that triggered callback            */
    Boolean         doit;       /* do the action (True) or undo it (False)*/
    long            currInsert; /* the insert cursor's current position     */
    long            newInsert;  /* desired new position of insert cursor    */
    long            startPos;   /* start of text to change                  */
    long            endPos;     /* end of text to change                    */
    XmTextBlockWcs  text;       /* describes the text to insert             */
} XmTextVerifyCallbackStructWcs, *XmTextVerifyPtrWcs;
```

All of the fields in this structure are the same as the fields in the XmTextVerify-CallbackStruct *except text*, which points to a XmTextBlockRecWcs *structure.*

XmToggleButtonCallbackStruct

The callback structure passed to ToggleButton callback routines. It is defined as follows in <*Xm/Xm.h*>:

```
typedef struct {
    int         reason;     /* reason that callback was called      */
    XEvent      *event;     /* event that triggered callback        */
    int         set;        /* selection state of the toggle        */
} XmToggleButtonCallbackStruct;
```

XmToggleButtonState

An enumerated type that specifies the state of a ToggleButton. The valid values for the type are:

XmUNSET XmSET XmINDETERMINATE

XmTopLevelEnterCallbackStruct

The callback structure passed to the XmNtopLevelEnterCallback of a DragContext object. It is defined as follows in <*Xm/DragC.h*>:

```
typedef struct {
    int             reason;             /* reason callback was called       */
    XEvent          *event;             /* event that triggered callback    */
    Time            timestamp;          /* timestamp of logical event       */
    Screen          screen;             /* screen of top-level window       */
    Window          window;             /* window being entered             */
    Position        x;                  /* x-coordinate of pointer          */
    Position        y;                  /* y-coordinate of pointer          */
    unsigned char   dragProtocolStyle;  /* drag protocol of initiator       */
    Atom            iccHandle;          /* internal: not documented         */
} XmTopLevelEnterCallbackStruct, *XmTopLevelEnterCallback;
```

XmTopLevelLeaveCallbackStruct

The callback structure passed to the XmNtopLevelLeaveCallback of a DragContext object. It is defined as follows in <*Xm/DragC.h*>:

```
typedef struct {
    int         reason;     /* reason callback was called       */
    XEvent      *event;     /* event that triggered callback    */
    Time        timestamp;  /* timestamp of logical event       */
    Screen      screen;     /* screen of top-level window       */
    Window      window;     /* window being left                */
} XmTopLevelLeaveCallbackStruct, *XmTopLevelLeaveCallback;
```

XmTransferStatus

An enumerated type that specifies the status of a data transfer operation. The value is passed as a parameter to `XmTransferDone`() in order to terminate current data transfer. The valid values for the type are:

XmTRANSFER_DONE_SUCCEED
XmTRANSFER_DONE_CONTINUE
XmTRANSFER_DONE_FAIL
XmTRANSFER_DONE_DEFAULT

XmTraversalDirection

An enumerated type that specifies direction of traversal in a XmTraverseObscuredCallbackStruct. It is defined as follows in *<Xm/Xm.h>*:

```
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
    XmTRAVERSE_GLOBALLY_FORWARD      /* 2.0 */,
    XmTRAVERSE_GLOBALLY_BACKWARD     /* 2.0 */
} XmTraversalDirection;
```

XmTraverseObscureCallbackStruct

The callback structure passed to the XmNtraverseObscuredCallback of a ScrolledWindow widget. It is defined as follows in *<Xm/Xm.h>*:

```
typedef struct {
    int                 reason;         /* reason the callback was called */
    XEvent              *event;         /* event that triggered callback  */
    Widget              traversal_destination;
                                        /* widget or gadget to traverse to */
    XmTraversalDirection direction;     /* direction of traversal         */
} XmTraverseObscuredCallbackStruct;
```

XmVisibility
An enumerated type that specifies the visibility state of a widget. A value of type XmVisibility is returned by `XmGetVisibility`(). It is defined as follows in *<Xm/Xm.h>:*

```
typedef enum {
    XmVISIBILITY_UNOBSCURED,          /* completely visible  */
    XmVISIBILITY_PARTIALLY_OBSCURED,  /* partially visible   */
    XmVISIBILITY_FULLY_OBSCURED       /* not visible         */
} XmVisibility;
```

XrmValue
A structure defined in *<X11/Xresource.h>*, used in `XtConvert`() and other resource conversion routines:

```
typedef struct {
    unsigned int    size;
    XPointer        addr;
} XrmValue, *XrmValuePtr;
```

XrmValuePtr
See XrmValue.

XtAccelerators
A pointer to an opaque internal type, a compiled accelerator table. A pointer to an XtAccelerators structure is returned by a call to `XtParseAcceleratorTable`(). Usually, the compiled accelerator table is produced automatically by resource conversion of a string accelerator table stored in a resource file.

XtCallbackList
A structure defined as follows in *<X11/Intrinsic.h>*:

```
typedef struct _XtCallbackRec {
    XtCallbackProc      callback;
    XtPointer           closure;
} XtCallbackRec, *XtCallbackList;
```

Applications which use `XtAddCallback`() or `XtRemoveCallback`() do not need to use the XtCallbackList type. It can, however, be used to set a callback resource by passing the structure to `XtCreateWidget`() or `XtSetValues`(). Any structure so defined should be declared static. In most documentation, the closure member is referred to as client_data.

XtCallbackProc

The prototype for callback functions. It is defined as follows in *<X11/Intrinsic.h>*:

```
typedef void (*XtCallbackProc) (Widget widget, XtPointer client_data,
XtPointer call_data)
```

XtConvertSelectionIncrProc

The prototype for an incremental selection conversion procedure. The XmNconvertProc for a DragContext object is of this type, which is defined as follows in *<X11/Intrinsic.h>*:

```
typedef Boolean (*XtConvertSelectionIncrProc)(
                                    Widget          widget,
                                    Atom            *selection,
                                    Atom            *target,
                                    Atom            *type_return,
                                    XtPointer       *value_return,
                                    unsigned long   *length_return,
                                    int             *format_return,
                                    unsigned long   *max_length,
                                    XtPointer       client_data,
                                    XtRequestId     *request_id)
```

XtCreatePopupChildProc

The prototype for a procedure that pops up the child of a shell when the shell is popped up. The XmNcreatePopupChildProc resource of Shell specifies a procedure of this type, which is defined as follows in *<X11/Intrinsic.h>*:

```
typedef void (*XtCreatePopupChildProc) (Widget shell)
```

XtKeyProc

The prototype for a keycode-to-keysym translation procedure. `XmTranslateKey`() is the default XtKeyProc for Motif applications. The prototype is defined as follows in *<X11/Intrinsic.h>*:

```
typedef void (*XtKeyProc)(  Display     *display,
                            KeyCode     keycode,
                            Modifiers   modifiers,
                            Modifiers   *modifiers_return,
                            KeySym      *keysym_return)
```

XtOrderProc
: The prototype for a procedure that allows composite widgets to order their children. The XmNinsertPosition resource of Composite specifies a procedure of this type, which is defined as follows in *<X11/Composite.h>*:

```
typedef Cardinal (*XtOrderProc) (Widget child)
```

XtPointer
: A datum large enough to contain the largest of a char*, int*, function pointer, structure pointer, or long value. A pointer to any type or function, or a long, may be converted to an XtPointer and back again and the result will compare equally to the original value. In ANSI-C environments, it is expected that XtPointer will be defined as void *.

XtSelectionCallbackProc
: The prototype for a selection callback procedure. The XmNtransferProc for a DropTransfer object is of this type, and is defined as follows in *<X11/Intrinsic.h>*:

```
typedef void (*XtSelectionCallbackProc)(    Widget          widget,
                                            XtPointer       client_data,
                                            Atom            *selection,
                                            Atom            *type,
                                            XtPointer       value,
                                            unsigned long   *length,
                                            int             *format0
```

XtTranslations
: A pointer to an opaque internal type, a compiled translation table. A pointer to an XtTranslations structure is returned by a call to `XtParseTranslationTable`(). Usually, the compiled translation table is produced automatically by resource conversion of a string translation table stored in a resource file.

Appendix C - Table of Motif Resources

This appendix lists all of the resources for the widget classes provided by the Motif toolkit and the X Toolkit Intrinsics. The table lists the appropriate data types for specifying each resource with both Motif and UIL. For resources that cannot be specified in UIL, the table entry indicates that the resource is not applicable (NA). The table also specifies the widget classes that define each resource. If a widget class has a corresponding gadget class, the table lists only the widget class as defining resources, even though the resources pertain to both the widget and gadget classes. For more information on each resource, see the appropriate reference pages in Section 2, *Motif and Xt Widget Classes*.

