



EnhancementPak 3.0 Programmer's Reference



**Integrated Computer
Solutions, Incorporated**

Copyright © 1997-2009 Integrated Computer Solutions, Inc

The *EnhancementPak Programmer's Reference*[™] is copyrighted by Integrated Computer Solutions, Inc., with all rights reserved. No part of this book may be reproduced, transcribed, stored in a retrieval system, or transmitted in any form or by any means electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Integrated Computer Solutions, Inc.

Integrated Computer Solutions, Inc.

54 B Middlesex Turnpike, Bedford, MA 01730

Tel: 617.621.0060

Fax: 617.621.9555

E-mail: info@ics.com

Trademarks

EnhancementPak, EPak PRO, EPak, Builder Xcessory, BX, BX/Ada, Builder Xcessory PRO, BX PRO, BX/Win Software Development Kit, BX/Win SDK, Database Xcessory, DX, DatabasePak, DBPak, ViewKit ObjectPak, VKit, ICS Motif, and Ada/Motif are trademarks of Integrated Computer Solutions, Inc.

All other trademarks are properties of their respective owners.

Second printing

January 2009

Contents

How to Use This Manual

Overview	ix
Introduction to EnhancementPak.....	x
Notation Conventions	xi
epak-talk Mailing List	xii

Chapter 1—Widget Documentation Format

Overview	1
Widget Summary	2
Class Name.....	2
Class Pointer.....	2
Superclass Name	2
Creation Routine.....	3
Geometry Management	3
Classes and Inherited Resources.....	4
Resources.....	4
Resources Table Format	5
Data Types.....	6
Using Resource Names	6
Class Names	7
Resource Values	7
Constraint Resources	8
Translations and Actions	8
Compound Widget Hierarchy.....	9
Callback Routines.....	9
Convenience Routines	10

Chapter 2—Programming with EnhancementPak

Overview	11
External Symbol and Resource Naming.....	12
Compound Widgets	12
Building UNIX Applications with EnhancementPak	13
Standard Installation Example.....	14
Custom Installation Example	14
Version Information	14
Subclassing EnhancementPak Widgets.....	14
Programming Resolution-independent Interfaces	14
Strings and Memory Management	15
Utility Routine XiGetVersionInfo()	16

Chapter 3—Widget Reference

Overview	17
XiButtonBox	18
Classes and Inherited Resources	19
Resources	19
Translations and Actions	21
XiColorSelector	22
Classes and Inherited Resources	23
Resources	23
Compound Widget Hierarchy	25
XiColumn	26
Classes and Inherited Resources	27
Resources	27
Constraint Resources	29
Translations and Actions	30
XiCombinationBox	31
Geometry Management	33
Global Translations	33
Classes and Inherited Resources	34
Resources	34
Compound Widget Hierarchy	38
Callback Routines	39
Convenience Routine	39
XiDataField	40
Classes and Inherited Resources	40
Resources	41
Callback Routines	43
XiExtended18List	44
Using the Resource Database	46
Classes and Inherited Resources	46
Resources	46
Translations and Actions	51
Compound Widget Hierarchy	52
XiExt18ListCallbackStruct Structure	52
Xi18RowInfo Structure	53
Callback Routine	54
Sort Function	54
Convenience Routines	54

XiFontSelector	57
Basic Features	57
Advanced Features	58
Non XLFD Fonts	58
Resolution Control	59
Fixed or Proportional.....	59
Font Scaling.....	59
Encoding.....	59
XLFD Name Display.....	59
Classes and Inherited Resources.....	60
Resources.....	60
Compound Widget Hierarchy.....	64
XiHierarchy	66
Classes and Inherited Resources.....	66
Resources.....	67
Constraint Resources	69
Callback Routine	70
Convenience Routines	70
Class Methods.....	71
XiIconButton	73
Classes and Inherited Resources.....	74
Resources.....	75
Constraint Resources	76
Convenience Routine.....	76
XiIconButton	77
Classes and Inherited Resources.....	77
Resources.....	78
Translations and Actions	81
Callback Routines.....	82
XiOutline	83
Outline Node Types.....	84
XiOpen	84
XiClosed.....	84
XiAlwaysOpen	85
XiHidden	85
Geometry Management	85
Classes and Inherited Resources.....	85
Resources.....	86

XiPaned	87
Geometry Management	88
Classes and Inherited Resources	89
Resources	90
Constraint Resources	92
Translations and Actions	93
Convenience Routine	94
XiPanner	95
Classes and Inherited Resources	97
Resources	97
Translations and Actions	99
XiScrollReport	100
Callback Routine	101
Convenience Routine	101
XiPixmapEditor	102
Selecting and Adding Colors	104
Resizing the Pixmap	104
Panning and Zooming	105
Input and Output	105
Classes and Inherited Resources	106
Resources	106
Translations and Actions	111
Compound Widget Hierarchy	111
Convenience Routines	113
XiPorthole	114
Geometry Management	116
Classes and Inherited Resources	116
Resources	117
Callback Routine	118
Convenience Routines	119
XiStretch	120
Classes and Inherited Resources	120
Resources	121
Translations and Actions	122
Callback Routine	122
XiStretchWidgetInfo Structure	123

XiTabStack	124
Classes and Inherited Resources.....	125
Resources.....	125
Constraint Resources.....	129
Translations and Actions.....	130
XiTabStackCallbackStruct Structure.....	131
Convenience Routines.....	131
XiToolbar	132
Toolbar Popup Labels.....	132
Specifying Groups and Positions.....	133
Classes and Inherited Resources.....	133
Resources.....	133
Constraint Resources.....	135
XiToolbarCallbackStruct Structure.....	136
Convenience Routines.....	136
XiToolTip	138
Resources.....	139
Convenience Routines.....	141
XiTree	146
User Interaction.....	147
XiOpen.....	147
XiClosed.....	147
XiAlwaysOpen.....	147
XiHidden.....	148
Geometry Management.....	148
Classes and Inherited Resources.....	148
Resources.....	149
Constraint Resources.....	150

Appendix

Classes and Inherited Resources.....	153
Core Resources.....	153
Composite Resources.....	155
XmBulletinBoard Resources.....	155
XmFrame Resources.....	157
XmManager Resources.....	157
XmPrimitive Resources.....	159
XmTextField Resources.....	160

Index	163
--------------------	-----

How to Use This Manual

Overview

This chapter includes the following sections:

- **Introduction to EnhancementPak** on page x
- **Notation Conventions** on page xi
- **epak-talk Mailing List** on page xii

Overview

The following table provides an overview of each chapter of the *EnhancementPak 3.0 Programmer's Reference* manual:

<p><i>Chapter 1—Widget Documentation Format</i></p>	<p>Presents the format used to describe the EPak widgets listed in <i>Chapter 3—Widget Reference</i>, and describes the common EPak widget information.</p>
<p><i>Chapter 2—Programming with EnhancementPak</i></p>	<p>Discusses specifics of programming with EPak, including resource naming, compound widgets, strings, and memory management.</p>
<p><i>Chapter 3—Widget Reference</i></p>	<p>Technical reference section for the complete set of EPak widgets, classes, and callback and convenience routines</p>

Introduction to EnhancementPak

OSF/Motif™ has become the industry-standard graphical user interface toolkit. It is now shipped by many major hardware vendors as part of both user and development environments. Although Motif is a fine base, the basic widgets are insufficient for many development projects.

The ICS EnhancementPak™ provides a sophisticated and usable interface and very flexible widgets. The availability of sources for each EnhancementPak widget allows you to use the EPak widgets to build your own widgets. This helps you meet the requirements unique to your application and to your customer base, and will often save you time and money.

The widgets in EnhancementPak were designed to satisfy a variety of application needs, including both the need to provide the end user with additional controls and also to lay out those controls in ways that cannot easily be accomplished with the basic Motif widget set.

EPak widget categories

The widgets in EnhancementPak fit into these general categories:

- **Data-Selection and Data-Presentation Widgets**
These widgets provide the end user with the ability to directly manipulate data that your application presents, including simply selecting from existing items in a list or interactively creating data as complicated as a multi-color pixmap. Widgets that fall into this category are the `XiColorSelector`, `XiCombinationBox`, `XiDataField`, `XiExtended18List`, `XiFontSelector`, `XiIconButton`, and `XiPixmapEditor`.
- **Layout Widgets**
These widgets provide various means of arranging their children, including into hierarchical layouts and other space-saving layouts. Widgets that fall into this category are the `XiButtonBox`, `XiColumn`, `XiHierarchy`, `XiIconBox`, `XiOutline`, `XiPaned`, `XiPanner`, `XiPorthole`, `XiStretch`, `XiTabStack`, `XiToolbar`, and `XiTree`.
- **Utility Widgets**
These widgets or other pieces of code provide additional functionality. Widgets that fall into this category are the `XiToolTip`.

Notation Conventions

The following conventions are used throughout the *EnhancementPak Programmer's Reference*:

- *Italic* denotes variable names, book or chapter titles, and also indicates emphasis:
“The header files *must* be in a directory path that terminates with the path-name .”
- `Fixed width` denotes verbatim code entries. Text delimited by angle brackets indicates where you should substitute specific information:
“`subscribe <your_email_address_here>`”
- **Bold** denotes resources. The name, class, default value, type, and access for each resource are presented in each widget section as two tables, Resources and Constraint Resources. All resource names begin with **XmN** and all resource class names begin with **XmC**.

epak-talk Mailing List

You can subscribe to the `epak-talk` mailing list to communicate with other users of the ICS EnhancementPak™ and Builder Xcessory™ widgets.¹ This mailing list provides tips on different ways to use EnhancementPak widgets and how to integrate them into your development environment.

Subscribing

To subscribe to `epak-talk`, send the following in the body (not the subject line) of an e-mail message to `epak-talk-request@ics.com`:

```
subscribe
```

To subscribe to `epak-talk`, send e-mail to `epak-talk-request@ics.com` and include the following line in the body (not the subject line) of your message:

```
subscribe <your_email_address_here>
```

If you wish to subscribe to another address instead, such as a local redistribution list, you can use a command of the form:

```
subscribe other-address@your_site.your_net
```

Once you receive confirmation of your subscription request, all messages sent to `epak-talk@ics.com` are received by all subscribers. You might want to set up local redistribution lists within your organization, or have your system administrator set up this list for accessibility from within your news system.

Unsubscribing

To unsubscribe from `epak-talk`, send the following in the body (not the subject line) of an e-mail message to `epak-talk-request@ics.com`:

```
unsubscribe
```

This will unsubscribe the account from which you send the message. If you are subscribing with some other address, send a command of the following form instead:

```
unsubscribe other-address@your_site.your_net
```

Contributing

To contribute a comment, question, or suggestion to the mailing list, send an e-mail message to `epak-talk@ics.com`. The message will automatically be distributed


1. With version 3.0 of EnhancementPak, the name of the mailing list `widget-talk` has been changed to `epak-talk`. Mail sent to `widget-talk` and `widget-talk-request` will be rerouted to `epak-talk` and `epak-talk-request`.

to all members of the epak-talk mailing list.

Further Help

For a full list of commands that the mailing list server understands, send the following in the body (not the subject line) of an e-mail message to `epak-talk-request@ics.com`.

```
help
```



HOW TO USE THIS MANUAL
epak-talk Mailing List

Widget Documentation Format

1

Overview

This chapter includes the following sections:

- **Widget Summary** on page 2
- **Geometry Management** on page 3
- **Classes and Inherited Resources** on page 4
- **Resources** on page 4
- **Constraint Resources** on page 8
- **Translations and Actions** on page 8
- **Compound Widget Hierarchy** on page 9
- **Callback Routines** on page 9
- **Convenience Routines** on page 10

Widget Summary

Each widget reference section begins with a table summarizing the following information:

Application Header File	Public header file that you should include in any application that uses the widget.
Class Header File	Private header file required to subclass the widget.
Class Name	Widget's class name.
Class Pointer	Pointer to the widget's class.
Superclass Name	Widget from which this widget was subclassed.
Creation Routine	Routine that creates an instance of the widget.

Class Name

The class name is the formal name for the widget class. The documentation describing each widget uses either the full class name (such as "XiButtonBox") or the less formal nickname ("ButtonBox" or "Button Box").

The formal name is used only to set resources on all widgets of this class. The full class name is the string which is common for all widget instances.

For example, the following specification in a defaults file

```
*XiButtonBox.marginWidth: 10
```

sets the **XmNmarginWidth** resource on all instances of XiButtonBox widgets that have no more specific setting to the value 10.

Class Pointer

The class pointer is the pointer to the data structure defining the widget. It is the value that is passed to Xt functions to create the widget. Both XtCreateWidget() and XtCreateManagedWidget() take the class pointer to the widget as a function argument. The widget's creation routine uses this value.

Superclass Name

The "Superclass Name" specifies the widget class from which this widget inherits most of its behavior. It is a shortcut reference to the "Classes and Inherited Resources" description (see "*Classes and Inherited Resources*" on page 4).

Creation Routine

Each EnhancementPak widget offers a simple function that can be used to create the widget. These functions follow the format established by the Motif widget set. The name of the function is always based on the widget class name, with the word "Create" inserted after the "Xi" prefix (the exact name is listed in the summary table on the first manual page for the widget). All creation functions take the same set of arguments; they all return the identifier of the created widget.

Note: Unlike the Motif widget creation routines, the Xi creation routines are simply wrappers around the Xt functions that create widgets. The creation routines do not perform any additional processing on the widget. They are equivalent to calling `XtCreateWidget()` with the class pointer of the widget. However, it is preferable to use the creation routine in case it does do extra resource setting on the widgets in future versions of EnhancementPak.

Note: The creation routines, like the Motif creation routines, create the widgets in an unmanaged state. It is necessary to call `XtManageChild()` or `XtManageChildren()`, passing the widget identifier, to have the widget's parent allocate space to display the widget.

```
Widget XiCreateWidget( Widget parent,
                      String name,
                      ArgList args,
                      Cardinal num_args)
```

<i>parent</i>	Parent widget ID.
<i>name</i>	Name of the created widget. This name is the one used in a defaults file to define certain resource values for the widget instance.
<i>args</i>	Argument list naming resources to be set on the widget at its creation. (After creation, use <code>XtSetValues()</code> or <code>XtVaSetValues()</code> to change resource values, as normal.)
<i>num_args</i>	Number of attribute/value pairs in the argument list.

Geometry Management

The EnhancementPak widgets that handle layout offer very specialized behavior in arranging their child widgets. This section describes the mechanism by which the widget affects the size and location of its children.

Classes and Inherited Resources

The ICS EnhancementPak, like the Motif widget set, is based on the X Toolkit Intrinsic (Xt). Xt is a subclassing toolkit. It defines a base class, called **Core**, which provides a certain amount of functionality and a basic set of resources—the size and location of the widget and its background color, for example, are resources defined by **Core**. Additional widget classes subclass from **Core** and define additional resources and functionality. The complete set of resources and behavior for a widget is defined by "adding up" those of all its superclasses.

A section titled "Classes and Inherited Resources" appears in the description for widgets in EnhancementPak. It lists the full hierarchy of classes that defines each widget. The hierarchy always starts with **Core**, which is defined by Xt. Other parts of the hierarchy are defined by the Motif widget set, which provides the **XmPrimitive** and **XmManager**. Widgets that the end-user interacts with directly and that are fairly simple are subclassed from **XmPrimitive**. Widgets that do layout of other widgets, including children that they create themselves for the end-user to interact with, are subclassed from additional Xt classes, **Composite** and **Constraint**, and then from **XmManager**. Several widgets in the ICS EnhancementPak are subclassed from other Xi widgets; most are subclassed directly from **XmPrimitive** and **XmManager**, so only those class descriptions need to be referred to determine the complete capabilities of a widget in EnhancementPak.

The resources inherited from each Motif superclass are listed in the *Appendix* at the back of this book. For a complete description of each resource, refer to the OSF/Motif Programmer's Reference (Version 1.2 or better) published by The Open Group (formerly The Open Software Foundation) and Prentice-Hall.

Resources

The Resources table summarizes information about each resource that is new for the widget. The widget also contains any significant resources defined in any ancestor widget, but redefined in this widget. For information about the superclass resources, refer to the section "*Classes and Inherited Resources*" for the respective widgets.

Resources Table Format

The Resources table uses the following table template:

Name Class	Default Type	Access
---------------	-----------------	--------

Definitions

The following table defines each table item:

Table Item	Definition
Name	Resource name.
Class	Class of resource.
Type	Data type of resource.
Default	Default value of resource.
Access	<p>Access permissions of the resource:</p> <p>C indicates resource can be set at widget creation time.</p> <p>S indicates resource can be set at any time by using <code>XtSetValues</code> or a routine provided by the widget supporting the resource.</p> <p>G indicates resource value can be retrieved by using <code>XtGetValues()</code>.</p> <p>For resources of type <code>XtCallbackList</code>, "S" and "G" imply that special functions appropriate to callback resources can be used: <code>XtAddCallback()</code> and <code>XtRemoveCallback()</code>, most notably, <code>XtSetValues</code> replaces the list, so these special Xt functions should always be used for callback resources.</p>

The resource table is followed by an explanation of each resource. The descriptions are generally alphabetical, except where two closely-related resources are discussed together (for example, **XmNhorizontalMargin** and **XmNverticalMargin** for a widget supporting margin padding).

The resource name follows the conventions established by the Motif widget set. The name begins with the Xm prefix; because the Xi widgets are subclassed from Motif widgets, all resources use Xm universally (preferable to mixing Xm and Xi prefixes). The prefix is followed by "N" (for Name). The rest of the resource name describes its purpose. The initial letter is always in lower-case. The name can be composed of multiple words; if so, the name contains inter-caps.

Data Types

Generally, geometry resources (offsets, margins, spacings) are of the Xt type Dimension, while location resources (x and y) are of the Xt type Position. However, occasionally some position and location resources are of type int; these cannot be changed because of issues of backward compatibility.