Resource Name	Motif Type	UIL Type	Defined in Class
XmNacceleratorText	XmString	compound_string	XmLabel
XmNaccelerators	XtAccelerators	translation_table	Core
XmNactivateCallback	XtCallbackList	procedure	XmArrowButton XmCascadeButton XmDrawnButton XmPushButton XmText XmTextField
XmNadjustLast	Boolean	boolean	XmRowColumn
XmNadjustMargin	Boolean	boolean	XmRowColumn
XmNalignment	unsigned char	integer	XmIconGadget XmLabel XmTab
XmNallowOverlap	Boolean	boolean	XmBulletinBoard
XmNallowResize	Boolean	boolean	XmPanedWindow
XmNallowShellResize	Boolean	boolean	Shell
XmNancestorSensitive	Boolean	boolean	Core XmRectObj
XmNanimationMask	Pixmap	NA	XmDropSite
XmNanimationPixmap	Pixmap	NA	XmDropSite
XmNanimationPixmapDepth	int	NA	XmDropSite
XmNanimationStyle	unsigned char	NA	XmDropSite

Resource Name	Motif Type	UIL Type	Defined in Class
XmNapplyCallback	XtCallbackList	procedure	XmSelectionBox
XmNapplyLabelString	XmString	compound_string	XmSelectionBox
XmNargc	int	NA	ApplicationShell
XmNargv	String *	NA	ApplicationShell
XmNarmCallback	XtCallbackList	procedure	XmArrowButton XmDrawnButton XmPushButton XmToggleButton
XmNarmColor	Pixel	color	XmPushButton
XmNarmPixmap	Pixmap	pixmap	XmPushButton
XmNarrowDirection	unsigned char	integer	XmArrowButton
XmNarrowLayout	unsigned char	integer	XmSpinBox
XmNarrowOrientation	unsigned char	integer	XmSpinBox
XmNarrowSensitivity	unsigned char	integer	XmSpinBox XmSimpleSpinBox
XmNarrowSize	int	integer	XmComboBox XmSpinBox
XmNarrowSpacing	int	integer	XmComboBox
XmNattachment	unsigned char	NA	XmDragIcon
XmNaudibleWarning	unsigned char	integer	VendorShell
XmNautoDragModel	XtEnum	integer	XmScrolledWindow
XmNautoShowCursorPosition	Boolean	boolean	XmText
XmNautoUnmanage	Boolean	boolean	XmBulletinBoard
XmNautomaticSelection	unsigned char	integer	XmContainer XmList
XmNbackPageBackground	Pixel	color	XmNotebook
XmNbackPageForeground	Pixel	color	XmNotebook
XmNbackPageNumber	int	integer	XmNotebook
XmNbackPagePlacement	unsigned char	integer	XmNotebook
XmNbackPageSize	Dimension	integer	XmNotebook
XmNbackground	Pixel	color	Core XmGadget

Appendix C: Table of Motif Resources

Resource Name	Motif Type	UIL Type	Defined in Class
XmNbackgroundPixmap	Pixmap	pixmap	Core XmGadget
XmNbaseHeight	int	integer	WMShell
XmNbaseWidth	int	integer	WMShell
XmNbindingPixmap	Pixmap	pixmap	XmNotebook
XmNbindingType	unsigned char	integer	XmNotebook
XmNbindingWidth	int	integer	XmNotebook
XmNbitmapConversionModel	XtEnum	NA	XmScreen
XmNblendModel	unsigned char	NA	XmDragContext
XmNblinkRate	int	integer	XmText XmTextField
XmNborderColor	Pixel	color	Core
XmNborderPixmap	Pixmap	pixmap	Core
XmNborderWidth	Dimension	integer	Core XmRectObj
XmNbottomAttachment	unsigned char	integer	XmForm
XmNbottomOffset	int	integer	XmForm
XmNbottomPosition	int	integer	XmForm
XmNbottomShadowColor	Pixel	color	XmGadget XmManager XmPrimitive
XmNbottomShadowPixmap	Pixmap	pixmap	XmGadget XmManager XmPrimitive
XmNbottomWidget	Widget	widget_ref	XmForm
XmNbrowseSelectionCallback	XtCallbackList	procedure	XmList
XmNbuttonAcceleratorText	XmStringTable	NA	XmRowColumn
XmNbuttonAccelerators	StringTable	NA	XmRowColumn
XmNbuttonCount	int	NA	XmRowColumn
XmNbuttonFontList	XmFontList	font_table	VendorShell XmBulletinBoard XmMenuShell

Resource Name	Motif Type	UIL Type	Defined in Class
XmNbuttonRenderTable	XmRenderTable	widget_ref	VendorShell XmBulletinBoard XmMenuShell
XmNbuttonMnemonicCharSets	XmStringCharSetTable	NA	XmRowColumn
XmNbuttonMnemonics	XmKeySymTable	NA	XmRowColumn
XmNbuttonSet	int	NA	XmRowColumn
XmNbuttonType	XmButtonTypeTable	NA	XmRowColumn
XmNbuttons	XmStringTable	NA	XmRowColumn
XmNcancelButton	Widget	widget_ref	XmBulletinBoard
XmNcancelCallback	XtCallbackList	procedure	XmMessageBox XmSelectionBox
XmNcancelLabelString	XmString	compound_string	XmMessageBox XmSelectionBox
XmNcascadePixmap	Pixmap	pixmap	XmCascadeButton
XmNcascadingCallback	XtCallbackList	procedure	XmCascadeButton
XmNchildHorizontalAlignment	unsigned char	integer	XmFrame
XmNchildHorizontalSpacing	Dimension	integer	XmFrame
XmNchildPlacement	unsigned char	integer	XmSelectionBox
XmNchildType	unsigned char	integer	XmFrame
XmNchildVerticalAlignment	unsigned char	integer	XmFrame
XmNchildren	WidgetList	NA	Composite
XmNclientData	XtPointer	NA	XmDragContext
XmNclipWindow	Widget	widget_ref	XmScrolledWindow
XmNcollapsedStatePixmap	Pixmap	pixmap	XmContainer
XmNcolorAllocationProc	XmAllocColorProc	NA	XmScreen
XmNcolorCalculationProc	XmScreenColorProc	NA	XmScreen
XmNcolormap	Colormap	identifier	Core
XmNcolumns	short	integer	XmComboBox XmSimpleSpinBox XmText XmTextField

Resource Name	Motif Type	UIL Type	Defined in Class
XmNcomboBoxType	unsigned char	integer	XmComboBox
XmNcommand	XmString	compound_string	XmCommand
XmNcommandChangedCallback	XtCallbackList	procedure	XmCommand
XmNcommandEnteredCallback	XtCallbackList	procedure	XmCommand
XmNcommandWindow	Widget	widget_ref	XmMainWindow
XmNcommandWindowLocation	unsigned char	integer	XmMainWindow
XmNconvertCallback	XtCallbackList	procedure	XmContainer XmDrawingArea XmPrimitive XmScale
XmNconvertProc	XtConvertSelectionIncrProc	NA	XmDragContext
XmNcreatePopupChildProc	XtCreatePopupChildProc	any	Shell
XmNcurrentPageNumber	integer	int	XmNotebook
XmNcursorBackground	Pixel	NA	XmDragContext
XmNcursorForeground	Pixel	NA	XmDragContext
XmNcursorPosition	XmTextPosition	integer	XmText XmTextField
XmNcursorPositionVisible	Boolean	boolean	XmText XmTextField
XmNdarkThreshold	int	NA	XmScreen
XmNdecimal	String	string	XmTab
XmNdecimalPoints	short	integer	XmScale XmSimpleSpinBox XmSpinBox
XmNdecrementCallback	XtCallbackList	procedure	XmScrollBar
XmNdefaultActionCallback	XtCallbackList	procedure	XmContainer XmList
XmNdefaultArrowSensitivity	unsigned char	integer	XmSpinBox
XmNdefaultButton	Widget	widget_ref	XmBulletinBoard
XmNdefaultButtonEmphasis	XtEnum	NA	XmDisplay
XmNdefaultButtonShadowThickness	Dimension	integer	XmPushButton

Resource Name	Motif Type	UIL Type	Defined in Class
XmNdefaultButtonType	unsigned char	integer	XmMessageBox
XmNdefaultCopyCursorIcon	Widget	NA	XmScreen
XmNdefaultFontList	XmFontList	font_table	VendorShell XmMenuShell
XmNdefaultInvalidCursorIcon	Widget	NA	XmScreen
XmNdefaultLinkCursorIcon	Widget	NA	XmScreen
XmNdefaultMoveCursorIcon	Widget	NA	XmScreen
XmNdefaultNoneCursorIcon	Widget	NA	XmScreen
XmNdefaultPosition	Boolean	boolean	XmBulletinBoard
XmNdefaultSourceCursorIcon	Widget	NA	XmScreen
XmNdefaultValidCursorIcon	Widget	NA	XmScreen
XmNdefaultVirtualBindings	String	NA	XmDisplay
XmNdeleteResponse	unsigned char	integer	VendorShell
XmNdepth	int	identifier	Core XmDragIcon
XmNdestinationCallback	XtCallbackList	procedure	XmContainer XmDrawingArea XmList XmText XmTextField
XmNdestroyCallback	XtCallbackList	procedure	Core XmObject
XmNdetail	XmStringTable	string_table	XmIconGadget
XmNdetailColumnHeading	XmStringTable	string_table	XmContainer
XmNdetailColumnHeadingCount	Cardinal	integer	XmContainer
XmNdetailCount	int	integer	XmIconGadget
XmNdetailOrder	Cardinal *	integer_table	XmContainer
XmNdetailOrderCount	Cardinal	integer	XmContainer
XmNdetailShadowThickness	int	integer	XmArrowButton XmSpinBox XmToggleButton
XmNdetailTabList	XmTabList	widget_ref	XmContainer

Resource Name	Motif Type	UIL Type	Defined in Class
XmNdialogStyle	unsigned char	integer	XmBulletinBoard
XmNdialogTitle	XmString	compound_string	XmBulletinBoard
XmNdialogType	unsigned char	integer	XmMessageBox XmSelectionBox
XmNdirListItemCount	int	integer	XmFileSelectionBox
XmNdirListItems	XmStringTable	string_table	XmFileSelectionBox
XmNdirListLabelString	XmString	compound_string	XmFileSelectionBox
XmNdirMask	XmString	compound_string	XmFileSelectionBox
XmNdirSearchProc	XmSearchProc	any	XmFileSelectionBox
XmNdirSpec	XmString	compound_string	XmFileSelectionBox
XmNdirTextLabelString	XmString	NA	XmFileSelectionBox
XmNdirectory	XmString	compound_string	XmFileSelectionBox
XmNdirectoryValid	Boolean	NA	XmFileSelectionBox
XmNdisarmCallback	XtCallbackList	procedure	XmArrowButton XmDrawnButton XmPushButton XmToggleButton
XmNdoubleClickInterval	int	integer	XmList
XmNdragCallback	XtCallbackList	procedure	XmScale XmScrollBar
XmNdragDropFinishCallback	XtCallbackList	NA	XmDragContext
XmNdragInitiatorProtocolStyle	unsigned char	NA	XmDisplay
XmNdragMotionCallback	XtCallbackList	NA	XmDragContext
XmNdragOperations	unsigned char	NA	XmDragContext
XmNdragProc	XtCallbackProc	NA	XmDropSite
XmNdragReceiverProtocolStyle	unsigned char	NA	XmDisplay
XmNdragStartCallback	XtCallbackList	NA	XmDisplay
XmNdropFinishCallback	XtCallbackList	NA	XmDragContext
XmNdropProc	XtCallbackProc	NA	XmDropSite
XmNdropRectangles	XRectangle *	NA	XmDropSite

Resource Name	Motif Type	UIL Type	Defined in Class
XmNdropSiteActivity	unsigned char	NA	XmDropSite
XmNdropSiteEnterCallback	XtCallbackList	NA	XmDragContext
XmNdropSiteLeaveCallback	XtCallbackList	NA	XmDragContext
XmNdropSiteOperations	unsigned char	NA	XmDropSite
XmNdropSiteType	unsigned char	NA	XmDropSite
XmNdropStartCallback	XtCallbackList	NA	XmDragContext
XmNdropTransfers	XmDropTransferEntryRec *}	NA	XmDropTransfer
XmNeditMode	int	integer	XmText
XmNeditable	Boolean	boolean	XmScale XmScrollBar XmSimpleSpinBox XmSpinBox XmText XmTextField
XmNenableBtn1Transfer	XtEnum	NA	XmDisplay
XmNenableButtonTab	Boolean	NA	XmDisplay
XmNenableEtchedInMenu	Boolean	NA	XmDisplay
XmNenableMultiKeyBindings	Boolean	NA	XmDisplay
XmNenableThinThickness	Boolean	NA	XmDisplay
XmNenableToggleColor	Boolean	NA	XmDisplay
XmNenableToggleVisual	Boolean	NA	XmDisplay
XmNenableUnselectableDrag	Boolean	NA	XmDisplay
XmNenableWarp	XtEnum	NA	XmDisplay
XmNentryAlignment	unsigned char	integer	XmRowColumn
XmNentryBorder	Dimension	integer	XmRowColumn
XmNentryCallback	XtCallbackList	procedure	XmRowColumn
XmNentryClass	WidgetClass	class_rec_name	XmRowColumn
XmNentryParent	Widget	widget_ref	XmContainer
XmNentryVerticalAlignment	unsigned char	integer	XmRowColumn
XmNentryViewType	unsigned char	integer	XmContainer

Resource Name	Motif Type	UIL Type	Defined in Class
XmNexpandedStatePixmap	Pixmap	pixmap	XmContainer
XmNexportTargets	Atom *	NA	XmDragContext
XmNexposeCallback	XtCallbackList	procedure	XmDrawingArea XmDrawnButton
XmNextendedSelectionCallback	XtCallbackList	procedure	XmList
XmNfileFilterStyle	XtEnum	NA	XmFileSelectionBox
XmNfileListItemCount	int	integer	XmFileSelectionBox
XmNfileListItems	XmStringTable	string_table	XmFileSelectionBox
XmNfileListLabelString	XmString	compound_string	XmFileSelectionBox
XmNfileSearchProc	XmSearchProc	any	XmFileSelectionBox
XmNfileTypeMask	unsigned char	integer	XmFileSelectionBox
XmNfillOnArm	Boolean	boolean	XmPushButton
XmNfillOnSelect	Boolean	boolean	XmToggleButton
XmNfilterLabelString	XmString	compound_string	XmFileSelectionBox
XmNfirstPageNumber	int	integer	XmNotebook
XmNfocusCallback	XtCallbackList	procedure	XmBulletinBoard XmText XmTextField
XmNfont	XFontStruct *	NA	XmScreen
XmNfont	XFontStruct *	font	XmRendition
XmNfontList	XmFontList	font_table	XmComboBox XmContainer XmIconGadget XmLabel XmList XmScale XmText XmTextField
XmNfontName	String	string	XmRendition
XmNfontType	XmFontType	integer	XmRendition
XmNforeground	Pixel	color	XmManager XmPrimitive
XmNforegroundThreshold	int	NA	XmScreen

Resource Name	Motif Type	UIL Type	Defined in Class
XmNfractionBase	int	integer	XmForm
XmNframeBackground	Pixel	color	XmNotebook
XmNframeChildType	unsigned char	integer	XmFrame
XmNframeShadowThickness	Dimension	integer	XmNotebook
XmNgainPrimaryCallback	XtCallbackList	procedure	XmText XmTextField
XmNgeometry	String	string	Shell
XmNheight	Dimension	integer	Core XmDragIcon XmRectObj
XmNheightInc	int	integer	WMShell
XmNhelpCallback	XtCallbackList	procedure	XmGadget XmManager XmPrimitive
XmNhelpLabelString	XmString	compound_string	XmMessageBox XmSelectionBox
XmNhighlightColor	Pixel	color	XmGadget XmManager XmPrimitive
XmNhighlightOnEnter	Boolean	boolean	XmGadget XmPrimitive XmScale
XmNhighlightPixmap	Pixmap	pixmap	XmGadget XmManager XmPrimitive
XmNhighlightThickness	Dimension	integer	XmComboBox XmGadget XmPrimitive XmScale
XmNhistoryItemCount	int	integer	XmCommand
XmNhistoryItems	XmStringTable	string_table	XmCommand
XmNhistoryMaxItems	int	integer	XmCommand
XmNhistoryVisibleItemCount	int	integer	XmCommand
XmNhorizontalFontUnit	int	NA	XmScreen

Resource Name	Motif Type	UIL Type	Defined in Class
XmNhorizontalScrollBar	Widget	widget_ref	XmList XmScrolledWindow
XmNhorizontalSpacing	Dimension	integer	XmForm
XmNhotX	Position	NA	XmDragIcon
XmNhotY	Position	NA	XmDragIcon
XmNiconMask	Pixmap	pixmap	WMShell
XmNiconName	String	NA	TopLevelShell
XmNiconNameEncoding	Atom	NA	TopLevelShell
XmNiconPixmap	Pixmap	pixmap	WMShell
XmNiconWindow	Window	any	WMShell
XmNiconX	int	integer	WMShell
XmNiconY	int	integer	WMShell
XmNiconic	Boolean	NA	TopLevelShell
XmNimportTargets	Atom *	NA	XmDropSite
XmNincrement	int	integer	XmScrollBar
XmNincrementCallback	XtCallbackList	procedure	XmScrollBar
XmNincrementValue	int	integer	XmSimpleSpinBox XmSpinBox
XmNincremental	Boolean	NA	XmDragContext XmDropTransfer
XmNindeterminateInsensitivePixmap	Pixmap	pixmap	XmToggleButton
XmNindeterminatePixmap	Pixmap	pixmap	XmToggleButton
XmNindicatorOn	Boolean	boolean	XmToggleButton
XmNindicatorSize	Dimension	integer	XmToggleButton
XmNindicatorType	unsigned char	integer	XmToggleButton
XmNinitialDelay	int	integer	XmScrollBar XmSpinBox
XmNinitialFocus	Widget	widget_ref	XmManager
XmNinitialResourcesPersistent	Boolean	boolean	Core
XmNinitialState	int	integer	WMShell

Resource Name	Motif Type	UIL Type	Defined in Class
XmNinnerMarginHeight	Dimension	integer	XmNotebook
XmNinnerMarginWidth	Dimension	integer	XmNotebook
XmNinput	Boolean	boolean	WMShell
XmNinputCallback	XtCallbackList	procedure	XmDrawingArea
XmNinputMethod	String	string	VendorShell
XmNinputPolicy	XmInputPolicy	integer	VendorShell
XmNinsensitiveStipplePixmap	Pixmap	NA	XmScreen
XmNinsertPosition	XtOrderProc	identifier	Composite
XmNinvalidCursorForeground	Pixel	NA	XmDragContext
XmNisAligned	Boolean	boolean	XmRowColumn
XmNisHomogeneous	Boolean	boolean	XmRowColumn
XmNitemCount	int	integer	XmComboBox XmList
XmNitems	XmStringTable	string_table	XmComboBox XmList