Resources which are of type "unsigned char" are of size unsigned char, but the valid values for the resources are typically values chosen from a set of enumerated values or constants (which are documented).

In other cases, widgets that have defined their own types name the resource type as being of that type, which is defined in the widget's public header file. The valid values are documented.

It is important to use the correct type, especially in calls to XtGetValues(). The internal mechanism that Xt uses to read and write resource values into a widget depends greatly on the size of the resource value's being computed correctly.

Using Resource Names

For resources other than those of type XtCallbackList, the resource name is used in calls to the Xt functions XtSetValues() and XtGetValues() (among others) to change the resource value of a widget. For example, to change the **XmNmarginWidth** resource of a widget which is an XiButtonBox, the following call can be made:

```
Widget bbox = XiCreateButtonBox(shell, "topBox", NULL, 0);
Dimension margin = 15;
      :
      :
XtVaSetValues(bbox, XmNmarginWidth, margin, NULL);
```

It is important that the data type used for a resource match that in the resource table, particularly when you use XtGetValues().

In addition, the resource name can be used in a defaults file (such as \$HOME/.Xdefaults, which can be picked up automatically) to set the initial value of a resource. In this case, the ButtonBox could be created with an initial value of **XmNmarginWidth** with this line in a defaults file:

```
*topBox.marginWidth: 15
```

Strip off the "XmN" prefix to get the resource name to use in this specification.

Class Names

The resource class names also begin with the **Xm** prefix. The prefix is followed by "C" (for Class). The rest of the resource name describes its purpose. The initial letter is always in upper-case. The name can be composed of multiple words; if so, the name contains inter-caps. When more than one resource is in a class, setting a value for the resource class sets both resources. For example, both **XmNmarginWidth** and **XmNmarginHeight** are listed in resource pages as being of class **XmCMargin**. Setting this resource specification sets both resources to the value named:

```
*topBox.Margin: 15
```

Resource Values

Some resources provided by EnhancementPak have values which are not basic data types. For example, the XiButtonBox provides a resource **XmNfillOption**, which names how children are arranged within the ButtonBox. The legitimate values of **XmNfillOption** are symbolic constants such as XiFillMajor and XiFillMinor. All such enumerated values and symbolic constants in EnhancementPak follow either this format (Xi prefix followed by words with inter-caps) or the format used by the XiButtonBox's **XmNorientation** resource, which has values of XmHORIZONTAL and XmVERTICAL (Xi prefix followed by upper-cased words with underbars separating words). In either case, drop the "Xi" prefix to use the value in a defaults file (and case doesn't matter):

```
*topBox.fillOption: fillMajor  
*topBox.fillOption: fillmajor  
*topBox.orientation: horizontal
```

In addition, the widgets typically support the obvious short names for the resource values, such as "major" in this case.

Resources of other types, such as lists of strings, also have converters. Information on the converters is documented where these resources are used.

Refer to the Xt Toolkit Intrinsic documentation for more information about using defaults files and on where resource specifications may be defined and how they can automatically be included when your program runs.

Constraint Resources

The widgets in EnhancementPak that are containers for other widgets can take advantage of one of the facilities offered by their superclass—these widgets are subclasses of **XmManager**, which in turn is a subclass of the **XtConstraint** widget. That widget class offers the ability to associate data with each child. This mechanism allows the widgets in EnhancementPak internally to store information about each child widget. For example, the **XiColumn** uses this data to store information about each child widget's associated label.

The programmer's interface to this extra data is through the resource mechanism. The Constraint Resources section documents the public data values that can be set for each child, using the same format as the Resources table. For example, the **XiColumn** supports a resource **XmNentryLabelString** to set the label string associated with each child; the Constraint Resources table shows the default value of that resource and its data type.

Although this resource is offered by the **XiColumn**, the value should be set on the child of the **XiColumn**, just as though the resource were actually one supported by the child widget. The X Toolkit Intrinsic handles the overhead of determining which resources are actually for the child widget and which are constraint resources that are for the constraint information supported by the parent widget. This work by Xt is done both for direct setting of widget resources via **XtSetValues()** and also through defaults files.

Translations and Actions

The X Toolkit Intrinsic offers a facility for mapping sequences of user interaction to particular capabilities offered by widgets. For example, a certain key sequence can be set to trigger a "move to beginning of line" function in a text-editing widget. Xt provides this facility by use of Translations and Actions. Translations name key sequences and associated action names to call when the sequence has been received; internally, widgets map these action names to functions to perform the particular task.

EnhancementPak widgets generally inherit this functionality from their superclasses. Any additional functionality available as actions is documented in this section.

Refer to the X Toolkit Intrinsic documentation for information on how to define a translation table for a widget.

Compound Widget Hierarchy

Widgets in EnhancementPak can be treated as single objects. When you create an XiCombinationBox, for example, you make a single call and receive a single widget identifier; you can then manipulate the XiCombinationBox using that identifier (for example, by unmanaging or re-managing it).

However, to have complete control over the interface for the application that you are creating, you will want to have access to the widgets that the XiCombinationBox creates internally as part of its own interface.

This section lists the compound widget hierarchy for widgets in EnhancementPak that create their own children.

Refer to **Chapter 2—Programming with EnhancementPak** for details on how to manipulate these automatically-created children.

Callback Routines

In order to provide the application with notification that the end-user has done something interesting, several widgets in EnhancementPak provide callbacks. Callbacks are special resources that are manipulated using XtAddCallback() and XtRemoveCallback() and several other Xt functions. The value of the resource is a list of procedures to call when the event of interest happens. Each procedure is of type XtCallbackProc; each is called with three parameters—the widget in question, a pointer to data defined by the application and registered in the call to XtAddCallback, and a pointer to data provided by the widget. This data (often called "call_data") is typically a pointer to a structure (and cast to an XtPointer) containing extra information of use to the callback procedure. EnhancementPak widgets that provide such a structure document it in this section.

Convenience Routines

Many of the widget in EnhancementPak define functions that can be used to access data defined by the widget or to affect its behavior (the creation routine is a very specialized convenience routine). Any functions provided by the widget are listed in this section.

Functions defined by widgets in EnhancementPak have names that begin with the formal class name of the widget (for example, `XiCombinationBoxGetValue()` is defined by the `XiCombinationBox`). The first argument to the function is generally the widget itself (the `XiCombinationBox` widget that has been created) or, in rare cases, the child of that widget.

Additional arguments to the function are described for each widget. Most often, the arguments are of a data type that is defined by the C language, such as "int" or "unsigned char" or by Xt, such as "Boolean", "Dimension", or "String" (refer to your Xt documentation for details). In several cases, arguments are of data types, which are defined by the Xi widgets; additional information on these data types, such as enumerated values or pointers to structures, is included in this section or is mentioned in the section titled "Resources".

The return value of each function is also described. Several functions allocate memory in order to return the value. The documentation for these functions mentions that it is necessary to call `XtFree()` on the value to avoid a memory-leak.

Programming with EnhancementPak

2

Overview

The chapter includes the following sections:

- **External Symbol and Resource Naming** on page 12
- **Compound Widgets** on page 12
- **Building UNIX Applications with EnhancementPak** on page 13
- **Version Information** on page 14
- **Subclassing EnhancementPak Widgets** on page 14
- **Programming Resolution-independent Interfaces** on page 14
- **Strings and Memory Management** on page 15
- **Utility Routine XiGetVersionInfo()** on page 16

External Symbol and Resource Naming

Most external symbols defined by the EnhancementPak (for example: functions, types, structure definitions, enumeration values, and widget class names) begin with **Xi**.

Resource names (**XmN**) and classes (**XmC**) are the exceptions.

Compound Widgets

Many EnhancementPak widgets automatically create their children. These compound *widgets* combine simple components to accomplish more complex tasks such as selecting a font or editing a pixmap.

Passing resources to components

The resource list used at creation time, or during an `XtGetValues()` call, is passed down to all widgets created. Before the information is passed down, the following resources are removed:

- **XmNdestroyCallback**
- **XmNheight**
- **XmNnavigationType**
- **XmNsensitive**
- **XmNuserData**
- **XmNwidth**
- **XmNx**
- **XmNy**

In addition, the following constraint resources are removed:

- **XmNallowResize**
- **XmNbottomAttachment**
- **XmNbottomOffset**
- **XmNbottomPosition**
- **XmNbottomWidget**
- **XmNleftAttachment**
- **XmNleftOffset**
- **XmNleftPosition**
- **XmNleftWidget**
- **XmNpaneMaximum**
- **XmNpaneMinimum**
- **XmNpreferredPaneSize**
- **XmNresizeToPreferred**
- **XmNrightAttachment**
- **XmNrightOffset**
- **XmNrightPosition**
- **XmNrightWidget**
- **XmNshowSash**
- **XmNskipAdjust**
- **XmNtopAttachment**
- **XmNtopOffset**
- **XmNtopPosition**
- **XmNtopWidget**

Passing the resource list in this way allows all widgets in the compound widget to, for example, change color when you set the **XmNbackground** resource, while still removing resources that usually have different values for each sub-widget in the compound widget.

Controlling resources of components

If you need more control over the individual components of a compound widget, the widget hierarchy for every compound widget is included in the documentation of each widget. Use `XtNameToWidget()` to retrieve the widget ID of any widget and modify the resources of the component widgets. When you use `XtGetValues()` to retrieve resource attributes, compound widgets will not pass the get values call down to any of their children. The `XtGetValues()` call must be made directly on the child to obtain the resource of the child.

Resource specifications

Although the Compound Widgets expose enough details of their implementation to make it possible to fetch a child widget and set resources directly on it, the compound widgets often do not directly support the resources on the child except by coincidence. That is, using `XtSetValues()` on **XmNLabelString** on an `XiCombinationBox` will set the **XmNLabelString** of its child `XmLabel`, but only because the resource arguments are passed on to the child. The `XiCombinationBox` does not support a resource **XmNLabelString** itself.

This means that a resource specification for an `XiCombinationBox` named "comboBox" of the following form:

```
*comboBox.labelString: Choose Items
```

will not work, because **XmNLabelString** is not a resource of `comboBox`. However, this specification:

```
*comboBox*labelString: Choose Items
```

has the desired effect, as does a resource specification that names the `XmLabel` child explicitly, by name or by class.

Building UNIX Applications with EnhancementPak

To build an application using the EnhancementPak widgets, compile the source code and link against the additional library `libEpak.a`. If you have performed a standard installation, the library will be in `/usr/lib`. If you have performed a custom installation, you must tell the compiler where to find the library by using the `-L` flag to the linker.

Header files

Header files are expected to be installed in `/usr/include/Xi`. If you have installed them in some other location, be sure to use the `-I` flag to specify the replacement for `/usr/include`.

Note: Regardless of location, the `Xi` path is coded into the application. Therefore, the header files must be under a directory path that terminates with `Xi/<header_file>`.

Standard Installation Example

For a standard installation:

- Widget Headers are installed in `/usr/includeXi/*.h`
- Widget Libraries are installed in `/usr/lib/libEPak.a`
`% cc main.c -lEPak -lXm -lXt -lX11`

Custom Installation Example

In this example, the installation is to pathname `/usr/widgets`:

- Widget Headers are installed in `/usr/widgets/Xi/*.h`
- Widget Libraries are installed in `/usr/widgets/libEPak.a`
`% cc main.c -I/usr/widgets -L/usr/widgets/lib
-lEPak -lXm -lXt -lX11`

Version Information

EnhancementPak Version Strings are defined in `Xi.h`. These strings mimic the OSF/Motif version string definitions except that they are version strings for the EnhancementPak. The version definitions include the following:

<code>XiVERSION</code>	Major release number
<code>XiREVISION</code>	Minor release number
<code>XiUPDATE_LEVEL</code>	Patch level
<code>XiVersion</code>	Combined version number
<code>XiVERSION_STRING</code>	String describing the release

Subclassing EnhancementPak Widgets

Each EnhancementPak widget uses the `XtGeometryYes` geometry management policy. When a call to `XtMakeGeometryRequest` is made, the widget's resize procedure is *never* called, even if the return is set to `XtGeometryYes`.

Programming Resolution-independent Interfaces

The EPak widgets support resolution-independent programming used by the Motif widget set. This mechanism allows you to specify their interfaces in terms of units other than pixels, for example, points, or thousandths of inches. These units are converted to be correct on the display on which the application is running. The **XmNunitType** resource on the widgets controls the mechanism. This resource inherits from both **XmPrimitive** and **XmManager**; and its value specifies the units

to be used in setting widgets' widths, heights, margins, and other sizing and positioning values of type `Dimension`. A widget's `XmNmarginWidth` resource, for example, can be specified with a value of 700 and an `XmNunitType` of `Xm100TH_MILLIMETERS`, which may convert to 40 pixels on a particular display.

Resolution independent mechanism

EPak widgets also support the resolution-independent mechanism for new resources that are of type `Dimension` and represent margins and widget geometry characteristics described in pixel units. This enables the EPak widgets to be used with other Motif-based widgets in coordinate systems based on units other than pixels.

Strings and Memory Management

Several resources supported EnhancementPak widgets have values that are strings—either C-language simple string "char *" or the Motif "XmString", or arrays of them. The EPak widgets follow conventions established by the Motif widget set for handling resources with string data types:

- When the widget is given a string value or a string table, either at initialization or later through a call to `XtSetValues()` or an equivalent routine, it makes a copy of the string or string table. This releases the application from the burden of maintaining a copy of the string data.
- When the widget is asked for a value that is an array of simple strings or `XmStrings` (an `XmStringTable`), it returns a pointer to the list. The application should make a copy of the data before performing operations on it.
- When the widget is asked for a value that is a simple string, it returns a pointer to the string. The application should make a copy of the data before performing operations on it.
- When the widget is asked for a value that is an `XmString`, it returns a copy of the `XmString`. The application should free the string using `XmStringFree()`.

In cases where the data returned by the widget is allocated when it would not otherwise be expected to be (for example, when the string is constructed from the internal values), the description for the resource notes how to free the string.

Utility Routine *XiGetVersionInfo()*

XiGetVersionInfo() returns information about the version of the EPak library to which the application is linked. This information allows the applications to make runtime decisions about how to operate with various versions of the EPak widget set when there are significant differences between library versions. By checking the version, the application takes advantage of new features while continuing to work with old versions of the EPak widget set.

The information returned is a pointer to the following structure (which should not be freed):

```
typedef struct {
    int  identifier;
    int  version;
    int  revision;
    int  update_level;
    char  (information;
}XiVersionInfo;
XiVersionInfo *XiGetVersionInfo (void)
```

The *version*, *revision*, and *update_level* members correspond to the “dot” values of the EPak version. For example, version 3.5 with two update patches would have values of 3, 5, and 2. The identifier information is provided so that application code can make the simplest possible comparison. It is the “version” member times 1000 plus the revision. In this case, the value is 3005. The information field is a string (which should not be freed), containing information that you can read about the software.

Note: Because this function is new to EnhancementPak 3.0 and applications may be compiled against earlier version of EnhancementPak, typical use is bracketed by this compile-time check

```
#      if      (XiVersion >=3000)
                /*code using XiGetVersionInfo here */
#      endif
```

Widget Reference

Overview

This chapter alphabetically lists and describes the following EPak widgets:

- **XiButtonBox** on page 18
- **XiColorSelector** on page 22
- **XiColumn** on page 26
- **XiCombinationBox** on page 31
- **XiDataField** on page 40
- **XiExtended18List** on page 44
- **XiFontSelector** on page 57
- **XiHierarchy** on page 66
- **XiIconBox** on page 73
- **XiIconButton** on page 77
- **XiOutline** on page 83
- **XiPaned** on page 87
- **XiPanner** on page 95
- **XiPixmapEditor** on page 102
- **XiPorthole** on page 114
- **XiStretch** on page 120
- **XiTabStack** on page 124
- **XiToolbar** on page 132
- **XiToolTip** on page 138
- **XiTree** on page 146

XiButtonBox

UNIX Application Header File	<code>Xi/ButtonBox.h</code>
UNIX Class Header File	<code>Xi/ButtonBoxP.h</code>
Class Name	<code>XiButtonBox</code>
Class Pointer	<code>xiButtonBoxWidgetClass</code>
Superclass Name	<code>XmManager</code>
Creation Routine	<code>XiCreateButtonBox</code>

The `ButtonBox` widget manages children (usually buttons) in a single column or single row layout, as shown in Figure 1.

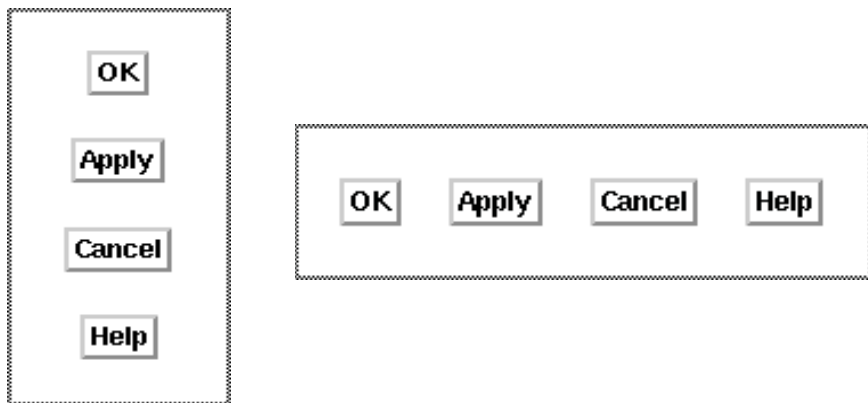


Figure 1. ButtonBox Widgets with `XmOrientation` Set to `XmVERTICAL` and `XmHORIZONTAL`, Respectively

The `ButtonBox` maintains equal spacing between its children at all times and attempts to adjust its height and width so that all children will fit. If this is not possible, due to parent or application programmer constraints, the `ButtonBox` resizes its children to fit within the available space.