XmNkeyboardFocusPolicy	unsigned char	integer	VendorShell
XmNlabelFontList	XmFontList	font_table	XmBulletinBoard VendorShell XmMenuShell
XmNlabelInsensitivePixmap	Pixmap	pixmap	XmLabel
XmNlabelPixmap	Pixmap	pixmap	XmLabel
XmNlabelRenderTable	XmRenderTable	widget_ref	VendorShell XmBulletinBoard XmMenuShell
XmNlabelString	XmString	compound_string	XmIconGadget XmLabel XmRowColumn
XmNlabelType	unsigned char	integer	XmLabel
XmNlargeCellHeight	Dimension	integer	XmContainer
XmNlargeCellWidth	Dimension	integer	XmContainer
XmNlargeIconMask	Pixmap	pixmap	XmIconGadget

Appendix C: Table of Motif Resources

Resource Name	Motif Type	UIL Type	Defined in Class
XmNlargeIconPixmap	Pixmap	pixmap	XmIconGadget
XmNlargeIconX	NA	float	XmContainer
XmNlargeIconY	NA	float	XmContainer
XmNlargeIconY	Pixmap	pixmap	XmIconGadget
XmNlastPageNumber	int	integer	XmNotebook
XmNlayoutDirection	XmDirection	integer	VendorShell XmGadget XmManager XmMenuShell XmPrimitive
XmNlayoutType	unsigned char	integer	XmContainer
XmNleftAttachment	unsigned char	integer	XmForm
XmNleftOffset	int	integer	XmForm
XmNleftPosition	int	integer	XmForm
XmNleftWidget	Widget	widget_ref	XmForm
XmNlightThreshold	int	NA	XmScreen
XmNlist	Widget	widget_ref	XmComboBox
XmNlistItemCount	int	integer	XmSelectionBox
XmNlistItems	XmStringTable	string_table	XmSelectionBox
XmNlistLabelString	XmString	compound_string	XmSelectionBox
XmNlistMarginHeight	Dimension	integer	XmList
XmNlistMarginWidth	Dimension	integer	XmList
XmNlistSizePolicy	unsigned char	integer	XmList
XmNlistSpacing	Dimension	integer	XmList
XmNlistUpdated	Boolean	boolean	XmFileSelectionBox
XmNlistVisibleItemCount	int	integer	XmSelectionBox
XmNloadModel	unsigned char	integer	XmRendition
XmNlosePrimaryCallback	XtCallbackList	procedure	XmText XmTextField
XmNlosingFocusCallback	XtCallbackList	procedure	XmText XmTextField

Resource Name	Motif Type	UIL Type	Defined in Class
XmNmainWindowMarginHeight	Dimension	integer	XmMainWindow
XmNmainWindowMarginWidth	Dimension	integer	XmMainWindow
XmNmajorTabSpacing	Dimension	integer	XmNotebook
XmNmapCallback	XtCallbackList	procedure	XmBulletinBoard XmRowColumn
XmNmappedWhenManaged	Boolean	boolean	Core
XmNmappingDelay	int	integer	XmCascadeButton
XmNmargin	Dimension	integer	XmSeparator
XmNmarginBottom	Dimension	integer	XmLabel
XmNmarginHeight	Dimension	integer	XmBulletinBoard XmComboBox XmContainer XmDrawingArea XmFrame XmIconGadget XmLabel XmPanedWindow XmRowColumn XmSpinBox XmText XmTextField
XmNmarginLeft	Dimension	integer	XmLabel
XmNmarginRight	Dimension	integer	XmLabel
XmNmarginTop	Dimension	integer	XmLabel
XmNmarginWidth	Dimension	integer	XmBulletinBoard XmComboBox XmContainer XmDrawingArea XmFrame XmIconGadget XmLabel XmPanedWindow XmRowColumn XmSpinBox XmText XmTextField
XmNmask	Pixmap	NA	XmDragIcon

Resource Name	Motif Type	UIL Type	Defined in Class
XmNmatchBehavior	unsigned char	integer	XmComboBox XmList
XmNmaxAspectX	int	integer	WMShell
XmNmaxAspectY	int	integer	WMShell
XmNmaxHeight	int	integer	WMShell
XmNmaxLength	int	integer	XmText XmTextField
XmNmaxWidth	int	integer	WMShell
XmNmaximum	int	integer	XmScale XmScrollBar
XmNmaximumValue	int	integer	XmSimpleSpinBox XmSpinBox
XmNmenuAccelerator	String	string	XmRowColumn
XmNmenuBar	Widget	widget_ref	XmMainWindow
XmNmenuCursor	String	NA	XmScreen
XmNmenuHelpWidget	Widget	widget_ref	XmRowColumn
XmNmenuHistory	Widget	widget_ref	XmRowColumn
XmNmenuPost	String	string	XmRowColumn
XmNmessageAlignment	unsigned char	integer	XmMessageBox
XmNmessageString	XmString	compound_string	XmMessageBox
XmNmessageWindow	Widget	widget_ref	XmMainWindow
XmNminAspectX	int	integer	WMShell
XmNminAspectY	int	integer	WMShell
XmNminHeight	int	integer	WMShell
XmNminWidth	int	integer	WMShell
XmNminimizeButtons	Boolean	boolean	XmMessageBox XmSelectionBox
XmNminimum	int	integer	XmScale XmScrollBar
XmNminimumValue	int	integer	XmSimpleSpinBox XmSpinBox

Resource Name	Motif Type	UIL Type	Defined in Class
XmNminorTabSpacing	Dimension	integer	XmNotebook
XmNmnemonic	KeySym	keysym	XmLabel XmRowColumn
XmNmnemonicCharSet	String	string	XmLabel XmRowColumn
XmNmodifyVerifyCallback	XtCallbackList	procedure	XmText XmTextField
XmNmodifyVerifyCallbackWcs	XtCallbackList	procedure	XmText XmTextField
XmNmotifVersion	int	NA	XmDisplay
XmNmotionVerifyCallback	XtCallbackList	procedure	XmText XmTextField
XmNmoveOpaque	Boolean	NA	XmScreen
XmNmultiClick	unsigned char	integer	XmArrowButton XmDrawnButton XmPushButton
XmNmultipleSelectionCallback	XtCallbackList	procedure	XmList
XmNmustMatch	Boolean	boolean	XmSelectionBox
XmNmwmDecorations	int	integer	VendorShell
XmNmwmFunctions	int	integer	VendorShell
XmNmwmInputMode	int	integer	VendorShell
XmNmwmMenu	String	string	VendorShell
XmNnavigationType	XmNavigationType	integer	XmGadget XmManager XmPrimitive
XmNnoFontCallback	XtCallbackList	NA	XmDisplay
XmNnoMatchCallback	XtCallbackList	procedure	XmSelectionBox
XmNnoMatchString	XmString	compound_string	XmFileSelectionBox
XmNnoRenditionCallback	XtCallbackList	NA	XmDisplay
XmNnoResize	Boolean	boolean	XmBulletinBoard
XmNnoneCursorForeground	Pixel	NA	XmDragContext
XmNnotebookChildType	unsigned char	integer	XmNotebook

Resource Name	Motif Type	UIL Type	Defined in Class
XmNnumChildren	Cardinal	NA	Composite
XmNnumColumns	short	integer	XmRowColumn
XmNnumDropRectangles	Cardinal	NA	XmDropSite
XmNnumDropTransfers	Cardinal	NA	XmDropTransfer
XmNnumExportTargets	Cardinal	NA	XmDragContext
XmNnumImportTargets	Cardinal	NA	XmDropSite
XmNnumValues	int	integer	XmSimpleSpinBox XmSpinBox
XmNoffsetModel	XmOffsetModel	integer	XmTab
XmNoffsetX	Position	NA	XmDragIcon
XmNoffsetY	Position	NA	XmDragIcon
XmNokCallback	XtCallbackList	procedure	XmMessageBox XmSelectionBox
XmNokLabelString	XmString	compound_string	XmMessageBox XmSelectionBox
XmNoperationChangedCallback	XtCallbackList	NA	XmDragContext
XmNoperationCursorIcon	Widget	NA	XmDragContext
XmNoptionLabel	XmString	NA	XmRowColumn
XmNoptionMnemonic	KeySym	NA	XmRowColumn
XmNorientation	unsigned char	integer	XmNotebook XmPanedWindow XmRowColumn XmScale XmScrollBar XmSeparator
XmNoutlineButtonPolicy	unsigned char	integer	XmContainer
XmNoutlineChangedCallback	XtCallbackList	procedure	XmContainer
XmNoutlineColumnWidth	Dimension	integer	XmContainer
XmNoutlineIndentation	Dimension	integer	XmContainer
XmNoutlineLineStyle	unsigned char	integer	XmContainer
XmNoutlineState	unsigned char	integer	XmContainer

Resource Name	Motif Type	UIL Type	Defined in Class
XmNoverrideRedirect	Boolean	boolean	Shell
XmNpacking	unsigned char	integer	XmRowColumn
XmNPageChangedCallback	XtCallbackList	procedure	XmNotebook
XmNpageDecrementCallback	XtCallbackList	procedure	XmScrollBar
XmNpageIncrement	int	integer	XmScrollBar
XmNpageIncrementCallback	XtCallbackList	procedure	XmScrollBar
XmNpageNumber	int	integer	XmNotebook
XmNpaneMaximum	Dimension	integer	XmPanedWindow
XmNpaneMinimum	Dimension	integer	XmPanedWindow
XmNpathMode	XtEnum	NA	XmFileSelectionBox
XmNpattern	XmString	compound_string	XmFileSelectionBox
XmNpendingDelete	Boolean	boolean	XmTextField
XmNpixmap	Pixmap	NA	XmDragIcon
XmNpopdownCallback	XtCallbackList	procedure	Shell
XmNpopupCallback	XtCallbackList	procedure	Shell
XmNpopupEnabled	Boolean	boolean	XmRowColumn
XmNpopupHandlerCallback	XtCallbackList	procedure	XmManager XmPrimitive
XmNposition	int	integer	XmSimpleSpinBox XmSpinBox
XmNpositionIndex	short	integer	XmContainer XmNotebook XmPanedWindow XmRowColumn
XmNpositionMode	XtEnum	integer	XmComboBox
XmNpositionType	unsigned char	integer	XmSimpleSpinBox XmSpinBox
XmNpostFromButton	int	NA	XmRowColumn
XmNpreeditType	String	string	VendorShell
XmNprimaryOwnership	unsigned char	integer	XmContainer XmList

Resource Name	Motif Type	UIL Type	Defined in Class
XmNprocessingDirection	unsigned char	integer	XmScale XmScrollBar
XmNpromptString	XmString	compound_string	XmCommand
XmNpushButtonEnabled	Boolean	boolean	XmDrawnButton
XmNqualifySearchDataProc	XmQualifyProc	any	XmFileSelectionBox
XmNradioAlwaysOne	Boolean	boolean	XmRowColumn
XmNradioBehavior	Boolean	boolean	XmRowColumn
XmNrecomputeSize	Boolean	boolean	XmLabel
XmNrefigureMode	Boolean	boolean	XmPanedWindow
XmNrenderTable	XmRenderTable	widget_ref	XmComboBox XmContainer XmIconGadget XmLabel XmList XmScale XmText XmTextField
XmNrenditionBackground	Pixel	color	XmRendition
XmNrenditionForeground	Pixel	color	XmRendition
XmNrepeatDelay	int	integer	XmScrollBar XmSpinBox
XmNresizable	Boolean	boolean	XmForm XmNotebook
XmNresizeCallback	XtCallbackList	procedure	XmDrawingArea XmDrawnButton
XmNresizeHeight	Boolean	boolean	XmRowColumn XmText
XmNresizePolicy	unsigned char	integer	XmBulletinBoard XmDrawingArea
XmNresizeWidth	Boolean	boolean	XmRowColumn XmText XmTextField
XmNrightAttachment	unsigned char	integer	XmForm
XmNrightOffset	int	integer	XmForm

Resource Name	Motif Type	UIL Type	Defined in Class
XmNrightPosition	int	integer	XmForm
XmNrightWidget	Widget	widget_ref	XmForm
XmNrowColumnType	unsigned char	integer	XmRowColumn
XmNrows	short	integer	XmText
XmNrubberPositioning	Boolean	boolean	XmForm
XmNsashHeight	Dimension	integer	XmPanedWindow
XmNsashIndent	Position	integer	XmPanedWindow
XmNsashShadowThickness	Dimension	integer	XmPanedWindow
XmNsashWidth	Dimension	integer	XmPanedWindow
XmNsaveUnder	Boolean	boolean	Shell
XmNscaleHeight	Dimension	integer	XmScale
XmNscaleMultiple	int	integer	XmScale
XmNscaleWidth	Dimension	integer	XmScale
XmNscreen	Screen *	identifier	Core
XmNscrollBarDisplayPolicy	unsigned char	integer	XmList XmScrolledWindow
XmNscrollBarPlacement	unsigned char	integer	XmScrolledWindow
XmNscrollHorizontal	Boolean	boolean	XmText
XmNscrollLeftSide	Boolean	boolean	XmText
XmNscrollTopSide	Boolean	boolean	XmText
XmNscrollVertical	Boolean	boolean	XmText
XmNscrolledWindowChildType	Widget	widget_ref	XmScrolledWindow
XmNscrolledWindowMarginHeight	Dimension	integer	XmScrolledWindow
XmNscrolledWindowMarginWidth	Dimension	integer	XmScrolledWindow
XmNscrollingPolicy	unsigned char	integer	XmScrolledWindow
XmNselectColor	Pixel	color	XmContainer XmList XmToggleButton
XmNselectInsensitivePixmap	Pixmap	pixmap	XmToggleButton

Resource Name	Motif Type	UIL Type	Defined in Class
XmNselectPixmap	Pixmap	pixmap	XmToggleButton
XmNselectThreshold	int	integer	XmTextField
XmNselectedItem	XmString	compound_string	XmComboBox
XmNselectedItemCount	int	integer	XmList
XmNselectedItems	XmStringTable	string_table	XmList
XmNselectedObjectCount	int	NA	XmContainer
XmNselectedObjects	WidgetList	NA	XmContainer
XmNselectedPosition	int	integer	XmComboBox
XmNselectedPositionCount	int	integer	XmList
XmNselectedPositions	int *	integer_table	XmList
XmNselectionArray	XtPointer	any	XmTextField
XmNselectionArrayCount	int	integer	XmTextField
XmNselectionCallback	XtCallbackList	procedure	XmComboBox XmContainer
XmNselectionLabelString	XmString	compound_string	XmSelectionBox
XmNselectionMode	unsigned char	integer	XmList
XmNselectionPolicy	unsigned char	integer	XmContainer XmList
XmNselectionTechnique	unsigned char	integer	XmContainer
XmNsensitive	Boolean	boolean	Core XmRectObj
XmNseparatorOn	Boolean	boolean	XmPanedWindow
XmNseparatorType	unsigned char	integer	XmSeparator
XmNset	unsigned char	integer	XmToggleButton
XmNshadowThickness	Dimension	integer	XmGadget XmManager XmPrimitive
XmNshadowType	unsigned char	integer	XmBulletinBoard XmDrawnButton XmFrame
XmNshellUnitType	unsigned char	integer	VendorShell

Resource Name	Motif Type	UIL Type	Defined in Class
XmNshowArrows	Boolean	boolean	XmScale XmScrollBar
XmNshowAsDefault	Dimension	integer	XmPushButton
XmNshowSeparator	Boolean	boolean	XmMainWindow
XmNshowValue	Boolean	boolean	XmScale
XmNsimpleCallback	XtCallbackProc	procedure	XmRowColumn
XmNsingleSelectionCallback	XtCallbackList	procedure	XmList
XmNskipAdjust	Boolean	boolean	XmPanedWindow
XmNsliderMark	XtEnum	integer	XmScale XmScrollBar
XmNsliderSize	int	integer	XmScrollBar
XmNsliderVisual	Visual *	integer	XmScale XmScrollBar
XmNslidingMode	XtEnum	integer	XmScale XmScrollBar
XmNsmallCellHeight	Dimension	integer	XmContainer
XmNsmallCellWidth	Dimension	integer	XmContainer
XmNsmallIconMask	Pixmap	pixmap	XmIconGadget
XmNsmallIconPixmap	Pixmap	pixmap	XmIconGadget
XmNsmallIconX	NA	float	XmContainer
XmNsmallIconY	NA	float	XmContainer
XmNsnapBackMultiple	unsigned short	integer	XmScrollBar
XmNsource	XmTextSource	any	XmText
XmNsourceCursorIcon	Widget	NA	XmDragContext
XmNsourcePixmapIcon	Widget	NA	XmDragContext
XmNspacing	Dimension	integer	XmIconGadget XmPanedWindow XmRowColumn XmScrolledWindow XmSpinBox XmToggleButton
XmNspatialIncludeModel	unsigned char	integer	XmContainer

Appendix C: Table of Motif Resources

Resource Name	Motif Type	UIL Type	Defined in Class
XmNspatialResizeModel	unsigned char	integer	XmContainer
XmNspatialSnapModel	unsigned char	integer	XmContainer
XmNspatialStyle	unsigned char	integer	XmContainer
XmNspinBoxChildType	unsigned char	integer	XmSimpleSpinBox XmSpinBox
XmNstateCursorIcon	Widget	NA	XmDragContext
XmNstrikethruType	unsigned char	integer	XmRendition
XmNstringDirection	XmStringDirection	integer	XmLabel XmList XmManager
XmNsubMenuId	Widget	widget_ref	XmCascadeButton XmRowColumn
XmNsymbolPixmap	Pixmap	pixmap	XmMessageBox
XmNtabList	XmTabList	widget_ref	XmRendition
XmNtabValue	NA	float	XmTab
XmNtag	XmStringTag	string	XmRendition
XmNtearOffMenuActivateCallback	XtCallbackList	procedure	XmRowColumn
XmNtearOffMenuDeactivateCallback	XtCallbackList	procedure	XmRowColumn
XmNtearOffModel	unsigned char	integer	XmRowColumn
XmNtearOffTitle	XmString	compound_string	XmRowColumn
XmNtextAccelerators	XtAccelerators	translation_table	XmSelectionBox
XmNtextColumns	short	integer	XmSelectionBox
XmNtextField	Widget	widget_ref	XmComboBox XmSimpleSpinBox XmSpinBox
XmNtextFontList	XmFontList	font_table	VendorShell XmBulletinBoard
XmNtextRenderTable	XmRenderTable	widget_ref	VendorShell XmBulletinBoard
XmNtextString	XmString	compound_string	XmSelectionBox
XmNtextTranslations	XtTranslations	translation_table	XmBulletinBoard

Resource Name	Motif Type	UIL Type	Defined in Class
XmNtitle	String	string	WMShell
XmNtitleEncoding	Atom	any	WMShell
XmNtitleString	XmString	compound_string	XmScale