Because the `ButtonBox` sizes its children equally, you can use several `ButtonBox` widgets next to one another to arrange children in a grid; if the `ButtonBox` widgets are the same size and have the same number of children, then those children will have the same size across their parent `ButtonBox` widgets.

Classes and Inherited Resources

ButtonBox inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Note: The term major direction refers to the direction of orientation (horizontal or vertical), and minor direction refers to the perpendicular direction. The major and minor dimensions are the values in the respective directions.

Name Class	Default Type	Access
XmNequalSize XmCEqualSize	False Boolean	CSG
XmNfillOption XmCfillOption	XiFillNone unsigned char	CSG
XmNmarginHeight XmCMargin	0 Dimension	CSG
XmNmarginWidth XmCMargin	0 Dimension	CSG
XmNorientation XmCOrientation	XmHORIZONTAL unsigned char	CSG

XmNequalSize

Specifies whether the children are to be maintained with equal sized heights and widths. The chosen height and width for the children is found by asking each child for its preferred size and taking the largest value in each direction.

XmNfillOption

Specifies how to use any extra space once all children have been sized according to either their preference or **XmNequalSize**. Figures 2 to 5 illustrate the four options:

XiFillNone

No automatic filling is performed. Center the children in the minor direction and place the children with equal padding between them in the major direction.



Figure 2. ButtonBox with $XmNfillOption = XiFillNone$ and $XmNoorientation = XmHORIZONTAL$

XiFillMinor

Equal padding between children in the major direction, but force all the children to take the value of the ButtonBox minor dimension as their own minor dimension.

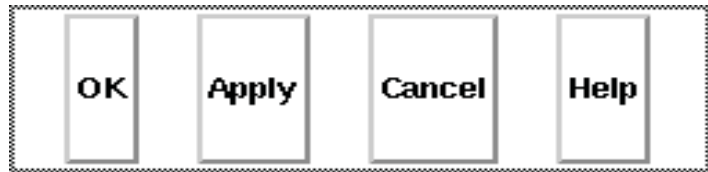


Figure 3. ButtonBox with $XmNfillOption = XiFillMinor$ and $XmNoorientation = XmHORIZONTAL$

XiFillMajor

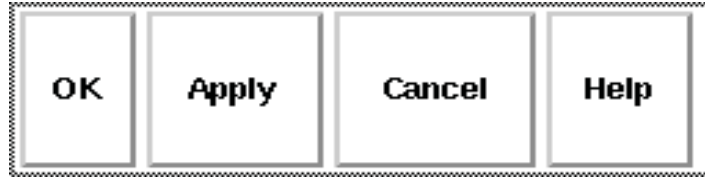
Center the children in the minor direction, but expand all the children in their major direction so that there is no padding between them. Expand the children such that their relative sizes remain constant.



Figure 4. ButtonBox with $XmNfillOption = XiFillMajor$ and $XmNoorientation = XmHORIZONTAL$

XiFillAll

This option combines the placement actions and sizing actions of XiFillMinor and XiFillMajor.



*Figure 5. ButtonBox with XmNfillOption = XiFillAll
and XmNorientation = XmHORIZONTAL*

Regardless of the fill mode, the ButtonBox widget always leaves the specified margin between its edge and the nearest child.

XmNmarginHeight**XmNmarginWidth**

Specifies the number of pixels used as a margin around the entire group of children. The **XmNmarginHeight** value applies to the top and bottom margins, while the **XmNmarginWidth** value applies to the left and right margins.

XmNorientation

Specifies whether children are to be placed in a row or a column. The orientation can be either XmHORIZONTAL or XmVERTICAL. If the orientation is XmHORIZONTAL, the children are placed in a row with the major direction being width and the minor direction being height. If the value is XmVERTICAL, the children are placed in a column with the major direction being height and the minor direction being width.

Translations and Actions

The ButtonBox manager inherits all of its translations and actions from XmManager.

XiColorSelector

UNIX Application Header File	Xi/ColorS.h
UNIX Class Header File	Xi/ColorSP.h
Class Name	XiColorSelector
Class Pointer	xiColorSelectorWidgetClass
Superclass Name	XmManager
Creation Routine	XiCreateColorSelector

The ColorSelector widget allows users to choose a color either by using a set of red/green/blue sliders or by choosing from a list of all colors available in the RGB database. The name or RGB value, as well as the color selected, are dynamically displayed. Figure 6 shows the ColorSelector in both list and slider modes.

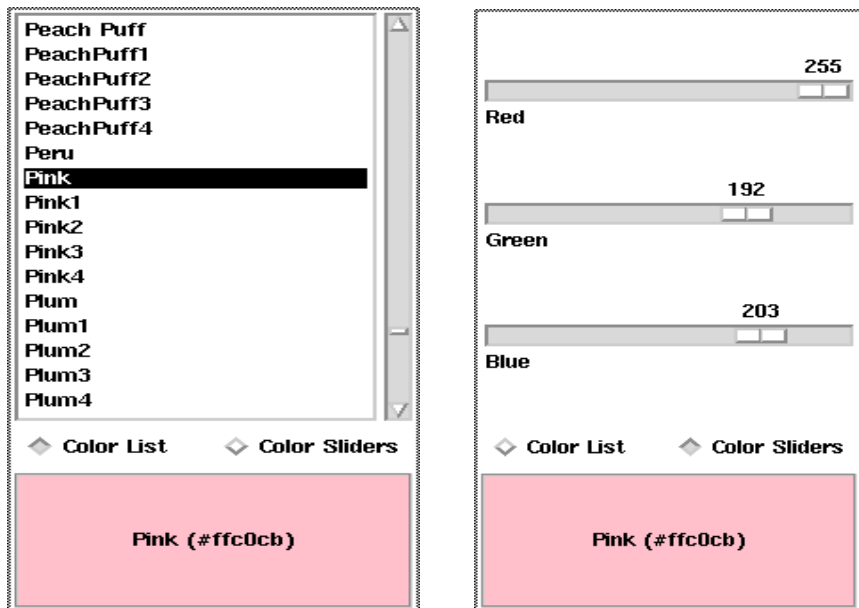


Figure 6. *XiColorSelector* Widgets with *XmNcolorMode* Set to *XiListMode* and *XiScaleMode*

Classes and Inherited Resources

ColorSelector inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNblueSliderLabel XmCSliderLabel	“Blue” XmString	CSG
XmNcolorListTogLabel XmCTogLabel	“Color List” XmString	CSG
XmNcolorMode XmCColorMode	XiScaleMode XiColorMode	CSG
XmNcolorName XmCString	“White” String	CSG
XmNfileReadError XmCFileReadError	“Could not read rgb.txt file:” XmString	CSG
XmNgreenSliderLabel XmCSliderLabel	“Green” XmString	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG
XmNnoCellError XmCNoCellError	“\n\nNo Color Cell Available!” XmString	CSG
XmNredSliderLabel XmCSliderLabel	“Red” XmString	CSG
XmNrgbFile XmCString	“/usr/lib/X11/rgb.txt” String	CSG
XmNsliderTogLabel XmCTogLabel	“Color Sliders” XmString	CSG

XmNblueSliderLabel

Specifies the string appearing for the label of the blue slider.

XmNcolorListTogLabel

Specifies the string appearing for the label of the color list toggle.

XmNcolorMode

Specifies the mode (*XiListMode* or *XiScaleMode*) that the *ColorSelector* should use when it is created. The user can then freely change modes by using a pair of radio buttons in the *ColorSelector*. A type converter is registered to convert the strings “ScaleMode” and “ListMode” to color modes for use with the resource database.

XmNcolorName

Specifies the color name currently displayed. Modifying this value changes the color displayed in the *ColorSelector* or queried to find the color the user has selected. The string returned here is either a color name or a pound sign (#) followed by a set of RGB values as specified in the Xlib specification.

XmNfileReadError

Specifies the message that is displayed when the *ColorSelector* cannot read the *rgb.txt* file. The message is displayed at the top of the window in which the color list would normally appear.

XmNgreenSliderLabel

Specifies the string appearing for the label of the green slider.

XmNmarginHeight**XmNmarginWidth**

Specifies the number of pixels between each child in the *ColorSelector* and between the outside children and the edge of the *ColorSelector* widget.

XmNnoCellError

Specifies the message that is displayed in the sample color field when the *ColorSelector* cannot allocate a read/write color cell.

XmNredSliderLabel

Specifies the string appearing for the label of the red slider.

XmNrgbFile

Specifies the name of the file to be loaded, which contains the valid color names. The list is sorted by name and the duplicates removed before being shown to the user. The file format should be the same as the *rgb.txt* file shipped with X11.

Although the file is loaded from the machine on which the X client is running, the colors are displayed by the X server, which may have a different list of colors. Most X11 distributions have the same basic set of colors, however, which is used by both the client- and server-side code.

XmNsliderTogLabel

The string that appears for the label of the color slider toggle.

Compound Widget Hierarchy

The ColorSelector is composed of several sub-widgets. Most resource values that are passed to the ColorSelector through the argument list—either at creation time or by XtSetValues()—are then passed to each of the widget’s children. An XtGetValues() request for a child widget’s resource value must be made explicitly on the child. For more information on passing arguments to the EnhancementPak compound widgets and retrieving the widget ID’s of the child widgets, refer to “Compound Widgets” on page 12.

Consult the *OSF/Motif Programmer’s Reference* for the list of any child widget’s resources.

XiColorSelector	<named by application>
XmScrolledWindow	scrolled
XmScrollBar	ListvScrollBar
XmScrollBar	ListhScrollBar
XmList	list
XiButtonBox	buttonBox
XmScale	scale /*repeat for three scales */
XmLabelGadget	scale_title
XmScrollBar	scale_scrollbar
XmRowColumn	radioBox
XmToggleButton	colorListToggle
XmToggleButton	colorSlidersToggle
XmFrame	colorFrame
XmLabel	colorWindow

XiColumn

Application Header File	Xi/Column.h
Class Header File	Xi/ColumnP.h
Class Name	XiColumn
Class Pointer	xiColumnWidgetClass
Superclass Name	XmBulletinBoard
Creation Routine	XiCreateColumn

The Column widget displays its children stacked in a column, each with an optional associated label: labels appear in one column, and children in another. This is useful for displaying, for example, labeled data-entry fields, as illustrated in Figure 7.

Figure 7. Column Widget with *XmOrientation* Set to *XmVERTICAL*

It can also display all label-child pairs in a horizontal orientation, as illustrated in Figure 8.

Figure 8. Column Widget with *XmOrientation* Set to *XmHORIZONTAL*

This widget offers several constraint resources that allow specification of characteristics of the label, such as displaying text or a pixmap, alignment of text, font to use, and so forth. It also offers several resources for setting defaults for children that specify no specific values.

Classes and Inherited Resources

Column inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XmBulletinBoard**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNdefaultEntryLabelAlignment XmCAlignment	XiALIGNMENT_BEGINNING unsigned char	CSG
XmNdefaultEntryLabelFontList XmCFontList	NULL XmFontList	CSG
XmNdefaultFillStyle XmCFillStyle	XiFILL_RAGGED unsigned char	CSG
XmNdistribution XmCDistribution	XiDISTRIBUTE_TIGHT unsigned char	CSG
XmNitemSpacing XmCItemSpacing	2 Dimension	CSG
XmNlabelSpacing XmCLabelSpacing	10 Dimension	CSG
XmNorientation XmCOrientation	XmVERTICAL unsigned char	CSG

XmNdefaultEntryLabelAlignment

Specifies the default **XmNentryLabelAlignment** to use when a child specifies no significant value. Resources that specify Alignment have values of XiALIGNMENT_BEGINNING, XiALIGNMENT_CENTER, XiALIGNMENT_END, and XiALIGNMENT_UNSPECIFIED. Valid string values that can be used in a resources file are: alignment_unspecified, unspecified, alignment_beginning, beginning, alignment_center, center, alignment_end, end.

XmNdefaultEntryLabelFontList

Specifies the default **XmNentryLabelFontList** to use when a child specifies no significant value. If unspecified, uses the **XmBulletinBoard**'s **XmNlabelFontList** resource.

XmNdefaultFillStyle

Specifies the default **XmNfillStyle** to use when a child specifies no significant value.

XmNdistribution

Specifies whether the spacing between each pair of rows should be increased equally (**XiDISTRIBUTE_SPREAD**) or remain constant (**XiDISTRIBUTE_TIGHT**) when the column is resized vertically to be larger than its natural size. This resource has no effect if any child has its **XmNstretchable** resource set to True. This resource is valid only when the orientation is vertical.

XmNitemSpacing

Specifies the spacing between each pair of rows (in vertical orientation) or between pairs of labels and children (in horizontal orientation).

XmNlabelSpacing

Specifies the spacing between the column containing the labels and the column containing the **XiColumn**'s children.

XmNorientation

Specifies the layout direction of the **XiColumn**. When **XmVERTICAL**, the widgets and their associated labels are laid out in two vertical columns. When **XmHORIZONTAL**, the widgets and their associated labels are laid out in a single row.

Constraint Resources

The visual appearance of columns is affected by setting constraint resources on the children of the `XiColumn`.

These resources are derived from those supported by `XmLabel`; see the manual page for `XmLabel` for valid values and usage.

Name Class	Default Type	Access
<code>XmNentryLabelAlignment</code> <code>XmCAlignment</code>	<code>XiALIGNMENT_UNSPECIFIED</code> unsigned char	CSG
<code>XmNentryLabelFontList</code> <code>XmCFontList</code>	<code>XmFontList</code> NULL	CSG
<code>XmNentryLabelPixmap</code> <code>XmCLabelPixmap</code>	<code>Pixmap</code> <code>XmUNSPECIFIED_PIXMAP</code>	CSG
<code>XmNentryLabelString</code> <code>XmCLabelString</code>	<code>XmString</code> NULL	CSG
<code>XmNentryLabelType</code> <code>XmCLabelType</code>	unsigned char <code>XmSTRING</code>	CSG
<code>XmNfillStyle</code> <code>XmCFillStyle</code>	unsigned char <code>XiFILL_UNSPECIFIED</code>	CSG
<code>XmNshowEntryLabel</code> <code>XmCShowLabel</code>	Boolean True	CSG
<code>XmNstretchable</code> <code>XmCStretchable</code>	Boolean False	CSG

XmNentryLabelAlignment

Specifies justification of text within the child's associated label. Valid values are the same as those for **XmNdefaultEntryLabelAlignment**.

XmNentryLabelFontList

Specifies fontList used to render the text within the child's associated label.

XmNentryLabelPixmap

Specifies the pixmap used in the child's associated label.

XmNentryLabelString

Specifies the text used in the child's associated label.

XmNentryLabelType

Specifies whether to display a string (XmSTRING) or a pixmap (XmPIXMAP) in the child's associated label.

XmNfillStyle

Specifies whether the child should be displayed at its natural size (XiFILL_RAGGED) or stretched to fill the entire width of the column it is displayed within (XiFILL_FLUSH). XiFILL_UNSPECIFIED uses the value of the XiColumn's **XmNdefaultFillStyle**.

XmNshowEntryLabel

Specifies whether or not to display the child's associated label.

XmNstretchable

Specifies whether the child should expand in size proportionately when the XiColumn is resized vertically to be larger than its natural size.

Translations and Actions

The XiColumn widget defines no translations of its own; nor does it augment the translations of its children.

XiCombinationBox

UNIX Application Header File	Xi/ComboBox.h
UNIX Class Header File	Xi/ComboBoxP.h
Class Name	XiCombinationBox
Class Pointer	xiCombinationBoxWidgetClass
Superclass Name	XmManager
Creation Routine	XiCreateCombinationBox

The `CombinationBox` widget allows users to select elements from a list of choices and enter their own values in a `Text` widget.

To conserve screen space, the list of choices is shown only when the user clicks the down arrow button. The choices can then be selected from this list. If the list widget is in `Browse Select` mode (the default) or `Single Select` mode (that is, the **XmNselectionPolicy** value of the `XmList`, which is part of the `CombinationBox`), the list is automatically unposted when the user selects an item in the list. When the list is in other modes, multiple items can be selected and the list can be posted by either another click on the arrow button, a click outside the list, or double-clicking an item.

When using keyboard traversal, the list can be posted by selecting the arrow button or `Alt+(Down Arrow)`, and unposted either by clicking the `osfActivate` key, a carriage return, or `Alt+(Up Arrow)`.

When the list is unposted, the selected item or items are placed in the `Text` widget separated by commas. Typing the `ESC` key when the list is posted restores the `CombinationBox` to the state it was in before the list was posted.

If the text field area is non-editable, click anywhere in the text field to post the list. Figures 9 and 10 show a `CombinationBox` in unposted and posted states:



Figure 9. `CombinationBox` Widget in Unposted State

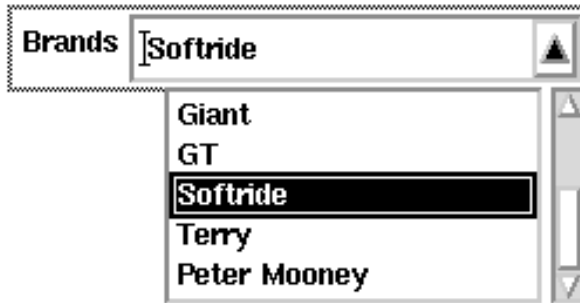


Figure 10. *CombinationBox* Widget in Posted State

Changeable resources

Changeable resources include:

- Margins
- Location of the left edge of the list
- Whether the label is shown
- Whether the TextField widget can be edited
- Whether the text in the text field is verified against the choices available in the list

Changes caused by setting children resources

Set resources for the children of the *CombinationBox* to allow you to change the following items:

- Contents of the list
- Number of items visible in the list
- Initial contents of the text field
- Value of the label, and whether it can be changed dynamically

Geometry Management

- Determining size** The `CombinationBox` widget lays out its children (the label, text and arrow) in a row. If the `CombinationBox` cannot size to the dimensions it desires, the size is determined as follows:
1. The arrow is always given its requested size.
 2. If the `CombinationBox` is larger than its desired size, all extra space is given to the `Text` widget.
 3. If the `CombinationBox` is smaller than its desired size, the `Text` and label widgets are each sized smaller than they desire in exactly the same ratio. For example:

If	Then
The amount of space for the label and text is 100 pixels, and the label wants to be 50 pixels wide, and the text wants to be 120 pixels wide	The text and label resize proportionally and the label is 40 pixels wide, and the text is 80 pixels wide.