XmNtoBottomCallback	XtCallbackList	procedure	XmScrollBar
XmNtoTopCallback	XtCallbackList	procedure	XmScrollBar
XmNtoggleMode	unsigned char	integer	XmToggleButton
XmNtopAttachment	unsigned char	integer	XmForm
XmNtopCharacter	XmTextPosition	integer	XmText
XmNtopItemPosition	int	integer	XmList
XmNtopLevelEnterCallback	XtCallbackList	NA	XmDragContext
XmNtopLevelLeaveCallback	XtCallbackList	NA	XmDragContext
XmNtopOffset	int	integer	XmForm
XmNtopPosition	int	integer	XmForm
XmNtopShadowColor	Pixel	color	XmGadget XmManager XmPrimitive
XmNtopShadowPixmap	Pixmap	pixmap	XmGadget XmManager XmPrimitive
XmNtopWidget	Widget	widget_ref	XmForm
XmNtransferProc	XtSelectionCallbackProc	NA	XmDropTransfer
XmNtransferStatus	unsigned char	NA	XmDropTransfer
XmNtransient	Boolean	boolean	WMShell
XmNtransientFor	Widget	widget_ref	TransientShell
XmNtranslations	XtTranslations	translation_table	Core
XmNtraversalOn	Boolean	boolean	XmGadget XmManager XmPrimitive
XmNtraverseObscuredCallback	XtCallbackList	procedure	XmScrolledWindow
XmNtroughColor	Pixel	color	XmScrollBar
XmNunderlineType	unsigned char	integer	XmRendition

Appendix C: Table of Motif Resources

Resource Name	Motif Type	UIL Type	Defined in Class
XmNunitType	unsigned char	integer	VendorShell XmGadget XmManager XmPrimitive XmTab
XmNunmapCallback	XtCallbackList	procedure	XmBulletinBoard XmRowColumn
XmNunpostBehavior	unsigned char	integer	XmScreen
XmNunselectColor	Pixel	color	XmToggleButton
XmNuseAsyncGeometry	Boolean	boolean	VendorShell
XmNuseColorObject	Boolean	NA	XmScreen
XmNuserData	XtPointer	any	XmDisplay XmGadget XmManager XmPrimitive XmScreen
XmNvalidCursorForeground	Pixel	NA	XmDragContext
XmNvalue	String		XmText XmTextField
XmNvalue	int	string	XmScale XmScrollBar
XmNvalues	XmStringTable	string_table	XmSimpleSpinBox XmSpinBox
XmNvalueChangedCallback	XtCallbackList	procedure	XmScale XmScrollBar XmText XmTextField XmToggleButton
XmNvalueWcs	wchar_t *	wide_character	XmText XmTextField
XmNverifyBell	Boolean	boolean	XmText XmTextField
XmNverticalFontUnit	int	NA	XmScreen
XmNverticalScrollBar	Widget	widget_ref	XmList XmScrolledWindow
XmNverticalSpacing	Dimension	integer	XmForm

Resource Name	Motif Type	UIL Type	Defined in Class
XmNviewType	unsigned char	integer	XmIconGadget
XmNvisibleItemCount	int	integer	XmComboBox XmList
XmNvisibleWhenOff	Boolean	boolean	XmToggleButton
XmNvisual	Visual *	any	Shell
XmNvisualEmphasis	unsigned char	integer	XmIconGadget
XmNvisualPolicy	unsigned char	integer	XmScrolledWindow
XmNwaitForWm	Boolean	boolean	WMShell
XmNwhichButton	unsigned int	integer	XmRowColumn
XmNwidth	Dimension	integer	Core XmDragIcon XmRectObj
XmNwidthInc	int	integer	WMShell
XmNwinGravity	int	integer	WMShell
XmNwindowGroup	Window	any	WMShell
XmNwmTimeout	int	integer	WMShell
XmNwordWrap	Boolean	boolean	XmText
XmNworkWindow	Widget	widget_ref	XmScrolledWindow
XmNwrap	Boolean	boolean	XmSimpleSpinBox XmSpinBox
XmNx	Position	integer	Core XmRectObj
XmNy	Position	integer	Core XmRectObj

Appendix D Table of UIL Objects

This appendix lists all of the objects supported by the User Interface Language (UIL). For each object, the table lists the corresponding widget or widgets in the Motif toolkit. The resources and callbacks for each object are the same as the resources and callbacks for the corresponding widget(s). UIL provides one additional callback, MrmNcreateCallback, for each object. This callback is invoked when the object is instantiated by the Motif Resource Manager (Mrm). The table also specifies the types of objects that can be children of a particular object, as well as the names and classes of any automatically-created children. For more information on each object, see the appropriate reference pages in Section 2, *Motif and Xt Widget Classes*.

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmArrowButton	XmArrowButton	XmPopupMenu		
XmArrowButton-Gadget	XmArrowButton-Gadget	none		
XmBulletinBoard	XmBulletinBoard	all UIL objects		
XmBulletinBoardDialog	XmDialogShell with XmBulletinBoard child	all UIL objects		
XmCascadeButton	XmCascadeButton	XmPopupMenu XmPulldownMenu		
XmCascadeButton-Gadget	XmCascadeButton-Gadget	XmPulldownMenu		
XmCheckBox	XmRowColumn	all UIL objects		
XmComboBox	XmComboBox	none	Xm_List Xm_Text Xm_TextField	Xm_List Xm_Text Xm_TextField
XmCommand	XmCommand	XmPopupMenu		
XmContainer	XmContainer	all UIL objects[a]		

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmDialogShell	XmDialogShell	XmBulletinBoard XmCheckBox XmComboBox XmContainer XmDrawingArea XmFileSelection-Box XmForm XmFrame XmMessageBox XmNotebook XmPanedWindow XmRadioBox XmRenderTable XmRowColumn XmScale XmScrolledWin-dow XmSelectionBox XmSimpleSpinBox XmSpinBox XmWorkArea		
XmDrawingArea	XmDrawingArea	all UIL objects		
XmDrawnButton	XmDrawnButton	XmPopupMenu		
XmErrorDialog	XmDialogShell with XmMessageBox child	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton
XmFileSelectionBox	XmFileSelection-Box	all UIL objects	Xm_Items Xm_ItemsList Xm_Separator Xm_OK Xm_Cancel Xm_Help Xm_FilterLabe l Xm_FilterText Xm_DirList Xm_Dir Xm_Filter	XmLabel XmScrolledList XmSeparator XmPushButton XmPushButton XmPushButton XmLabel XmText XmScrolledList XmLabel XmPushButton

Appendix D: Table of UIL Objects

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmFileSelectionDialog	XmDialogShell with XmFileSelectionBox child	all UIL objects	Xm_Items Xm_ItemsList Xm_Separator Xm_OK Xm_Cancel Xm_Help Xm_FilterLabel Xm_FilterText Xm_DirList Xm_Dir Xm_Filter	XmLabel XmScrolledList XmSeparator XmPushButton XmPushButton XmPushButton XmLabel XmText XmScrolledList XmLabel XmPushButton
XmForm	XmForm	all UIL objects		
XmFormDialog	XmDialogShell with XmForm child	all UIL objects		
XmFrame	XmFrame	all UIL objects		
XmIconGadget	XmIconGadget	XmRenderTable		
XmInformationDialog	XmDialogShell with XmMessageBox child	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton
XmLabel	XmLabel	XmPopupMenu XmRenderTable		
XmLabelGadget	XmLabelGadget	XmRenderTable		
XmList	XmList	XmPopupMenu XmRenderTable		
XmMainWindow	XmMainWindow	all UIL objects	Xm_Separator1 Xm_Separator2 Xm_Separator3	XmSeparator XmSeparator XmSeparator

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmMenuBar	XmRowColumn	XmCascadeButton XmCascadeButton-Gadget XmDrawnButton XmLabel XmLabelGadget XmPopupMenu XmPulldownMenu XmPushButton XmPushButton-Gadget XmSeparator XmSeparatorGadg-et XmToggleButton XmToggleButton-Gadget user_defined object		
XmMenuShell	XmMenuShell	XmRenderTable XmRowColumn		
XmMessageBox	XmMessageBox	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton
XmMessageDialog	XmDialogShell with XmMessageBox child	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton
XmNotebook	XmNotebook	all UIL objects	[b]	
XmOptionMenu	XmRowColumn	XmPulldownMenu	Xm_OptionLab-el Xm_OptionBut-ton	XmLabelGadget XmCascadeButton-Gadget
XmPanedWindow	XmPanedWindow	all UIL objects		