The popup shell widget containing the list is sized such that the width of the scrolled list widget is the width of the `Text` widget minus the value of `XmNpopupOffset`. If the popup shell resides in a customized `CombinationBox`, it is allowed to be whatever size the child of the popup shell would like to be.

Global Translations

The `XmNcomboTranslations` resource allows a set of key bindings that will work no matter which of the `CombinationBox`'s children has the input focus. This resource contains the binding for `Alt-Up` and `Alt-Down` to pop the list up and down, as well as the binding for `Escape` to cancel the current selection.

Alt <Key>osfDown:	<code>XiComboListDown()</code>
Alt <Key>osfUp:	<code>XiComboListUp()</code>
Any <Key>osfCancel:	<code>XiComboListCancel()</code>

`XiComboListDown()`

Displays the `CombinationBox` widget's list on the user's screen, as if the "down" arrow button was clicked with the list hidden. This action has no effect when the list is displayed.

XiComboListUp()

Removes the `CombinationBox` widget's list from the user's screen, as if the "up" arrow button was clicked with the list displayed. This action has no effect when the list is hidden.

XiComboCancel()

Removes the `CombinationBox` widget's list from the user's screen, but also restores the state of the `CombinationBox` to what it was before the list was shown. This allows the user to back out of any changes made to the list.

Classes and Inherited Resources

`CombinationBox` inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to "Appendix" on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer's Reference*.

Resources

Name Class	Default Type	Access
XmNcomboTranslations XmCTranslations	(computed) XtTranslations	CG
XmNcustomizedCombinationBox XmCBoolean	False Boolean	CSG
XmNcomboTranslations	see below	CG
XmNeditable XmCBoolean	True Boolean	CSG
XmNhorizontalMargin XmCMargin	2 Dimension	CSG
XmNnewVisualStyle XmCNewVisualStyle	True Boolean	CSG
XmNpopupCursor XmCCursor	left_ptr Cursor	CSG
XmNpopupOffset XmCPopupOffset	15 int	CSG

Name (continued) Class	Default Type	Access
XmNpopupShellWidget XmCWidget	NULL Widget	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNshowLabel XmCBoolean	True Boolean	CSG
XmNupdateShellCallback XmCCallback	NULL XtCallbackList	CSG
XmNupdateTextCallback XmCCallback	NULL XtCallbackList	CSG
XmNuseTextField XmCUseTextField	True Boolean	CG
XmNverify XmCVerify	True Boolean	CSG
XmNverifyTextCallback XmCCallback	NULL XtCallbackList	CSG
XmNverticalMargin XmCMargin	2 Dimension	CSG
XmNvisibleItemCount XmCVisibleItemCount	5 int	CSG

XmNcomboTranslations

Overrides translations of all children of the `CombinationBox`. They are exposed in a resource to facilitate custom input methods and focus processing. **Be careful.**

The default translations added are:

Ctrl <Key>osfDown:	XiComboListPost()
Ctrl <Key>osfUp:	XiComboListUnpost()
Any <Key>osfCancel:	XiComboListCancel()

where the action `XiComboListPost` posts the list, the action `XiComboListUnpost` unposts it, and the action `XiComboListCancel` unposts it without affecting the previous selected values.

XmNcustomizedCombinationBox

Creates a custom `CombinationBox` that contains something other than a list. If `True`, the widget will not automatically create a popup shell and list widget. That is, a shell must be provided to the `CombinationBox` using the **XmNpopupShellWidget** resource. Just before the shell is posted, the **XmNupdateShellCallback** is called. Just after the shell is unposted, the **XmNupdateTextCallback** is called. If **XmNverify** is `True`, the **XmNverifyTextCallback** is called when the `CombinationBox` needs to verify the contents of the `TextField` widget against the allowable values in the custom shell.

XmNeditable

Specifies whether the user is allowed to type into the `CombinationBox`'s `TextField` widget. If **XmNeditable** is `False`, selecting the `TextField` posts the list.

XmNhorizontalMargin**XmNverticalMargin**

Specifies the number of pixels between each widget and its neighbor or the edge of the `CombinationBox`.

XmNnewVisualStyle

In `EnhancementPak 3.0`, the visual style of the `XiCombinationBox` widget has changed to be more in keeping with the style of other popular graphical toolkits. The old style is still available by setting this resource to `False`.

XmNpopupCursor

Specifies the cursor displayed when the `CombinationBox`'s list is posted.

See Appendix B of *X Window System* by Robert Scheifler et al., for the choices.

XmNpopupOffset

Specifies the number of pixels between the left edge of the `Text` widget and the left edge of the popup list. Positive values mean the `TextField` widget's left edge is farther to the left, negative values mean the `List` widget's left edge is farther to the left. If this is a non-custom `CombinationBox`, the right edge of the text and the right edge of the arrow button always line up.

XmNpopupShellWidget

Specifies the widget identifier for the shell that is posted when the arrow is clicked. If **XmNcustomizedCombinationBox** is `False`, this widget is automatically created by the `CombinationBox`.

The shell should be created with **XmNoverrideRedirect** set to `True` (it is typically a `topLevelShellWidgetClass` created via `XtCreatePopupShell`).

XmNshadowThickness

Specifies the shadow around the XiCombinationBox's component widgets when **XmNnewVisualStyle** is True.

XmNshowLabel

Specifies whether or not to display the CombinationBox Label widget to the left of the TextField.

XmNupdateShellCallback**XmNupdateTextCallback**

Specifies the list of callback routines called when either the shell widget contents or the Text widget need to be updated to correspond with the other. The shell is updated just before it is posted. The text is updated just after the shell is unposted. If **XmNcustomizedComboBox** is False, the updates are done automatically by the CombinationBox. These routines are called to inform the application that an action has been taken, in case it would like to do any further processing.

XmNuseTextField

If this is True (the default), the CombinationBox creates an XmTextField widget to use as the text display area. When false, an XmText widget is created.

XmNverify

If True, the CombinationBox verifies the value of the text field against the list whenever it loses focus or on a carriage return.

XmNverifyTextCallback

This routine is called whenever the TextField widget's contents need to be verified against the popup shell's contents. If the **XmNcustomizedComboBox** resource is False, the CombinationBox has already performed the verification when this routine is called.

XmNvisibleItemCount

Specifies the number of items visible in the associated list.

Compound Widget Hierarchy

The `CombinationBox` is composed of several sub-widgets. Most resource values that are passed to the `CombinationBox` through the argument list—either at creation time or by `XtSetValues()`—are then passed to each of the widget’s children. An `XtGetValues()` request for a child widget’s resource value must be made explicitly on the child. For more information on passing arguments to the `EnhancementPak` compound widgets and retrieving the widget ID’s of the child widgets, refer to “*Compound Widgets*” on page 12.

Refer to the *OSF/Motif Programmer’s Reference* for the list of any child widget’s resources.

<code>XiCombinationBox</code>	<named by application>
<code>XmLabel</code>	<code>label</code>
<code>XmTextField</code>	<code>text</code>
<code>XmArrowButton</code>	<code>arrow</code>
<code>OverrideShell</code>	<code>popupShell</code>
<code>XmScrolledWindow</code>	<code>listSW</code>
<code>XmScrollBar</code>	<code>ListvScrollBar</code>
<code>XmScrollBar</code>	<code>ListhScrollBar</code>
<code>XmList</code>	<code>list</code>

The popup shell and its children are only created when **`XmNcustomizedCombinationBox`** is `False`.

The most commonly used resources of the `CombinationBox` descendants are listed here for convenience.

Name	Widget	Type	Initial Value
<code>XmNitems</code>	<code>XmList</code>	<code>XmStringTable</code>	<code>NULL</code>
<code>XmNitemCount</code>	<code>XmList</code>	<code>int</code>	<code>0</code>
<code>XmNvalue</code>	<code>XmText[Field]</code>	<code>String</code>	<code>“”</code>
<code>XmNlabelString</code>	<code>XmLabel</code>	<code>XmString</code>	<code>“label”</code>
<code>XmNselectionPolicy</code>	<code>XmList</code>	<code>unsigned char</code>	<code>XmBROWSE_SELECT</code>

`XmNitems`

Specifies the choices that are displayed in the popup list.

`XmNitemCount`

Specifies the number of items in the popup list.

XmNvalue

Specifies the string displayed in the Text widget.

XmNlabelString

Specifies the string displayed as the label of the ComboBox.

XmNselectionPolicy

Determines how the user can make selections in the list, particularly single versus multiple selections.

Callback Routines

Each callback passes to its list of callback routines a pointer to a structure of type `XmAnyCallbackStruct`. The "reason" member in that structure is `XiCR_UPDATE_TEXT` for the **XmNupdateTextCallback**, `XiCR_UPDATE_SHELL` for the **XmNupdateShellCallback**, and `XiCR_VERIFY_TEXT` for the **XmNverifyTextCallback**.

Convenience Routine

XiCombinationBoxGetValue()

Retrieves the value from the ComboBox TextField widget.

```
String XiCombinationBoxGetValue(Widget w)
```

w

The ComboBox TextField widget.

`XiCombinationBoxGetValue()` returns the string contained in the TextField widget of the ComboBox. This string is allocated with `XtMalloc()` and must be freed by the application with `XtFree()`.

XiDataField

Application Header File	Xi/DataF.h
Class Header File	Xi/DataFP.h
Class Name	XiDataField
Class Pointer	xiDataFieldWidgetClass
Superclass Name	XmTextField
Creation Routine	XiCreateDataField

The DataField widget is a Data Presentation widget that handles display and entry of data as text. The DataField widget is a subclass of the Motif XmTextField widget intended for data entry applications. In addition to all of the normal XmTextField functionality, it supports regular expression-based parsing and acceptance/rejection of its input through the **XmNpicture** resource, and right justification through the **XmNalignment** resource. A typical DataField widget is shown in Figure 11.



Figure 11. An XiDataField Widget with XmNpicture Set to Accept US-Style Telephone Numbers

Keyboard navigation

Typical business applications demand better keyboard traversal than provided by Motif. The DataField widget provides added capability by supporting several types of validation: a DataField widget containing an invalid value will not give up focus; the user must enter a correct value before proceeding to another field.

Classes and Inherited Resources

DataField inherits behavior and resources from **Core**, **XmPrimitive**, and **XmTextField**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNalignment XmCAlignment	XmALIGNMENT_BEGINNING unsigned char	CSG
XmNautoFill XmCAutoFill	True Boolean	CSG
XmNpicture XmCPicture	NULL String	CSG
XmNpictureErrorCallback XmCCallback	NULL XtCallbackList	CSG
XmNvalidateCallback XmCCallback	NULL XtCallbackList	CSG

XmNalignment

When set to `XmALIGNMENT_END`, the widget aligns all its text with the right hand side of the input area.

XmNautoFill

When set to `True`, the widget "auto-fills" its contents when it can determine that the next character in the string must be a particular literal. For instance, the picture `"###-####"` automatically inserts a '-' character after receiving three numeric digits as input.

XmNpicture

Specifies a picture for data entry in the widget. A picture acts as a template that formats the value you enter in a field. An example would be the US Phone Number picture: `(###)###-####`. The picture is used to convert characters entered into the field to a formatted value.

Character interpretations

The following table lists and defines the characters you can use in a picture, and how the DataField widget interprets them.

Character	Definition
#	Any numeric digit
?	Case insensitive letter
&	Uppercase letter (forces lowercase to uppercase)
@	Case insensitive character
!	Uppercase character
;	Interpret the following character literally
*	Repeat the following character some number of times
[]	Characters within brackets are optional
{ }	Characters within braces are grouped
,	Alternative values

Other characters are interpreted literally.

Set **XmNpicture** to NULL to disable regular expression processing. The DataField widget is cleared whenever the **XmNpicture** resource is changed.

XmNpictureErrorCallback

Specifies a list of callbacks to be called when the XiDataField widget determines that data is being entered that does not match the format specified by the **XmNpicture** resource.

XmNvalidateCallback

Specifies a list of callbacks to be called when data has been entered in the XiDataField widget and the user has moved out of the XiDataField widget (usually by pressing the Tab key). The callbacks can reject the movement of focus.

Callback Routines

A pointer to the following structure is passed to the list of routines for the `XmNvalidateCallback`:

```
typedef struct _XiDataFieldCallbackStruct {
    Widget    w;          /* The XiDataField */
    String    text;      /* Proposed string */
    Boolean    accept;    /* Accept return value, for validation */
}XiDataFieldCallbackStruct;
```

The initial value of the `accept` member is `True`. To have the data accepted for entry in the `XiDataField`, leave this value `True`. To reject the data, set the field to `False`.

XiExtended18List

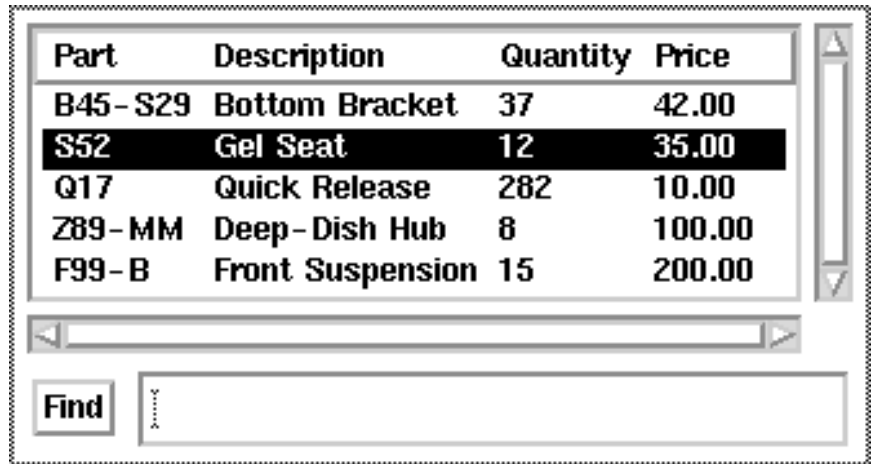
UNIX Application Header File	<code>Xi/Ext18List.h</code>
UNIX Class Header File	<code>Xi/Ext18ListP.h</code>
Class Name	<code>XiExtended18List</code>
Class Pointer	<code>xiExt18ListWidgetClass</code>
Superclass Name	<code>XmManager</code>
Creation Routine	<code>XiCreateExtended18List</code>

The Extended Internationalized List (`Extended18List`) widget contains a multi-column list with headers along the top and a search area along the bottom. All elements in the list are `XmStrings`. The list has scrollbars along the right and bottom edges that allow vertical and horizontal scrolling. Horizontal scrolling can be done by pixel (using the slider) or by column (by clicking on the arrow buttons in the scrollbar), while vertical scrolling is always by row.

The portion of the list data that is currently visible can be altered by scrollbar actions, setting resources, and the redisplay of the list data after a string search has been successful. Sorting elements within a particular column is also supported. To sort the list by the elements in a given column, the user selects the column's title with pointer button one.

To search for a particular string in the list, the user types the string value to be searched for in the list's associated text field and then presses the *Find* pushbutton. The search for the string begins in the currently selected row, after the location of the previously searched for string, or at the first column and first row if there is no column selected. If the string is not found in that row, then the search continues through all rows after, wrapping around to continue the search from row 0. If the string is found, the display of the list is adjusted to make the string visible. If the string is not found, or if the string is visible, the application issues a warning beep. The Find button and text field do not have to be displayed. The **`XmNshowFind`** resource controls whether the Find button and text field are managed or not.

Figure 12 shows a typical `Extended18List`:



Part	Description	Quantity	Price
B45-S29	Bottom Bracket	37	42.00
S52	Gel Seat	12	35.00
Q17	Quick Release	282	10.00
Z89-MM	Deep-Dish Hub	8	100.00
F99-B	Front Suspension	15	200.00

Find

Figure 12. Extended18List Widget

The callbacks on the `XmNdoubleClickCallback` list are called when the user double clicks pointer button one on a row in the list.

The list data can display a pixmap to the left of the row.

The `Extended18List` supports using multiple fonts in the `fontList` resource for the `XmString` data. `XmStrings` with newlines are also supported. The height of the title row is the maximum height of all of the column titles. If the list of column titles is `NULL`, no space is calculated for it in the layout.

Titles are justified against the top of the title row. The height of all the data rows in the list is equal to the height of the tallest string being displayed in the list data. Each string drawn in the data list is centered within that row.

Using the Resource Database

The Extended18List widget is actually a collection of pieces. It provides the geometry layout for the collection as well as tying together the pieces to form a consistent package. Many of the resources that are documented as being part of the Extended18List widget are really part of the internal list sub-component. The Extended18List widget passes these values through to the proper child when they are set at time of creation or with `XtSetValues()` or `XtGetValues()`. However, when setting a resource by using the resource database, you must use either the name of the child or the general specification (*) rather than the specific one (.).

Classes and Inherited Resources

The Extended18List widget inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNalignment XmCAlignment	XmALIGNMENT_BEGINNING unsigned char	CSG
XmNcolumnTitles XmCColumnTitles	NULL XmStringTable	CSG
XmNdoubleClickCallback XmCCallback	NULL XtCallbackList	CSG
XmNentryData XmCEntryData	NULL Xi18RowInfo	CSG
XmNfindLabel XmCFindLabel	“Find” XmString	CSG
XmNfirstColumn XmCFirstLocation	0 short	CSG
XmNfirstColumnPixmaps XmCPixmaps	False Boolean	CSG

Name (continued) Class	Default Type	Access
XmNfirstRow XmCFirstLocation	0 short	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNhorizontalScrollBar XmCScrollBar	(computed) Widget	G
XmNitemFoundCallback XmCCallback	NULL XtCallbacklist	CSG
XmNitemNotFoundCallback XmCCallback	NULL XtCallbackList	CSG
XmNnewVisualStyle XmCNewVisualStyle	True Boolean	CG
XmNnumColumns XmCNumColumns	0 short	CSG
XmNnumRows XmCNumRows	0 short	CSG
XmNselectCallback XmCCallback	NULL XtCallbackList	CSG
XmNselectedColumn XmCSelectedColumn	0 short	CSG
XmNselectionPolicy XmCSelectionPolicy	XmEXTENDED_SELECT unsigned char	CSG
XmNshowFind XmCShowFind	True Boolean	CSG
XmNsortFunctions XmCFunction	NULL XtSortFunction*	CSG
XmNstringDirection XmCStringDirection	XmSTRING_DIRECTION_L_TO_R unsigned char	CSG
XmNtitle XmCTitle	NULL XmString	CSG
XmNtitleString XmCTitleString	NULL XmString	CSG
XmNverticalScrollBar XmCScrollBar	(computed) Widget	G
XmNvisibleItemCount XmCVisibleItemCount	(computed) int	CSG

XmNalignment

Specifies how the list members are aligned within their respective columns. Values for this resource are: `XmALIGNMENT_BEGINNING`, `XmALIGNMENT_CENTER`, or `XmALIGNMENT_END`.

XmNcolumnTitles

Specifies an array of length `XmNnumColumns` of compound strings displayed at the top of each column. The data is allocated and maintained by the client.

XmNdoubleClickCallback

Specifies the list of routines called whenever the user double clicks on a row in the `Extended18List`.

XmNentryData

Specifies the data associated with each row in the list. The data is an array of `Xi18RowInfo` structures of length `XmNnumRows` allocated by the client. The data is allocated and maintained by the client. The `Xi18RowInfo` structure is defined below. When the programmer sets the `XmNfirstRow` and `XmNfirstColumn` resources, `XmNentryData` should also be updated.

XmNfindLabel

Allows the client to specify the label that is placed on the Find button. The default label is the string "Find" displayed with the default font.

XmNfirstColumn

Allows the client to adjust the current view of the list data to have a new top left column location. When setting this resource, `XmNfirstRow` should also be updated.

XmNfirstColumnPixmaps

Specifies whether the first column appears as a pixmap or as a string; that is, whether the pixmap stored in the row info structure should be used instead of the first string in the list `Xi18RowInfo values[0]`. If this resource is `True`, `values[0]` is never referenced. If it is `False`, the `Xi18RowInfo` data pixmap is never referenced.

XmNfirstRow

Allows the client to adjust the current view of the list data to have a new top left row location. When setting this resource, `XmNfirstColumn` should also be updated.

XmNfontList

Specifies an OSF/Motif™ font list used for all strings displayed in the Extended18List widget. Multiple fonts are supported.

XmNhorizontalScrollBar**XmNverticalScrollBar**

Specifies the widget IDs of the XmScrollBar widgets used by the list. These widgets are created by the list and should not be set by the user.

XmNitemFoundCallback**XmNitemNotFoundCallback**

These callbacks are called in response to a user-initiated search of the list data using the built-in find button. They pass a pointer to a XiExt18ListCallbackStruct as their `call_data` to indicate which item was found (or NULL in the case of **XmNitemNotFoundCallback**), the string that was searched for, and the event that triggered the PushButton's activate callback.

XmNnewVisualStyle

In EnhancementPak 3.0, the visual style of the Extended18List widget has changed to be more in keeping with the style of other popular graphical toolkits. The old style is still available by setting this resource to False.

XmNnumColumns**XmNnumRows**

Specifies the number of columns and rows the widget expects to display. These resources are used as the maximum indices for many of the other resources in this widget. Care should be taken when modifying these resources to ensure that the other values are also modified.

XmNselectCallback

Specifies the list of callback functions to call whenever the user clicks on a line in the Extended18List, for both values of **XmNselectionPolicy**. If the Extended18List has an **XmNselectionPolicy** of XmSINGLE_SELECT, then the `call_data` parameter to the callback is a pointer to the Xi18RowInfo structure corresponding to the single line selected; if the list is in XmEXTENDED_SELECT mode, the `call_data` parameter is undefined, and the callback code should make a call to XiExt18ListGetSelectedRows() to determine the rows which are selected. The resource name **XmNsingleSelectionCallback** is preserved for backwards compatibility.

XmNselectedColumn

Specifies the index, beginning with 0, of the currently selected column. This is also the column by which the list is being sorted.

XmNselectionPolicy

Specifies the interpretation of the select action. This resource can have the values `XmSINGLE_SELECT` or `XmEXTENDED_SELECT`. Other values result in undefined behavior.

XmNshowFind

Specifies whether or not the *Find* button and text field are displayed. If this resource is `False`, the list uses this space for displaying row data.

XmNsortFunctions

Specifies a list of functions—one for each column—called to determine the ordering of the rows in the column. The format is similar to that of `qsort()`, and is described in “*Sort Function*” on page 54.

XmNstringDirection

Specifies the initial direction to draw the string. The values for this resource are `XmSTRING_DIRECTION_L_TO_R` and `XmSTRING_DIRECTION_R_TO_L`. The value of this resource is determined at creation time. If the widget’s parent is a manager, this value is inherited from the widget’s parent, otherwise it is set to `XmSTRING_DIRECTION_L_TO_R`.

XmNtitle

Note: This resource has been replaced by the **XmNtitleString** resource. **XmNtitle** is retained only for backwards compatibility.

Specifies the title that is displayed at the top of the `Extended18List` widget. If this resource is `NULL` (the default), no title is displayed and the list’s column titles are positioned at the very top of the `Extended18List` widget.

XmNtitleString

Specifies the title that is displayed at the top of the `Extended18List` widget. If this resource is `NULL` (the default), no title is displayed and the list’s column titles are positioned at the very top of the `Extended18List` widget.

This is a renaming of the existing **XmNtitle** resource.

XmNvisibleItemCount

Specifies the number of rows visible in the `Extended18List`.

Translations and Actions

The following table lists the default translation bindings used by the icon button:

~Ctrl ~Shift <Btn1Down>:	ButtonDown()
Ctrl ~Shift <Btn1Down>:	ButtonDown(Toggle)
~Ctrl Shift <Btn1Down>:	ButtonDown(Extend)
Button1 <Motion>:	Motion()
<Btn1Up>:	ButtonUpOrLeave()

The following actions are supported by the Extended18List widget:

ButtonDown(type)

Processes a button press action that can begin with either a select or a double click. The *type* argument can be either Toggle or Extend. These values determine which mode of an extended select will be initiated on this button event. Consult the OSF/Motif™ *Style Guide* for details.

Motion()

Processes motion events to allow the selection region to be modified when in extended selection mode. It is assumed that this action is called between a ButtonDown() and ButtonUpOrLeave() action.

ButtonUpOrLeave()

Cleans up after ButtonDown() and Motion().

Compound Widget Hierarchy

The Extended18List widget is composed of several sub-widgets. Most resource values that are passed to the Extended18List widget through the argument list at creation time or via XtSetValues() are then passed to each of the widget's children. An XtGetValues() request for a child widget's resource value must be made explicitly on the child. For more information on passing arguments to the EnhancementPak compound widgets and retrieving the widget id's of the child widgets, refer to “*Compound Widgets*” on page 12.

Refer to the *OSF/Motif Programmer's Reference* for the list of any child widget's resources.

XiExtended18List	<named by application>
XmLabel	title
Xi18List	list
XmScrollbar	vertBar
XmScrollBar	horizBar
XmFrame	frame
XmPushButton	find
XmText	findText

XiExt18ListCallbackStruct Structure

The XiExt18ListCallbackStruct structure is passed as `call_data` to the **XmNitemFoundCallback** and **XmNitemNotFoundCallback** callbacks in response to user-initiated find commands.

```
typedef struct _Xi18ExtListCallbackStruct{int reason;
                                         XEvent *event;
                                         String string;
                                         int column;
                                         Xi18RowInfo *row;
} XiExt18ListCallbackStruct;
```

<i>reason</i>	Either XiEXT18LIST_FOUND or XiEXT18LIST_NOT_FOUND depending on which callback has been invoked.
<i>event</i>	X event associated with XmPushButton activateCallback from Find button.
<i>string</i>	String for which the user searched.
<i>column</i>	Column index into row values.
<i>row</i>	Xi18RowInfo structure of the matching row.

Xi18RowInfo Structure

The Xi18RowInfo structure contains the information associated with each row in the Extended18List.

```
typedef struct _Xi18RowInfo{XmString * values;
                           Pixmap pixmap;
                           Boolean selected;
                           short * sort_id;
                           XtPointer data;
                           /*
* Private to the Extended18List widget (do not modify these)
*/
                           short pix_width;
                           short pix_height;
                           short pix_depth;
                           Boolean old_sel_state;
} Xi18RowInfo;
```

<i>values</i>	An array of Motif compound strings of length XmNnumColumns which represents the strings displayed in each column of this row. The data is allocated and maintained by the client. If XmNfirstColumnPixmaps is True, then value[0] is never referenced.
<i>pixmap</i>	The pixmap displayed to the left of this row. If the resource XmNfirstColumnPixmaps is False, then this value is never referenced and can remain unset. If no pixmap is desired for this row, even though XmNfirstColumnPixmaps is True, set the value of pixmap to None. The pixmap can be of depth one (that is, a bitmap) or of the same depth as the visual. The user is responsible for creating and destroying pixmap memory.
<i>sort_id</i>	This is provided for the convenience of the client and is expected to be used as a sort index for this row. One value should be specified for each column of the row. See “ <i>Sort Function</i> ” on page 54 for details.
<i>data</i>	This is provided for the convenience of the client and can be used for any purpose. It is intended to be used as an identifier for the object pointed to by this row.
<i>selected</i>	This value is True if this row is selected; can be set by the application.

The Extended18List widget does not use *sort_id* or *data*; they exist solely for your convenience.

Callback Routine

All procedures on the Extended18List's **XmNsingleSelectionCallback** and **XmNdoubleClickCallback** lists have a pointer to a `Xi18RowInfo` structure passed to them in the `call_data` field. This structure is defined above. If an **XmNsingleSelectionCallback** is registered on an Extended18List in *extended_select_mode*, the value of `call_data` is undefined.

Sort Function

```
typedef int (Xi18SortFunction) (short column,
                               Xi18RowInfo * row1,
                               Xi18RowInfo * row2);
```

column The column currently being sorted.

row1, row2 The two rows being compared. The return value is an integer less than, equal to, or greater than 0, depending on whether the first argument is less than, equal to, or greater than the second.

Convenience Routines

XiExt18ListDeselectItems

Sets the selection state by matching column entries to `XmString`.

```
XiExt18ListDeselectItems (Widget w,
                          XmString item,
                          int column)
```

w The Extended18List widget.

item `XmString` to use as selection key.

column Column number (0 - N) to match (or `XiANY_COLUMN`).

XiExt18ListDeselectRow

Clears the selection state on a specific row.

```
XiExt18ListDeselectRow (Widget w, int row)
```

w The Extended18List widget.

row The row to select.

XiExt18ListGetSelectedRowArray

Takes Extended18List and returns NULL-terminated array of pointers to selected

rows from the internal list.

```
int * XiExt18ListGetSelectedRowArray (Widget w, int *num_rows)
```

w The Extended18List widget.
num_rows Pointer to the number of rows.

XiExt18ListGetSelectedRows()

Returns a NULL-terminated array of Xi18RowInfo pointers. Responsible for freeing the returned pointer with Xtfree(). Returns NULL if no elements are selected.

```
Xi18RowInfo ** XiExt18ListGetSelectedRows (Widget w)
```

w The Extended18List widget.

XiExt18ListMakeRowVisible

Scrolls the Extended18List to make the specified row visible.

```
XiExt18ListMakeRowVisible (Widget w, int row)
```

w The Extended18List widget.
row The row number wished to be made visible.

XiExt18ListSelectAllItems

Sets the selection state on all rows.

```
XiExt18ListSelectAllItems (Widget w, Boolean notify)
```

w The Extended18List widget.
notify If True, call XmNsingleSelectionCallback for each item in list.

XiExt18ListSelectItems

Sets the selection state by matching column entries to XmString.

```
XiExt18ListSelectItems (Widget w,  
                        XmString item,  
                        int column,  
                        Boolean notify)
```

w Extended18List widget.
item XmString to use as selection key.
column Column number (0 - N) to match (or XiANY_COLUMN).
notify If True, call **XmNsingleSelectionCallback**.

XiExt18ListSelectRow

Sets the selection state on a specific row.

```
XiExt18ListSelectRow (Widget w, int row, Boolean notify)
```

w Extended18List widget.
row Row to select.
notify If True, call **XmNsingleSelectionCallback**.

XiExt18ListUnselectAllItems()

Unselects all selected rows of the passed widget.

```
XiExt18ListUnselectAllItems (Widget w)
```

w Widget ID of the widget in which rows are to be unselected.

XiExt18ListUnselectItems()

Unselects the specified row of the passed widget.

```
XiExt18ListUnselectItems (Widget w, Xi18RowInfo *row)
```

w Widget ID of the widget in which a row is to be unselected.
row Row to be unselected.

XiExt18ListToggleRow()

Toggles the selection state of a specified row.

```
XiExt18ListToggleRow (Widget w, short row)
```

w Widget ID of the Extended18List widget.
row Specified row.

XiFontSelector

UNIX Application Header File	Xi/FontS.h
UNIX Class Header File	Xi/FontSP.h
Class Name	XiFontSelector
Class Pointer	xiFontSelectorWidgetClass
Superclass Name	XiPaned
Creation Routine	XiCreateFontSelector

The FontSelector widget allows users to choose a font easily by selecting font family, size, weight, slant, and string direction (left to right or right to left). Any font can be passed by the application to the FontSelector as the initial value displayed. Several advanced features greatly extend the widget's capabilities.

Basic Features

The FontSelector widget is displayed as two combination box widgets: one with a list of choices for the font family, and the other with the choices for the font size. In addition, two independent toggle buttons allow the user to make the font bold or italic or both. Below the font choice area is a text widget that displays sample text in the chosen font. This text area is editable, allowing the user to add or remove text to preview how various characters appear. For the novice user, this set of features allows access to all standard fonts on the system. The FontSelector dynamically removes choices that are inappropriate, so the user is free to experiment with different combinations and is assured that a selected font will exist on the machine on which the FontSelector is running.

For example, if the Times Roman 14 point font is unavailable and the user selects a point size of 14, then Times Roman will not be available in the *Family* combination box. To obtain all available choices, choose a *Size* and *Family* of **Any**.

Figure 13 shows a FontSelector widget:

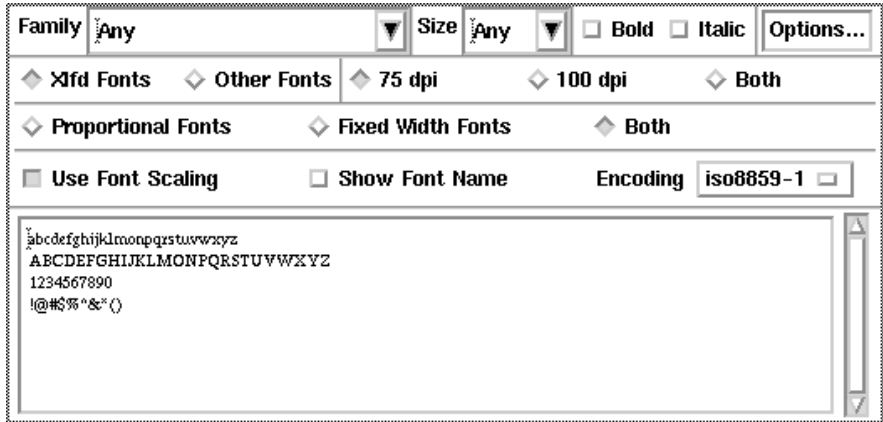


Figure 13. *FontSelector* Widget

Advanced Features

For the advanced user, the FontSelector provides tremendous flexibility in XLFD (X Logical Font Description) choices. Clicking the *Options* button displays an additional panel of controls. This allows access to non-XLFD fonts, control of the resolutions of the fonts chosen, choice of fixed or proportional fonts only, removal or use of font scaling, selection of different font encodings, and dynamic display of the XLFD name of the font which the FontSelector is constructing.

Non XLFD Fonts

By choosing the “Other Fonts” toggle from the option panel, the *Family* and *Size* lists, as well as the *Bold* and *Italic* toggles, are replaced with a combination box containing all non-XLFD fonts available on your system. This feature allows users to select non-XLFD fonts with the FontSelector. A string entered in the text field of the combination box is interpreted as a font name. You can also enter XLFD names manually. The combination of this feature and the other FontSelector options allow you to display *any* font on the entire system.

Resolution Control

The FontSelector selects as its default the standard resolution that is closest to the current display. Choose a font of a different resolution or display both resolutions to allow a wider range of choices.

Fixed or Proportional

In most cases, the fact that a font is fixed width or proportional is unimportant to the user. However, some applications require a fixed width font, such as terminal emulators, and many users prefer proportional fonts for appearance purposes. The FontSelector allows users to limit the font choices to fixed width or proportional or to allow both.

Font Scaling

The font scaling technology that is available in X11R5 uses bitmap scaling which, although useful in some cases, generally results in very ugly fonts. We noticed that users often wanted to know which fonts are scaled and which ones exist as hand crafted bitmaps. To remove the scaled fonts from the list of choices, toggle the “Use Font Scaling” button off. This value is resource controllable and defaults to on.