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmPopupMenu	XmDialogShell with XmRowColumn child	XmCascadeButton XmCascadeButton-Gadget XmDrawnButton XmLabel XmLabelGadget XmPushButton XmPushButton-Gadget XmSeparator XmSeparatorGadg-et XmToggleButton XmToggleButton-Gadget user_defined object		
Xm_TearOffControl	XmTearOffButton	none		
XmPromptDialog	XmDialogShell with XmSelectionBox child	all UIL objects	Xm_Items Xm_ItemsList Xm_Selection Xm_Text Xm_Separator Xm_OK Xm_Cancel Xm_Help Xm_Apply	XmLabel XmScrolledList XmLabel XmText XmSeparator XmPushButton XmPushButton XmPushButton XmPushButton
XmPulldownMenu	XmDialogShell with XmRowColumn child	XmCascadeButton XmCascadeButton-Gadget XmDrawnButton XmLabel XmLabelGadget XmPushButton XmPushButton-Gadget XmSeparator XmSeparatorGadg-et XmToggleButton XmToggleButton-Gadget user_defined object	Xm_TearOffCo ntrol	XmTearOffButton

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmPushButtonGadget	XmPushButton-Gadget	none		
XmQuestionDialog	XmDialogShell with XmMessageBox child	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton
XmRadioBox	XmRowColumn	all UIL objects		
XmRenderTable	(pseudo-object)	XmRendition		
XmRendition	(pseudo-object)	XmTabList		
XmRowColumn	XmRowColumn	all UIL objects		
XmScale	XmScale	all UIL objects	Xm_Title	XmLabel
XmScrollBar	XmScrollBar	XmPopupMenu		
XmScrolledList	XmScrolledWin-dow with XmList child	XmPopupMenu XmRenderTable	Xm_VertScroll Bar Xm_HorScroll Bar	XmScrollBar XmScrollBar
XmScrolledText	XmScrolledWin-dow with XmText child	XmPopupMenu XmRenderTable	Xm_VertScroll Bar Xm_HorScroll Bar	XmScrollBar XmScrollBar
XmScrolledWindow	XmScrolledWin-dow	all UIL objects	Xm_VertScroll Bar Xm_HorScroll Bar	XmScrollBar XmScrollBar
XmSelectionBox	XmSelectionBox	all UIL objects	Xm_Items Xm_ItemsList Xm_Selection Xm_Text Xm_Separator Xm_OK Xm_Cancel Xm_Help Xm_Apply	XmLabel XmScrolledList XmLabel XmText XmSeparator XmPushButton XmPushButton XmPushButton XmPushButton

Appendix D: Table of UIL Objects

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmSelectionDialog	XmDialogShell with XmSelectionBox child	all UIL objects	Xm_Items Xm_ItemsList Xm_Selection Xm_Text Xm_Separator Xm_OK Xm_Cancel Xm_Help Xm_Apply	XmLabel XmScrolledList XmLabel XmText XmSeparator XmPushButton XmPushButton XmPushButton XmPushButton
XmSeparator	XmSeparator	XmPopupMenu		
XmSeparatorGadget	XmSeparatorGadget	none		
XmSimpleSpinBox	XmSimpleSpinBox	XmTextField		
XmSpinBox	XmSpinBox	all UIL objects		
XmTearOffButton	none	XmPopupMenu		
XmTemplateDialog	XmDialogShell with XmMessageBox child	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton
XmTab	(pseudo-object)	none		
XmTabList	(pseudo-object)	XmTab		
XmText	XmText	XmPopupMenu XmRenderTable		
XmTextField	XmTextField	XmPopupMenu XmRenderTable		
XmToggleButton	XmToggleButton	XmPopupMenu		
XmToggleButton-Gadget	XmToggleButton-Gadget	none		

UIL Object	Corresponding Motif Widget(s)	Allowable Children	Automatically Created Children	
			Name	Class
XmWarningDialog	XmDialogShell with XmMessageBox child	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton
XmWorkArea	XmRowColumn	all UIL objects		
XmWorkingDialog	XmDialogShell with XmMessageBox child	all UIL objects	Xm_Symbol Xm_Separator Xm_Message Xm_OK Xm_Cancel Xm_Help	XmLabel XmSeparator XmLabel XmPushButton XmPushButton XmPushButton

a.Surely a mistake. The Container is designed for IconGadget children only.
b.No access to the abstract ArrowButton and SpinBox controls.

Appendix E - New Features in Motif 2.0 and 2.1

This appendix provides a summary of the new features in Motif 2.1 and 2.0. It lists the new toolkit functions and widget classes in Motif 2.1 and 2.0, as well as any new resources added to existing widget classes. For more information on the functions and widgets, see the appropriate reference pages in Section 1, *Motif Functions and Macros*, and Section 2, *Motif and Xt Widget Classes*

The appendix also lists the new Motif Resource Manager (Mrm) functions and User Interface Language (UIL) data types in Motif 2.1. For more information on these functions and data types, see the reference pages in Section 3, *Mrm Functions*, and Section 6, *UIL Data Types.*

E.1 New Toolkit Functions

XmComboBoxAddItem()	Adds an item to a ComboBox.
XmComboBoxDeletePos()	Deletes an item from a ComboBox.
XmComboBoxSelectItem()	Selects an item in a ComboBox.
XmComboBoxSetItem()	Selects and makes visible an item in a ComboBox.
XmComboBoxUpdate()	Update the state of a ComboBox.
XmContainerCopy()	Copy selected Container items to the clipboard.
XmContainerCopyLink(	Copy links to selected Container items to the clipboard.
XmContainerCut()	Cut selected Container items to the clipboard.
XmContainerGetItemChildren()	Retrieve the logical children of a Container item.
XmContainerPaste()	Copy clipboard data into a Container.
XmContainerPasteLink()	Copy links to clipboard data into a Container.
XmContainerRelayout()	Force the relayout of Container items.
XmContainerReorder()	Sort the items within a Container.
XmConvertStringToUnits()	Convert a unit specification expressed as a string to an integral value.
XmCvtByteStreamToXmString()	Convert a byte stream representation to a compound string.
XmCvtTextPropertyToXmStringTable()	Convert a text property to a compound string table.
XmCvtXmStringTableToTextProperty()	Convert a compound string table to a text property.
XmCvtXmStringToByteStream()	Convert a compound string to a byte stream representation.

XmDirectionMatch()	Compare two XmDirection quantities.
XmDirectionMatchPartial()	Loosely compare two XmDirection quantities.
XmDirectionToStringDirection()	Convert an XmDirection type to a XmStringDirection type.
XmFontListCreate_r()	Create an XmFontList in a thread-safe manner.
XmFontListEntryCreate_r()	Create an XmFontListEntry in a thread-safe manner.
XmGetScaledPixmap()	Read a pixmap and scale it for printing.
XmImCloseXIM()	Close all input contexts associated with an X Input Method.
XmImFreeXIC()	Unregister all widgets associated with an X Input Context.
XmImGetXIC()	Create an X Input Context for a widget.
XmImMbResetIC()	Reset an Input Context.
XmImSetXIC()	Register a widget with an existing X Input Context.
XmNotebookGetPageInfo()	Retrieve data about a Notebook page.
XmObjectAtPoint()	Find the widget most closely associated with a point.
XmParseMappingCreate()	Create a parse mapping object.
XmParseMappingFree()	Free a parse mapping object.
XmParseMappingGetValues()	Retrieve the values of a parse mapping object.
XmParseMappingSetValues()	Set the values of a parse mapping object.
XmParseTableFree()	Free an array of parse mapping objects.
XmPrintPopupPDM()	Issue a request to the Print Display Manager.
XmPrintSetup()	Initialize a PrintShell and X Print connection.
XmPrintToFile()	Print X Print Server data to file.
XmRedisplayWidget()	Synchronously force a widget to expose itself.
XmRenderTableAddRenditions()	Add rendition objects to a render table.
XmRenderTableCopy()	Copy a render table.
XmRenderTableCvtFromProp()	Convert a text property into a render table.
XmRenderTableCvtToProp()	Convert a render table to a text property.
XmRenderTableFree()	Free a render table.
XmRenderTableGetRendition()	Find a rendition object in a render table.
XmRenderTableGetRenditions()	Find a group of rendition objects in a render table.
XmRenderTableGetTags()	Retrieve all the rendition tags in a render table.
XmRenderTableRemoveRenditions()	Remove renditions from a render table.
XmRenditionCreate()	Create a rendition object.
XmRenditionFree()	Free a rendition object.
XmRenditionRetrieve()	Retrieve the values of a rendition object.
XmRenditionUpdate()	Set the values of a rendition object.

XmScaleSetTicks()	Place tick marks along the edge of a Scale.
XmSimpleSpinBoxAddItem()	Add an item to a SimpleSpinBox.
XmSimpleSpinBoxDeletePos()	Delete an item from a SimpleSpinBox.
XmSimpleSpinBoxSetItem()	Select an item in a SimpleSpinBox.
XmSpinBoxValidatePosition()	Validate a position in a SpinBox.
XmStringByteStreamLength()	Return the length of a byte stream representation of a compound string.
XmStringComponentCreate()	Create a component in a compound string.
XmStringConcatAndFree()	Concatenate two compound strings, and free the originals.
XmStringDirectionToDirection()	Convert an XmStringDirection type into an XmDirection type.
XmStringGenerate()	Generate a new compound string using the default parse table.
XmStringGetNextTriple()	Retrieve the type, length, and value of the next compound string component.
XmStringIsVoid()	Check if a compound string is empty of components.
XmStringParseText()	Convert a string into a compound string using a parse table.
XmStringPeekNextTriple()	Look ahead at the type of the next compound string component.
XmStringPutRendition()	Add rendition components around a compound string.
XmStringTableParseStringArray()	Convert an array of strings into an array of compound strings using a parse table.
XmStringTableProposeTablist()	Calculate an XmTabList for a compound string.
XmStringTableToXmString()	Convert a compound string table into a compound string.
XmStringTableUnparse()	Convert an array of compound strings into an array of strings using a parse table.
XmStringToXmStringTable()	Convert a string into an array of compound strings.
XmStringUnparse()	Convert a compound string into a string using a parse table.
XmTabCreate()	Create a new XmTab object.
XmTabFree()	Free an XmTab object.
XmTabGetValues()	Retrieve the values of an XmTab object.
XmTabListCopy()	Copy an array of XmTab objects.
XmTabListFree()	Free an array of XmTab objects.
XmTabListGetTab()	Find an XmTab object within an array.

XmTabListInsertTabs()	Add XmTab objects to an array of tabs.
XmTabListRemoveTabs()	Remove XmTab objects from an array of tabs.
XmTabListReplacePositions()	Replace XmTab objects within an array of tabs.
XmTabListTabCount()	Return the number of XmTab objects in an array of tabs.
XmTabSetValue()	Set the value of an XmTab object.
XmTextCopyLink()	Copy a link to the primary selection into the clipboard.
XmTextGetCenterline()	Return the centerline of a vertically oriented text string.
XmTextPasteLink()	Copy a link from the clipboard selection at the insertion cursor.
XmToggleButtonSetValue()	Set the value of a tri-state ToggleButton.
XmTransferDone()	Signal the end of data transfer operations.
XmTransferSendRequest()	Transmit a multiple data transfer request.
XmTransferSetParameters()	Specify parameters for the next data transfer request.
XmTransferStartRequest()	Initiate a multiple data transfer request.
XmTransferValue()	Issue a data transfer request.

E.2 Obsolete Toolkit Functions

XmFontListAppendEntry()	Superseded by XmRenderTableAddRenditions().
XmFontListCopy()	Superseded by XmRenderTableCopy().
XmFontListCreate()	Superseded by XmRenditionCreate() and XmRenderTableAddRenditions().
XmFontListEntryCreate()	Superseded by XmRenditionCreate().
XmFontListEntryFree()	Superseded by XmRenditionFree().
XmFontListEntryGetFont()	Superseded by XmRenditionRetrieve().
XmFontListEntryGetTag()	Superseded by XmRenditionRetrieve().
XmFontListEntryLoad()	Superseded by setting the XmNloadModel of a rendition object.
XmFontListFree()	Superseded by XmRenderTableFree().
XmFontListGetNextFont()	Superseded by XmRenditionRetrieve().
XmFontListNextEntry()	Superseded by XmRenderTableGetRendition().
XmFontListRemoveEntry()	Superseded by XmRenderTableRemoveRenditions().
XmGetAtomName()	Superseded by XGetAtomName().
XmInternAtom()	Superseded by XInternAtom().
XmRepTypeInstallTearOffModelConverter()	Internally installed.
XmMainWindowSep1()	Superceded by XtNameToWidget().
XmMainWindowSep2()	Superceded by XtNameToWidget().

XmMainWindowSep3()	Superceded by XtNameToWidget().
XmMainWindowSetAreas()	Superseded by XtSetValues() of XmNhorizontalScrollBar,XmNverticalScrollBar, XmNworkWindow, XmNmenuBar, and XmNcommandWindow, resources.
XmScrolledWindowSetAreas()	Superseded by XtSetValues() of XmNhorizontalScrollBar,XmNverticalScrollBar, and XmNworkWindow resources.
XmStringCreateLtoR()	Superceded by XmStringGenerate().
XmStringGetLtoR()	Superceded by XmStringUnparse().
XmStringGetNextComponent()	Superceded by XmStringGetNextTriple().
XmStringGetNextSegment()	Superceded by XmStringGetNextTriple().
XmStringPeekNextComponent()	Superceded by XmStringPeekNextTriple().
XmStringSegmentCreate()	Superceded by XmStringComponentCreate() and XmStringConcat().

E.3 New Widget Classes

XmComboBox	A widget which combines text input with list selection.
XmContainer	A general purpose manager for providing grid, tree, or free-format layout of IconGadget children.
XmGrabShell	A popup shell which grabs the keyboard and pointer.
XmIconGadget	A gadget which supports labelled pixmaps.
XmNotebook	A layout widget which displays one child at a time.
XmPrintShell	An interface to the X11R6 Xp X Print utilities.
XmSimpleSpinBox	A widget which cycles through a set of choices.
XmSpinBox	A widget which controls cycling through possibly multiple sets of choices.

E.4 New Resources in Existing Widget Classes

VendorShell	XmNbuttonRenderTable XmNinputPolicy XmNlabelRenderTable XmNlayoutDirection XmNtextRenderTable XmNunitType

XmArrowButton	XmNdetailShadowThickness
XmBulletinBoard	XmNbuttonRenderTable XmNlabelRenderTable XmNtextRenderTable
XmDrawingArea	XmNconvertCallback XmNdestinationCallback
XmDisplay	XmNdefaultButtonEmphasis XmNenableBtn1Transfer XmNenableButtonTab XmNenableDragIcon XmNenableEtchedInMenu XmNenableMultiKeyBindings XmNenableThinThickness XmNenableToggleColor XmNenableToggleVisual XmNenableUnselectableDrag XmNenableWarp XmNmotifVersion XmNnoFontCallback XmNnoRenditionCallback XmNuserData
XmFileSelectionBox	XmNdirTextLabelString XmNfileFilterStyle XmNpathMode
XmFrame	XmNframeChildType
XmGadget	XmNbackground XmNbackgroundPixmap XmNbottomShadowPixmap XmNhighlightPixmap XmNlayoutDirection XmNtopShadowPixmap
XmLabel	XmNrenderTable

XmList	XmNhorizontalScrollBar XmNmatchBehavior XmNprimaryOwnership XmNrenderTable XmNselectColor XmNselectedPositionCount[1] XmNselectedPositions[2] XmNselectionMode XmNverticalScrollBar XmNdestinationCallback
XmManager	XmNlayoutDirection XmNpopupHandlerCallback
XmMenuShell	XmNbuttonRenderTable XmNlabelRenderTable XmNlayoutDirection
XmPanedWindow	XmNorientation XmNlayoutDirection
XmPrimitive	XmNlayoutDirection XmNconvertCallback XmNpopupHandlerCallback
XmRowColumn	XmNtearOffTitle
XmScreen	XmNbitmapConversionModel XmNcolorAllocationProc XmNcolorCalculationProc XmNinsensitiveStipplePixmap XmNuseColorObject
XmScrollBar	XmNeditable XmNsliderVisual XmNslidingMode XmNsnapbackMultiple

1.Erroneously given as XmNselectPositionCount in 2nd edition.

2.Erroneously given as XmNselectPositions in 2nd edition.

XmScrolledWindow	XmNautoDragModel XmNscrolledWindowChildType
XmScale	XmNeditable XmNrenderTable XmNshowArrows XmNsliderMark XmNsliderSize XmNsliderVisual XmNslidingMode XmNconvertCallback
XmText	XmNtotalLines XmNdestinationCallback
XmTextField	XmNdestinationCallback
XmToggleButton	XmNdetailShadowThickness XmNindeterminateInsensitivePixmap XmNindeterminatePixmap XmNtoggleMode XmNunselectColor

Index

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X