Encoding

The programmer can specify which encodings are valid. These encoding choices appear in an option menu. The list of font choices is restricted to those that use the current selected coding.

XLFD Name Display

Clicking the *Show* toggle displays the current font’s XLFD name at the bottom of the FontSelector.

Classes and Inherited Resources

FontSelector inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XiPaned**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmN100DPIStrig XmC100DPIStrig	“100 dpi” XmString	CSG
XmN75DPIStrig XmC75DPIStrig	“75 dpi” XmString	CSG
XmNanyLowerString XmCAnyLowerString	“any” XmString	SCG
XmNanyString XmCAnyString	“Any” XmString	CSG
XmNboldString XmCBoldString	“Bold” XmString	CSG
XmNbothString XmCBothString	“Both” XmString	CSG
XmNcurrentFont XmCString	NULL String	CSG
XmNdefaultEncodingString XmCDefaultEncodingString	“iso8859-1” String	CSG
XmNencodingList XmCEncodingList	“iso8859-1” StringTable	CSG
XmNencodingString XmCEncodingString	“Encoding” XmString	CSG
XmNfamilyString XmCBothString	“Family” XmString	CSG
XmNitalicString XmCItalicString	“Italic” XmString	CSG
XmNmarginHeight XmCMargin	0 Dimension	CSG

Name (continued) Class	Default Type	Access
XmNmonoSpaceString XmCMonoSpaceString	“Fixed Width Fonts” XmString	CSG
XmNoptionString XmCOptionString	“Options...” XmString	CSG
XmNotherString XmCOtherString	“Other Fonts” XmString	CSG
XmNpropSpaceString XmCPropSpaceString	“Proportional Fonts” XmString	CSG
XmNsampleText XmCSampleText	“abcdef...” XmString	CSG
XmNscalingString XmCScalingString	“Use Font Scaling” XmString	CSG
XmNshowFontName XmCBoolean	False Boolean	CSG
XmNshowNameString XmCShowNameString	“Show Font Name” XmString	CSG
XmNsizeString XmCSizeString	“Size” XmString	CSG
XmNspacing XmCSpacing	2 Dimension	CSG
XmNtextRows XmCTextRows	8 Dimension	CSG
XmNuseScaling XmCBoolean	True Boolean	CSG
XmNvalueChangedCallback XmCCallback	NULL XtCallbackList	CSG
XmNxlfidString XmCXlfidSpaceString	“Xlfd Fonts” XmString	CSG

XmN100DPIstring

Specifies the label for the 100 DPI radio button.

XmN75DPIstring

Specifies the label for the 75 DPI radio button.

XmNanyLowerString

Specifies the label for the *Any* button.

XmNanyString

Specifies the label for the *Any* button.

XmNboldString

Specifies the label for the *Bold* toggle button.

XmNbothString

Specifies the labels for the *Both* radio buttons controlling both the dpi and width of the fonts displayed. The same resource is used to ensure consistent labels.

XmNcurrentFont

Provides the main application input and output to the FontSelector. If the programmer sets the value at creation time or with XtSetValues(), the currently displayed family, size, bold and italic are changed to correspond to the values shown in the current font. Otherwise, the name of the font is shown. The FontSelector's mode is set to correspond to the type of font passed.

Note: currentFont must contain 14 hyphens (-) to be considered an XLFD font. This resource is also used to retrieve the font the user has selected from the FontSelector. The value returned is only valid until the next time XtGetValues is called on this instance of the FontSelector widget.

XmNdefaultEncodingString

Specifies the default selection from the *Encoding* options menu.

XmNencodingList

Specifies the list of encodings available from the FontSelector *Encoding* options menu.

XmNencodingString

Specifies the label for the *Encoding* options menu.

XmNfamilyString

Specifies the label for the *Family* options menu.

XmNitalicString

Specifies the label for the *Italic* toggle button.

XmNmarginHeight

Specifies the margin height of the FontSelector.

XmNmonoSpaceString

Specifies the label of the *Fixed Width Fonts* radio button.

XmNoptionString

Specifies the label for the *Options...* pushbutton.

XmNotherString

Specifies the label for the *Other Fonts* radio button.

XmNpropSpaceString

Specifies the label for the *Proportional Fonts* radio button.

XmNsampleText

Specifies the string which appears in the sample text area.

XmNscalingString

Specifies the label for the *Use Font Scaling* toggle button.

XmNshowFontName

Controls and maintains the state of the *Show Font Name* toggle button.

XmNshowNameString

Specifies the label of the *Show Font Name* toggle button.

XmNsizeString

Specifies the label for the *Size* option menu.

XmNspacing

Specifies the space between the toggle indicator and the toggle label.

XmNtextRows

Specifies the number of rows shown in the text widget that displays sample text in the currently selected font. Since this is a scrolled text widget, it will never dynamically resize, regardless of the font displayed. A value of at least 4 optimizes ease of use.

XmNuseScaling

Specifies the state of the *Use Font Scaling* toggle button.

XmNvalueChangedCallback

Specifies the list of callbacks called when the **XmNcurrentFont** value is changed.

XmNxlfString

Specifies the label for the *Xlfd Fonts* radio button.

Compound Widget Hierarchy

The FontSelector is composed of several sub-widgets. Most resource values that are passed to the FontSelector through the argument list at creation time or by XtSetValues() are then passed to each of the widget's children. An XtGetValues() request for a child widget's resource value must be made explicitly on the child. For more information on passing arguments to the EnhancementPak compound widgets and retrieving the widget ids of the child widgets, refer to "Compound Widgets" on page 12.

Consult the *OSF/Motif Programmer's Reference* for the list of any child widget's resources.

XiFontSelector	<named by application>
XiPaned	topPane
XiCombinationBox	families
<refer to Combination Box for hierarchy>	
XmSeparator	separator
XiCombinationBox	sizes
<refer to Combination Box for hierarchy>	
XmSeparator	separator
XiButtonBox	boldItalicBox
XmToggleButton	boldButton
XmToggleButton	italicButton
XmSeparator	separator
XmToggleButton	optionButton
XmSeparator	separator
XiPaned	middlePane
XiPaned	leftPane
XiButtonBox	choiceBox
XmToggleButton	xlfdButton
XmToggleButton	otherButton
XmSeparator	separator
XiButtonBox	resolutionBox
XmToggleButton	dpi75Button

XmToggleButton	dpi100Button
XmToggleButton	anyButton
XmSeparator	separator
XmSeparator	separator
XiButtonBox	spacingBox
XmToggleButton	proportionalButton
XmToggleButton	monoButton
XmToggleButton	bothButton
XmSeparator	separator
XiButtonBox	otherChoiceBox
XmToggleButton	scalingButton
XmToggleButton	showNameButton
XmRowColumn	encodingOptionMenu
XmLabelGadget	OptionLabel
XmCascadeButtonGadget	OptionButton
XmMenuShell	menuShell
XmRowColumn	pulldownMenu
<dependent on XmNencoding>	
XmSeparator	separator
XmSeparator	separator
XiButtonBox	box
XmScrolledWindow	textSW
XmScrollBar	VertScrollBar
XmText	text
XmSeparator	separator
XmLabel	nameLabel
XmSeparator	separator

XiHierarchy

UNIX Application Header File	<code>Xi/Hierarchy.h</code>
UNIX Class Header File	<code>Xi/HierarchyP.h</code>
Class Name	<code>XiHierarchy</code>
Class Pointer	<code>xiHierarchyWidgetClass</code>
Superclass Name	<code>XmManager</code>
Creation Routine	Not applicable

The Hierarchy widget is not instantiated by itself, but should be used as the base class for any widget that displays a hierarchy of information. (This refers to the information displayed, not to the hierarchy of objects in a compound widget.) This base class is used for the `XiTree` and `XiOutline` widgets, providing them with very similar APIs.

Note: The Hierarchy widget assumes it will be responsible for mapping and unmapping its children. Therefore, no child of Hierarchy should ever modify its **XmNmappedWhenManaged** resource. If a child does modify this resource, the behavior is undefined.

Classes and Inherited Resources

Hierarchy inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNautoClose XmCAutoClose	True Boolean	CSG
XmNcloseFolderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNhorizontalMargin XmCDimension	2 Dimension	CSG
XmNnodeStateBeginEndCallback XmCCallback	NULL XtCallbackList	CSG
XmNnodeStateCallback XmCNodeStateCallback	NULL XtCallbackList	CSG
XmNnodeStateChangedCallback XmCNodeStateChangedCallback	NULL XtCallbackList	CSG
XmNopenFolderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNrefigureMode XmCBoolean	True Boolean	CSG
XmNverticalMargin XmCDimension	2 Dimension	CSG

XmNautoClose

Specifies whether the Hierarchy should automatically restore a parent node's children when the parent node is reopened. If **XmNautoClose** is False, and the Hierarchy is fully expanded when the root node is closed, the entire Hierarchy is displayed when the root node is reopened. If **XmNautoClose** is True, the root node's children are closed when the root node is reopened.

XmNcloseFolderPixmap

XmNopenFolderPixmap

Specifies the pixmaps displayed in all folder button widgets that are associated with nodes with a state of XiClosed and XiOpen for nodes that specify no **XmNnodeCloseFolderPixmap** or **XmNnodeOpenFolderPixmap**, respectively. If the value of **XmNopenFolderPixmap** is set to XmUNSPECIFIED_PIXMAP (either at creation or via XtSetValues()), a default open-folder bitmap or color pixmap is displayed. Behavior is analogous for **XmNcloseFolderPixmap**.

XmNhorizontalMargin**XmNverticalMargin**

The definitions of these resources are left to the subclass of the Hierarchy widget that does the geometry layout. They are intended to be used as the number of pixels between the object and the edges of the window in which it is contained. They are included here for consistency.

XmNnodeStateBeginEndCallback

This callback is invoked at the beginning and end of a group of node state changes (as for the closing of an entire tree, for instance). The `call_data` is a Boolean type. True indicates that this is the beginning of a state change, False indicates that the changes have finished.

XmNnodeStateCallback

Specifies the list of callback routines called when a folder button is clicked. See “*Callback Routine*” on page 69 for more details.

XmNnodeStateChangedCallback

This callback list is invoked when a node's state (`XiOpen` or `XiClosed`) has changed. The callbacks are passed an `XiHierarchyNodeStateData*` as the `call_data` containing the information on the node whose state has changed.

XmNrefigureMode

Specifies whether the Hierarchy should adjust the sizes of the children after a geometry or resize request, or simply ignore the request. This resource is very useful in improving the performance of an application that is making a large number of geometry changes all at once.

Constraint Resources

Name Class	Default Type	Access
XmNinsertBefore XmCinsertBefore	NULL Widget	CSG
XmNnodeCloseFolderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNnodeOpenFolderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNnodeState XmCNodeState	XiOpen XiHierarchyNodeState	CSG
XmNparentNode XmCparentNode	NULL Widget	CSG

XmNinsertBefore

Places the current node immediately before another node in the Hierarchy that has the same **XmNparentNode** value. If this value is NULL, the current node is inserted at the end of the list. This resource allows the Hierarchy's children to be reordered.

XmNnodeCloseFolderPixmap

Specifies the pixmap to be used when the node is in state XiClosed. The default is the parent widget's value for **XmNcloseFolderPixmap**.

XmNnodeOpenFolderPixmap

Specifies the pixmap to be used when the node is in state XiOpen. The default is the parent widget's value for **XmNopenFolderPixmap**.

XmNnodeState

Specifies the state of the current node. Acceptable values are: XiOpen, XiClosed, XiAlwaysOpen, and XiHidden. A type converter has been registered that can convert the following strings: "open", "closed", "alwaysOpen", and "hidden".

XmNparentNode

Specifies the parent of the current node. The parent node *must* be a widget sibling of the current node. If **XmNparentNode** is set to NULL, the node is placed on the screen as root. No node may have multiple parents.

Callback Routine

When a folder is clicked, the routines registered on the **XmNnodeStateCallback** list are

passed a pointer to the following structure as client data:

```
typedef struct _XiHierarchyNodeStateData {
    Widget widget;
    XiHierarchyNodeState state;
} XiHierarchyNodeStateData;
```

widget The child node of Hierarchy being open or closed.

state The current **XmNnodeState** (after the click) of this node. Legal values are XiOpen, XiClosed, XiAlwaysOpen, and XiHidden.

Convenience Routines

XiHierarchyGetChildNodes()

Returns a list of widgets that name the node widget as their **XmNparentNode**; that is, it returns the list of "node children" of the node widget. Note that all the widgets are children (in the Xt sense) of the XiHierarchy or its subclass. The returned list is NULL if there are no node children. The array of widgets returned is NULL-terminated. It should be freed with XtFree().

```
WidgetList XiHierarchyGetChildNodes (Widget nw)
```

nw The node widget whose list of node children is requested

XiHierarchyOpenAllAncestors()

Opens all ancestors of a given node, so that the node is displayed.

```
void XiHierarchyOpenAllAncestors (Widget nw)
```

nw The node widget that wishes to be shown.

Class Methods

To aid subclassing, a description of each of the new class methods is provided here. ICS strongly suggests purchasing the source code of the Hierarchy widget if you intend to subclass.

Change Node State Routine

Updates the Hierarchy when the state of a node has changed. This routine is called from `ConstraintSetValues` on the state variable, as well as from the `toggle_node_state` routine. This method can be inherited with `XiInheritChangeNodeState`.

```
void (change_node_state) (HierarchyConstraints node)
    node                    The node that has had its state changed.
```

Map Node Routine

Assures that a node is visible. Nodes that the Hierarchy believes are visible have the state bit `IS_MAPPED` set. The default routine does this by mapping the node. This method can be inherited with `XiInheritMapNode`.

```
void (map_node) (HierarchyConstraints node)
    node                    The node to be mapped.
```

Unmap Node Routine

Assures that a node is no longer visible. Nodes that the Hierarchy believes are visible have the state bit `IS_MAPPED` set. The default routine does this by unmapping the node. This method can be inherited with `XiInheritUnmapNode`.

```
void (unmap_node) (HierarchyConstraints node)
    node                    The node to be unmapped.
```

Unmap All Extra Nodes Routine

Assures that all nodes that have the state bit `IS_COMPRESSED` set are no longer visible. This is done to remove hidden nodes. This method can be inherited with `XiInheritUnmapAllExtraNodes`.

```
void (unmap_all_extra_nodes) (HierarchyConstraints node)
    node                    The node to be unmapped.
```

Build Node Table Routine

Fills the `node_table` variable in the Hierarchy widget structure with a list of all nodes that are visible. This method can be inherited with `XiInheritBuildNodeTable`.

```
void (build_node_table) (Widget w,
                        HierarchyConstraints node,
                        Cardinal * current_index)
w           Hierarchy widget.
node        Root node to begin building the table with.
current_index Value, initially set to zero, that contains the location
              to add the next node in the table, allowing this
              function to be called recursively.
```

Reset Open Closed Button Routine

Creates an `open_close_button` for a node, if it is necessary, and ensures that the image in that button corresponds to the current state of the node. This method can be inherited with `XiInheritResetOpenCloseButton`.

```
void (reset_open_close_button) (Widget w,
                                HierarchyConstraints node)
w           Hierarchy widget.
node        Node that is to have its folder button reset.
```

Toggle Node State Routine

Changes the node from open to closed and back again. This routine should be assigned to each node button in the Hierarchy. This method can be inherited with `XiInheritToggleNodeState`.

```
void (toggle_node_state) (Widget w,
                           XtPointer node_ptr,
                           XtPointer call_data)
w           Folder button selected.
node_ptr    Pointer to the node selected.
call_data   Value is undefined.
```

XiIconBox

UNIX Application Header File	<code>Xi/IconBox.h</code>
UNIX Class Header File	<code>Xi/IconBoxP.h</code>
Class Name	<code>XiIconBox</code>
Class Pointer	<code>xiIconBoxWidgetClass</code>
Superclass Name	<code>XmManager</code>
Creation Routine	<code>XiCreateIconBox</code>

The `IconBox` widget lays out its children on a grid. Each child is forced to be the same size. The location of each child is specified as an X and Y location on the grid. Figure 14 shows an example of an `IconBox` containing icon buttons:

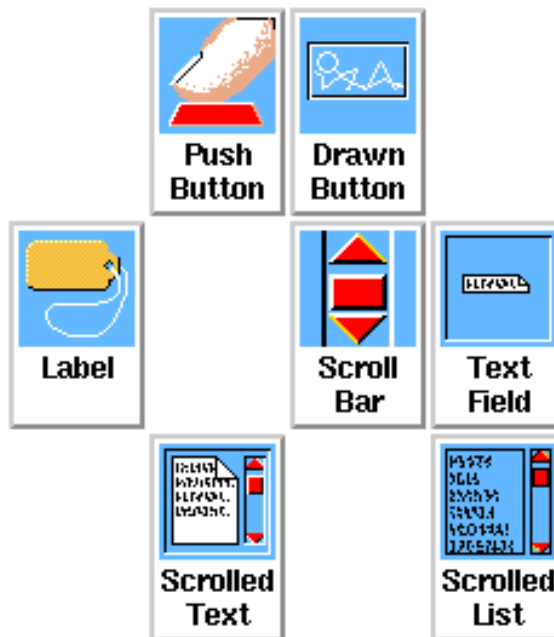


Figure 14. IconBox Widget Containing Icon Buttons

All children are always shown and should be given their desired size whenever possible. You may add or delete cells by resizing this window, using either the window manager or the Stretch widget. The size of the IconBox, its children, and the number of cells displayed are calculated as follows.

- The preferred size is calculated by using the maximum desired child height or width and making sure that these are no smaller than the minimum sizes. This size is multiplied by the number of cells along the axis and properly padded to come up with a preferred size. The number of cells is the maximum of the largest **XmNcellX** or **XmNcellY** value and the minimum number of horizontal or vertical cells.
- If the IconBox is forced larger than its preferred size, more cells are added at the bottom-right of the widget while the children all remain at their preferred sizes.
- If the IconBox is forced smaller than its preferred size, each cell is forced to be smaller in order to allow all children to fit within the IconBox. All children are forced to the same smaller size.

Classes and Inherited Resources

IconBox inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to “*Appendix*” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNhorizontalMargin XmCMargin	4 Dimension	CSG
XmNminimumHorizontalCells XmCDefaultCells	2 int	CSG
XmNminimumVerticalCells XmCDefaultCells	2 int	CSG
XmNminimumCellHeight XmCMinimumCellSize	10 Dimension	CSG
XmNminimumCellWidth XmCMinimumCellSize	20 Dimension	CSG
XmNverticalMargin XmCMargin	4 Dimension	CSG

XmNhorizontalMargin

XmNverticalMargin

Specifies the number of pixels between each cell and its neighbor or the edge of the IconBox.

XmNminimumHorizontalCells

XmNminimumVerticalCells

Specifies the minimum number of cells to display in the horizontal and vertical directions. This number of cells is always displayed.

XmNminimumCellWidth

XmNminimumCellHeight

Specifies the smallest number of pixels that the cells are allowed to occupy in the direction specified.

Constraint Resources

Name Class	Default Type	Access
XmNcellX XmCCellX	XiIconBoxAnyCell short	CSG
XmNcellY XmCCellY	XiIconBoxAnyCell short	CSG

XmNcellX

XmNcellY

Specifies the location of this cell in *cell space*. These coordinates can be any positive integer, including 0. They determine where this widget is placed relative to its neighbors. Having two children at the same **XmNcellX** and **XmNcellY** location will result in undefined behavior. To place a cell at any empty cell, set **XmNcellX** and **XmNcellY** to the value `XiIconBoxAnyCell`.

Convenience Routine

XiIconBoxIsEmpty()

Determines whether a cell in the `IconBox` is empty.

```
Boolean XiIconBoxIsEmpty(Widget w,
                          Position cell_x,
                          Position cell_y,
                          Widget ignore)
```

<i>w</i>	The <code>IconBox</code> widget.
<i>cell_x</i>	The x location of the cell to check.
<i>cell_y</i>	The y location of the cell to check.
<i>ignore</i>	If the widget id specified by <i>ignore</i> is present in the specified cell, it will be ignored and the function will return <code>True</code> .

This function returns `True` if the specified cell contains no child, and `False` otherwise.

XiIconButton

UNIX Application Header File	Xi/IconButton.h
UNIX Class Header File	Xi/IconButtonP.h
Class Name	XiIconButton
Class Pointer	xiIconButtonWidgetClass
Superclass Name	XmPrimitive
Creation Routine	XiCreateIconButton

The `IconButton` widget is a selectable area of the screen that contains both a label and an icon. When you select this button with the Motif select button, its **XmNactivateCallback** is called. This widget can also be used as a Toggle button (see page 81), although it will have no indicator. The relationship of the icon to the text can be modified using the **XmNiconPlacement** resource. The widget takes an `XmString` as its label.

Figure 15 shows `IconButton`s in all six icon displacements:



Figure 15. `IconButton`s Displayed in All Six Icon Placements

Classes and Inherited Resources

`IconButton` inherits behavior and resources from **Core** and **XmPrimitive**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNactivateCallback XmCCallback	NULL XtCallbackList	CSG
XmNalignment XmCAlignment	XmALIGNMENT_BEGINNING unsigned char	CSG
XmNarmColor XmCArmColor	dynamic Pixel	CSG
XmNdoubleClickCallback XmCCallback	NULL XtCallbackList	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNhorizontalMargin XmCSPACE	2 Dimension	CSG
XmNiconTextPadding XmCSPACE	2 Dimension	CSG
XmNiconPlacement XmCIconPlacement	XiIconTop XiIconPlacement	CSG
XmNlabel XmCLabel	NULL String	CSG
XmNlabelString XmCLabelString	dynamic XmString	CSG
XmNpixmap XmCPixmap	None Pixmap	CSG
XmNrecomputeSize XmCBoolean	True Boolean	CSG
XmNset XmCBoolean	False Boolean	CSG
XmNstringDirection XmCStringDirection	XmSTRING_DIRECTION_L_TO_R unsigned char	CSG
XmNverticalMargin XmCSPACE	2 Dimension	CSG

XmNactivateCallback

Clicking the `IconButton` calls this list of callback routines. The `call_data` for the callback is specified in “*Callback Routines*” on page 82.

XmNalignment

Specifies the label alignment for text or pixmap. `XmALIGNMENT_BEGINNING` causes a left alignment with the left edge of the widget. `XmALIGNMENT_CENTER` and `XmALIGNMENT_END` cause analogous center and right alignments, respectively.

When the **XmNstringDirection** is `XmSTRING_DIRECTION_R_TO_L`, the definitions of `XmALIGNMENT_BEGINNING` and `XmALIGNMENT_END` are reversed.

XmNarmColor

Specifies the color with which to fill the widget when armed by a pointer click.

XmNdoubleClickCallback

Double-clicking the `IconButton` calls this list of callback routines. The format of the callback routines is specified in “*Callback Routines*” on page 82.

XmNfontList

Specifies the default font in this list is used for the label string of the `IconButton`.

XmNhorizontalMargin**XmNverticalMargin**

Specifies the number of pixels to be left between the edge of the shadow and the text or pixmap displayed. The vertical and horizontal spacing can be controlled independently.

XmNiconTextPadding

Specifies the number of pixels to be left between the pixmap and the label string.

XmNiconPlacement

Specifies the location of the pixmap (icon) with respect to the displayed text. This resource can take one of the following values: `XiIconTop`, `XiIconBottom`, `XiIconLeft`, `XiIconRight`, `XiIconNone`, or `XiIconOnly`.

If only a string is displayed in the `IconButton`, this resource can be used to change the justification of the label. `XiIconTop=bottom`, `XiIconBottom=top`, `XiIconRight=left`, `XiIconLeft=right`, `XiIconNone=center`. In order to use these options, **XmNpixmap** must be set to `None`.

XmNlabel

Note: This resource has been superseded by **XmNlabelString**, and is now included for backwards-compatibility only. If **XmNlabelString** is set, **XmNlabel** is ignored.

Specifies the string to display in this button. This string can only have one font, but can be any number of lines long. Use the newline character “\n” to separate lines.

XmNlabelString

Specifies the compound string to be displayed in the button. If this value is NULL, the value of **XmNlabel** is used. If both are NULL, it is initialized by converting the name of the widget to a compound string. Refer to **XmString(3X)** in the *OSF/Motif Programmer's Reference* for more information on the creation and structure of compound strings.

XmNpixmap

Specifies the pixmap to display. This pixmap can either be of depth one (1), or the same depth as the screen where this widget is being displayed. If the pixmap is of depth one, `XCopyPlane()` is used to render the pixmap in the foreground and background colors. If the pixmap is not of depth one, `XCopyArea()` is used and all the original colors of the pixmap are preserved. Unlike the Motif `PushButton` widget, the pixmap is automatically stippled when the `IconButton` becomes insensitive.

XmNrecomputeSize

If this Boolean value is True, the `IconButton` asks its parent to resize it to be just large enough to contain the pixmap, label and shadows. If it is False, the `IconButton` does not attempt a resize.

XmNset

Specifies the Boolean value that represents the current state of the `IconButton`. If this value is True, the `IconButton` is set and is rendered as *depressed*. Otherwise it is unset and is rendered normally.

XmNstringDirection

Specifies the direction in which the string is to be drawn.

`XmNSTRING_DIRECTION_L_TO_R` is drawn left to right, while

`XmNSTRING_DIRECTION_R_TO_L` is drawn right to left.

The default for this resource is determined at creation time. If no value is specified for this resource and the widget's parent is a manager, the value is inherited from the parent; otherwise, it defaults to `XmSTRING_DIRECTION_L_TO_R`.

Translations and Actions

The following table lists the default translation bindings used by the `IconButton`:

<Btn1Down>,<Btn1Up>:	XiToggle() XiNotify() XiButtonUp()
<Btn1Down>:	XiGetFocus() XiToggle()
<Key>osfSelect:	XiArmAndActivate()
<Key>osfActivate:	XiArmAndActivate()
None<Key>space:	XiArmAndActivate()
None<Key>Return:	XiArmAndActivate()
<Btn1Down>,<Leave>	XiToggle()

The following actions are supported by the `IconButton` widget:

XiToggle()

Toggles the state of the `IconButton`.

XiNotify()

If the interval between the last `XiButtonUp()` action and this event's timestamp is less than `MultiClickTime`, calls all routines on the **XmNdoubleClickCallback** list. Otherwise, calls all routines on **XmNactivateCallback** list.

XiButtonUp()

Records the timestamp of a button-up event for use in double-click processing.

XiArmAndActivate()

Arms the button, calls the actions on the **XmNactivateCallback** list, waits a fraction of a second, and disarms the button.

To use this button as a toggle button rather than a push button, replace the default translation table with the following table.

<Btn1Down>:	XiToggle() XiNotify()
<Btn1Down>,<Btn1Up>:	XiButtonUp()
<Key>osfSelect:	XiToggle() XiNotify()
<Key>osfActivate:	XiToggle() XiNotify()
None<Key>space:	XiToggle() XiNotify()
None<Key>Return:	XiToggle() XiNotify()

Callback Routines

Clicking the `IconButton` calls the activate callbacks. Whenever the widget is double clicked, the first click will call the **XmNactivateCallback**, and the second click, if it occurs within the value returned by `XtGetMultiClickTime()`, will call the **XmNdoubleClickCallback**.

All procedures on the `IconButton`'s **XmNactivateCallback** and **XmNdoubleClickCallback** lists have a pointer to an `IconButtonCallbackInfo` structure passed to them in the `call_data` field. This structure is defined in the `IconButton` widget's public header file as follows:

```
typedef struct _XiIconButtonCallbackInfo {  
    Boolean state;  
    XEvent *event;  
} XiIconButtonCallbackInfo;
```

state Current state of the `IconButton`. When this is used as a push button (the default) the state variable is always True.

event X Event that caused this action. See “*Translations and Actions*” on page 81 for details on the events that can cause these callbacks to be called.

XiOutline

UNIX Application Header File	Xi/Outline.h
UNIX Class Header File	Xi/OutlineP.h
Class Name	XiOutline
Class Pointer	xiOutlineWidgetClass
Superclass Name	XiHierarchyWidgetClass
Creation Routine	XiCreateOutline

The Outline widget is a container that shows the relationship of its children in a Outline format. Each child of the Outline widget is a node in the Outline.

The hierarchy of nodes is created by specifying the “parent” of each node as a constraint resource. If a node’s parent is NULL, it is assumed to be a root of the Outline. Although each widget can only have one parent, the Outline widget supports adding more than one “root” node to a single Outline.

Clicking on the associated control button hides the node’s child nodes.

Refer to “*XiHierarchy*” on page 66 for further information, including the convenience routines for XiHierarchy that can be used for the XiOutline.

Figure 16 shows a typical Outline widget:

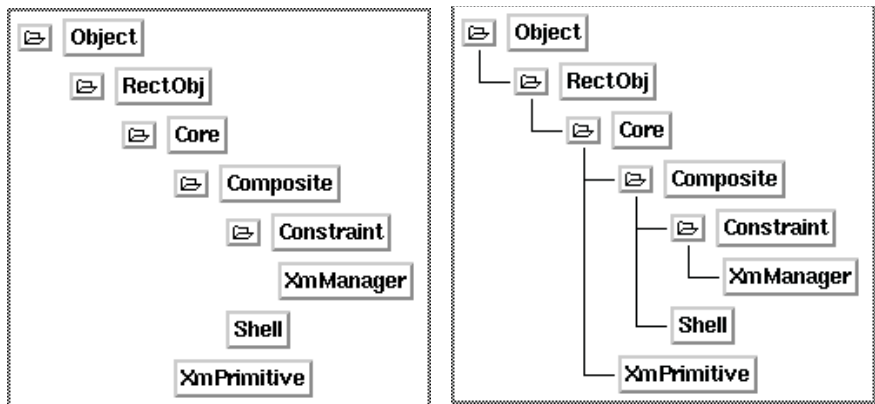


Figure 16. Outline Widgets with *XmNconnectNodes* Set to False and True

Figure 17 shows an Outline widget with some nodes closed:

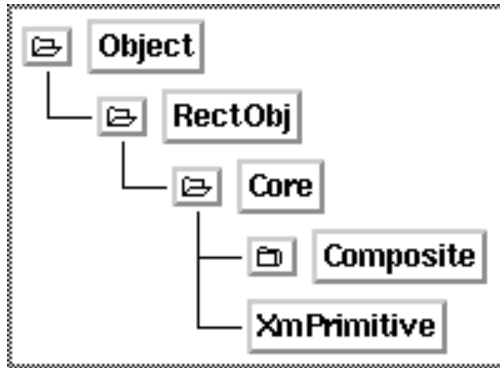


Figure 17. Outline Widget with Some Nodes Closed

The Outline widget assumes that it is totally responsible for mapping and unmapping its children. Therefore, no child of this widget should ever modify its **XmNmappedWhenManaged** resource. If a child does modify this resource, the behavior is undefined.

Outline Node Types

Each node in the Outline can have one of four values of **XmNnodeStates**: **XiOpen**, **XiClosed**, **XiAlwaysOpen**, or **XiHidden**. The appearance of the node and the actions that are available to the user vary with its state as follows:

XiOpen

This node has an open folder shown to its left; clicking on it closes the node. When a node is open, all of its children are visible.

XiClosed

This node has a closed folder shown to its left; clicking on it opens the node. When a node is closed, none of its children are visible.

XiAlwaysOpen

This node has no folder button associated with it. All of its children are visible.

Note: To maintain consistency of the user interface, it is best to use the node state `XiAlwaysOpen` for nodes with no children. This way the user sees a folder button only next to a node that has children to display. A folder button associated with a node that has no children has no defined behavior.

XiHidden

This node is not visible. All of its children appear and behave exactly as if they were children of the node's parent.

Geometry Management

The preferred size of the Outline will be just large enough to contain all nodes in the hierarchy. As the node state changes, the Outline attempts to resize itself to just contain its currently visible children. If the Outline is forced away from its desired size, the children are not moved; they are either clipped or they appear in the upper left hand corner of the window. For this reason it is usually advisable to put the Outline into a Scrolled Window or `XiPorthole` widget.

Classes and Inherited Resources

Outline inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XiHierarchy**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer's Reference*.

Resources

Name Class	Default Type	Access
XmNconnectNodes XmCBoolean	False Boolean	CSG
XmNconstrainWidth XmCConstrainWidth	False Boolean	CSG
XmNindentSpace XmCIndentSpace	30 Dimension	CSG

XmNconnectNodes

Specifies whether or not the child nodes of a node should be visually connected to it by lines drawn on the XiOutline's background.

XmNconstrainWidth

Specifies that the Outline widget should negotiate with its children to offer them the best layout when the Outline itself cannot grow to display them at their preferred sizes.

XmNindentSpace

Specifies the number of pixels by which each new level of the Outline is indented.

XiPaned

UNIX Application Header File	Xi/Paned.h
UNIX Class Header File	Xi/PanedP.h
Class Name	XiPaned
Class Pointer	xiPanedWidgetClass
Superclass Name	XmManager
Creation Routine	XiCreatePaned

The Paned widget manages children in a horizontally or vertically tiled fashion with each child in a separate pane. The panes can be dynamically resized by the user using *control sashes* that appear between the panes (Figure 18). Application programmers have control over whether or not sashes and separators are displayed, the preferred size of each pane, and which pane is forced to absorb size restrictions imposed by the Paned widget's parent.

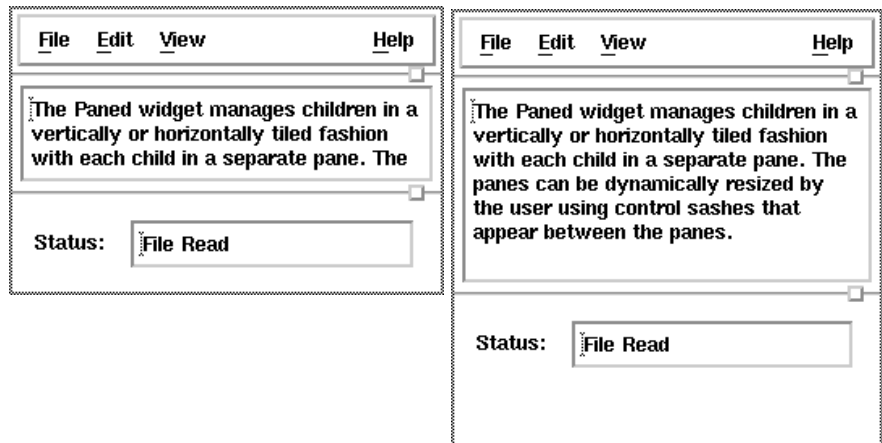


Figure 18. Paned Widget with XmOrientation Set to XmVERTICAL

Geometry Management

Sizing children

The Paned widget usually resizes its children to their preferred sizes when a new child is managed. It first attempts to resize itself to contain its panes exactly. If this is not possible, it hunts through the children from bottom to top (or right to left), looking for a pane to resize.

The Paned widget attempts to comply with the geometry request of its children. It first attempts to resize itself to satisfy the request. Next, it goes through the criteria specified in “*Resizing panes*” to satisfy the request. Only if all panes are at their minimum or maximum values is a geometry request refused. If the **B** resource is False for the child, all geometry requests are denied.

Resizing panes

When the Paned widget is resized it must determine a new size for each pane. There are two methods of doing this. The Paned widget can either give each pane its preferred size and then resize the panes to fit, or it can use the current sizes and then resize the panes to fit. The **XmNresizeToPreferred** constraint resource allows the application to tell the Paned widget whether to query the child about its preferred size (subject to the **XmNpreferredPaneSize**) or to use the current size when refiguring the pane locations after the Paned widget has been resized.

All panes assume they should resize to their preferred size until the Paned widget becomes visible.

In order to make effective use of the Paned widget, it is helpful to know the rules it uses to determine which child will be resized in any given situation.

Rules for determined resizing

There are three rules used to determine which child is resized. While these rules are always the same, the panes that are searched can change depending upon what caused the change of layout.

1. Do not let a pane grow larger than its maximum or smaller than its minimum size. In addition do not let a pane without a sash shrink below its preferred size due to a sash movement of another pane.
2. Do not adjust panes when **XmNskipAdjust** is True.
3. Do not adjust panes away from their preferred size, although moving one closer to its preferred size is permitted.

When searching the children of the Paned widget, it looks for panes that satisfy all the rules. If unsuccessful, it overrides rule 3 and then rule 2. Rule 1 is always enforced.

Search order

If **XmNorientation** is XmVERTICAL, the following search order is observed. If **XmNorientation** is XmHORIZONTAL, substitute “right” for “bottom” and “left” for “top” in what follows:

- If the layout is due to a resize or change in management, the panes are searched from bottom to top.
- If space is needed above the current sash, the panes are searched from bottom to top beginning with the second pane above the sash that was moved.
- If space is needed below the current sash the panes are searched from top to bottom beginning with the second pane below the sash that was moved.

The Paned widget never wraps around to the opposite side in search of a pane to resize. For example, if space is needed below the sash, no widget above the sash is ever resized.

Special considerations

When a user resizes a pane using the sashes, the Paned widget assumes that this new size is the preferred size of both the two adjacent panes, unless the **B** constraint resource is True for that pane.

Classes and Inherited Resources

Paned inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNcursor XmCCursor	None Cursor	CSG
XmNmarginHeight XmCMargin	3 Dimension	CSG
XmNmarginWidth XmCMargin	3 Dimension	CSG
XmNorientation XmCOrientation	XmVERTICAL unsigned char	CSG
XmNrefigureMode XmCBoolean	True Boolean	CSG
XmNsashHeight XmCSashHeight	10 Dimension	CSG
XmNsashIndent XmCSashIndent	-10 Position	CSG
XmNsashShadowThickness XmCSashShadowThickness	2 Dimension	CSG
XmNsashTranslations XmCTranslations	see below XtTranslations	CSG
XmNsashWidth XmCSashWidth	10 Dimension	CSG
XmNseparatorOn XmCSeparatorOn	True Boolean	CSG
XmNspacing XmCSpacing	10 Dimension	CSG

XmNcursor

Specifies the image that is displayed as the pointer cursor whenever the pointer is over this widget. If a child does not explicitly set its cursor attribute, it inherits this **XmNcursor** value.

XmNmarginHeight

XmNmarginWidth

Specifies the number of pixels between the children and the edges of the Paned widget.

XmNorientation

Specifies the orientation of the panes, either XmVERTICAL or XmHORIZONTAL. The value XmVERTICAL means the panes are placed one above the other in a column, while XmHORIZONTAL means the panes are placed next to each other in a row.

XmNrefigureMode

Specifies whether the Paned widget should adjust the sizes of the children after a resize. **XmNrefigureMode** should probably be set to False when many of the Paned widget's children will make geometry requests. This causes the Paned widget to restrict the actual movement and resize requests of the children until refigure mode is set back to True. For more information, refer to “*Geometry Management*” on page 88.

XmNsashHeight**XmNsashWidth**

Specifies the height and width of all sashes used by the Paned widget.

XmNsashIndent

Specifies the position of the sash along each pane. Positive values specify an indent from the left or top edge. Negative values specify an indent from the right or bottom edge.

XmNsashShadowThickness

Specifies thickness of the shadows drawn on each sash.

XmNsashTranslations

Specifies translation bindings for the sash. See “*Translations and Actions*” on page 93.

XmNseparatorOn

If True, the widget places a Separator between each pane.

XmNspacing

Specifies the amount of space left between the panes. This is always forced to be at least as large as the size of the sash between the panes if that sash is shown.

Constraint Resources

Name Class	Default Type	Access
XmNallowResize XmCBoolean	False Boolean	CSG
XmNpaneMaximum XmCPaneMaximum	1000 Dimension	CSG
XmNpaneMinimum XmCPaneMinimum	1 Dimension	CSG
XmNpreferredPaneSize XmCPreferredPaneSize	XiPanedAskChild Dimension	CSG
XmNresizeToPreferred XmCBoolean	False Boolean	CSG
XmNshowSash XmCBoolean	True Boolean	CSG
XmNskipAdjust XmCBoolean	False Boolean	CSG

These resources can be specified on each child of the Paned widget.

XmNallowResize

If False, the widget ignores all requests by this child to change its geometry.

XmNpaneMaximum

XmNpaneMinimum

Specifies the maximum and minimum size of a pane. If **XmNpaneMaximum** is equal to **XmNpaneMinimum**, no sash is shown.

XmNpreferredPaneSize

Specifies the preferred size of the pane. If this value is set to **XiPanedAskChild** (use the value 0 in a defaults file), the Paned widget queries the child for a preferred size. This resource allows the user or application to provide a new preferred size.

XmNresizeToPreferred

Specifies whether to resize all panes to the preferred size when the Paned window is resized. If False, only those panes not previously resized with the sashes are resized to their preferred size.

XmNshowSash

Displays the Sash below or to the right of the pane.

XmNskipAdjust

If True, the Paned window skips over this child on its first pass when choosing a pane to be forced away from its preferred size.

Translations and Actions

The Paned window inherits its translations from the **XmManager** class.

The following table lists the default translation bindings used by the control sashes within the Paned window:

<Key>osfHelp:	Help()
!Control<Key>osfUp:	SashAction(Key,10,Up)
None<Key>osfUp:	SashAction(Key,1,Up)
!Control<Key>osfDown:	SashAction(Key,10,Down)
None<Key>osfDown:	SashAction(Key,1,Down)
!Control<Key>osfLeft:	SashAction(Key,10,Left)
None<Key>osfLeft:	SashAction(Key,1,Left)
!Control<Key>osfRight:	SashAction(Key,10,Right)
None<Key>osfRight:	SashAction(Key,1,Right)
Shift~Meta~Alt<Key>Tab:	PrevTabGroup()
~Meta~Alt<Key>Tab:	NextTabGroup()
None<Btn1Down>:	SashAction(Start)
None<Btn1Motion>:	SashAction(Move)
None<Btn2Down>:	SashAction(Start)
None<Btn2Motion>:	SashAction(Move)
<BtnUp>:	SashAction(Commit)
<FocusIn>:	SashFocusIn()
<FocusOut>:	SashFocusOut()
<Unmap>:	PrimitiveUnmap()
<EnterWindow>:	enter()
<LeaveWindow>:	leave()

The following actions are supported by the Paned widget on its sashes:

SashAction(Start)

Sets up the Paned Window for sash resizing.

SashAction(Move)

Causes track lines to be drawn following the current pointer position.

SashAction(Commit)

Commits the placement operation.

SashAction(Key, Incr, Dir)

Controls the placement of the sash that has keyboard focus. **Incr** may be either DefaultIncr (1 pixel), LargeIncr (10 pixels), or a number of pixels to increment with each keystroke. **Dir** is either Up, Down, Left or Right. Only Up and Down operate when **XmNorientation** is XmVERTICAL. Only Left and Right operate when **XmNorientation** is XmHORIZONTAL.

Convenience Routine

XiPanedGetPanes()

Retrieves the panes in the widget . Because the Paned widget adds children other than the panes, these values are not the same as those retrieved with the **XmNchildren** and **XmNnumChildren** resources.

```
void XiPanedGetPanes( Widget w,
                    WidgetList *panes_returned,
                    int *num_panes_returned)

w                Widget ID of the Paned window.

panes_returned   List of panes in the Paned window widget.

num_panes_returned   Number of panes in the Paned window widget.
```

Note: *panes_returned* and *num_panes_returned* can be pointers to actual elements in the widget structure. Any changes to the managed state of the widget, or deletion of widget children can cause the elements pointed to by these values to change. The values should *never* be modified by the application.

XiPanner

UNIX Application Header File	Xi/Panner.h
UNIX Class Header File	Xi/PannerP.h
Class Name	XiPanner
Class Pointer	xiPannerWidgetClass
Superclass Name	XmPrimitive
Creation Routine	XiCreatePanner

The Panner widget represents a rectangular region (called the *canvas*) of which only a smaller, enclosed rectangular region (called the *slider*) is visible at any given time. It is typically used with a Porthole widget to scroll a third widget in two dimensions.

The slider can be moved around the canvas by pressing, dragging, and releasing the Select Button. While scrolling is in progress, the application or other widget receives notification through a callback procedure which it can use to update any associated widgets.

Notification can be done either continuously whenever the slider moves or discretely whenever the slider has been given a new location.

Figure 19 shows two views of a Panner with the slider in different positions:



Figure 19. Panner and Porthole Widgets

The simplest method of adding a Panner to your application is to connect it to a Porthole widget with `XiPortholeConnectPanner()`, and add the child to be scrolled to the porthole widget.

Classes and Inherited Resources

Panner inherits behavior and resources from **Core** and **XmPrimitive**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNcanvasHeight XmCCanvasHeight	0 Dimension	CSG
XmNcanvasWidth XmCCanvasWidth	0 Dimension	CSG
XmNdefaultScale XmCDefaultScale	8 Dimension	CSG
XmNinternalSpace XmCInternalSpace	2 Dimension	CSG
XmNreportCallback XmCReportCallback	NULL XtCallbackList	CSG
XmNresize XmCResize	False Boolean	CSG
XmNrubberBand XmCRubberBand	False Boolean	CSG
XmNsliderHeight XmCSliderHeight	0 Dimension	CSG
XmNsliderWidth XmCSliderWidth	0 Dimension	CSG
XmNsliderX XmCSliderX	0 Position	CSG
XmNsliderY XmCSliderY	0 Position	CSG

XmNcanvasHeight

XmNcanvasWidth

Specifies the height and width of the canvas. The preferred size of the slider is calculated as the canvas size multiplied by the scaling factor **XmNdefaultScale**, expressed as a

percentage.

XmNdefaultScale

Specifies the percentage size that the Panner widget would prefer to have relative to the size of the canvas.

XmNinternalSpace

Specifies the width of internal border in pixels between a slider representing the full size of the canvas and the edge of the Panner widget.

XmNreportCallback

All functions on this callback list are called when the notify action is invoked. See “*Translations and Actions*” on page 99 for details.

XmNresize

Specifies whether or not to maintain **XmNdefaultScale** by resizing the Panner when the canvas size is changed.

Note: Feedback loops can result if the Panner and the object panned affect each other’s size.

XmNrubberBand

Specifies whether or not scrolling should be discrete (only moving a rubber-banded rectangle until the scrolling is done) or continuous (moving the slider itself). This resource also controls whether or not the move action procedure also invokes the notify action procedure.

XmNshadowThickness

Specifies the width in pixels of the shadow drawn around the slider and around the Panner widget itself.

XmNsliderHeight**XmNsliderWidth**

Specifies the height and width of the slider expressed in the coordinates of the canvas.

XmNsliderX**XmNsliderY**

Specifies the location of the slider in the coordinates relative to the canvas.

Translations and Actions

The translations supported by the Panner widget are:

<Btn1Down>:	XiGetFocus() XiStart()
<Btn1Motion>:	XiMove()
<Btn1Up>:	XiNotify() XiStop()
<Btn2Down>:	XiAbort()
<Btn3Down>,<Btn3Up>:	XiWarpTo()
<Key>osfSelect:	XiSet(rubberband,toggle)
<Key>osfActivate:	XiPage(+1p,+1p)
None<Key>Return:	XiPage(+1p,+1p)
None<Key>space:	XiPage(+1p,+1p)
<Key>osfDelete:	XiPage(-1p,-1p)
<Key>osfBackSpace:	XiPage(-1p,-1p)
<Key>osfLeft:	XiPage(-.5p,+0)
<Key>osfRight:	XiPage(+.5p,+0)
<Key>osfUp:	XiPage(+0,-.5p)
<Key>osfDown:	XiPage(+0,+.5p)

The following actions are supported by the Panner widget:

XiStart()

Begins movement of the slider.

XiAbort()

Ends movement of the slider and restores its original position.

XiGetFocus()

Forces the Panner to have the current Motif keyboard traversal sent to it.

XiMove()

Moves the outline of the slider (if the **XmNrubberBand** resource is True) or the slider itself (by invoking the notify action procedure).

XiNotify()

Informs the application of the slider's current position by invoking the **XmNreportCallback** functions registered by the application.

XiPage(xamount, yamount)

Moves slider by specified amounts. Format for the amounts is a signed or unsigned floating-point number (such as +1.0 or -.5) followed by either ‘p’ indicating pages (slider sizes), or ‘c’ indicating absolute position as a fraction of canvas size.

XiPage(+0,+.5p) represents vertical movement down one-half height of slider and XiPage(0,0) represents moving to upper left corner of canvas.

XiSet(what, value)

Changes the behavior of the Panner. The *what* argument must currently be the string “rubberband” and controls the value of the **XmNrubberBand** resource. The **value** argument can have the value of “on”, “off”, or “toggle”.

XiStop()

Ends movement of the slider.

XiWarpTo()

Sets the slider’s upper-left corner to the location of the event.

XiScrollReport

A generic structure used to communicate scrolling or panning information between widgets, or from widget to application and back again. This structure is defined in the header file `<Xi/Reports.h>`.

```
typedef struct _XiScrollReport {
    unsigned int changed;
    Position slider_x, slider_y;
    Dimension slider_width, slider_height;
    Dimension canvas_width, canvas_height;
} XiScrollReport;
```

changed Bitmask containing list of elements in report callback structure that have changed (causes this report to be sent). All fields valid with every call. Informs application which fields were modified. Acceptable values are XiPRSliderX, XiPRSliderY, XiPRSliderWidth, XiPRSliderHeight, XiPRCanvasWidth, and XiPRCanvasHeight.

slider_x Location of the slider in canvas coordinates and units.
slider_y

<i>slider_width</i>	The width and height of the slider in canvas coordinates.
<i>slider_height</i>	
<i>canvas_width</i>	The width and height of the canvas as computed by the size
<i>canvas_height</i>	of the Panner multiplied by the XmNdefaultScale resource.

Callback Routine

All routines on the **XmNreportCallback** list are called with an `XiScrollReport` structure as `call_data`.

Convenience Routine

XiPortholeConnectPanner()

Connects a Porthole widget to a Panner widget.

```
void XiPortholeConnectPanner (Widget porthole,  
                             Widget panner)
```

porthole Porthole widget to connect.

panner Panner widget to which the Porthole connects.

This routine connects the Porthole and Panner widget together to allow scrolling the Porthole's child by moving the Panner. In addition, the appropriate actions are taken when the user resizes the Porthole, makes a `XtSetValues()` call to move its child, or moves the Panner. There is no restriction on the relative location of Porthole and Panner. The location and size of each widget are completely customizable.

XiPixmapEditor

UNIX Application Header File	<code>Xi/PixEdit.h</code>
UNIX Class Header File	<code>Xi/PixEditP.h</code>
Class Name	<code>XiPixmapEditor</code>
Class Pointer	<code>xiPixmapEditorWidgetClass</code>
Superclass Name	<code>XiPaned</code>
Creation Routine	<code>XiCreatePixmapEditor</code>

The PixmapEditor widget allows users to generate graphics for use in an application. It supports many common functions including point, line, circle, filled circle, rectangle, filled rectangle, moving and copying areas, flood fill, and undo. The image can also be resized, zoomed, or scrolled using a Panner. This widget can accept an input image from either an image or a drawable.

If the input is from a drawable, any portion of that drawable may be loaded using the **XmNdrawableInfo** resource. Figure 20 shows the PixmapEditor display:

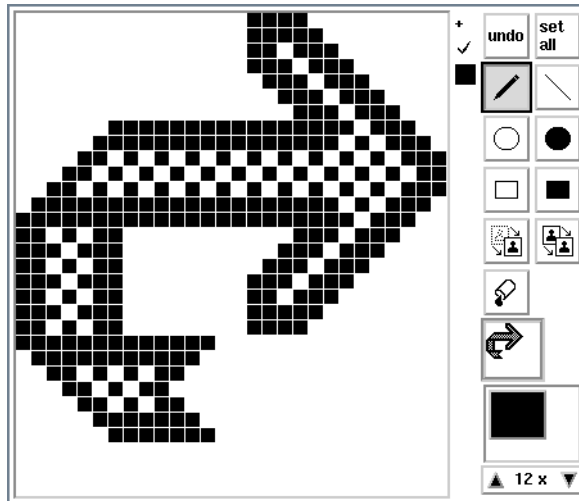


Figure 20. *PixmapEditor* Widget

Supported functions

This section lists the supported functions:

Set All

To fill the entire image with a given color, erasing its current contents, select the *Set All* button.

Undo

To undo the last operation, select the *Undo* button. Only the last operation can be undone. If draw point was the last operation, all points that were drawn with one click-drag-release action are removed.



Draw Point

To draw a single point, click and release the select pointer button. To draw multiple points, click and drag the pointer. Points are drawn until the pointer button is released.



Draw Line

To draw a line, click and hold to set one end of the line, then drag the pointer and release the button to set the other end of the line.



Draw Circle

To draw a circle, click and hold to set the center of the circle. Drag out along the radius to set the size of the circle.



Filled Circle

A filled circle is drawn in the same fashion as a circle.



Rectangle

To draw a rectangle, click and hold to position the upper-left corner of the rectangle, then drag the pointer and release the button to set the opposite corner.



Filled Rectangle

A filled rectangle is drawn in the same fashion as a rectangle.



Copy

To copy an area of the screen to another location, click and hold to set the upper-left corner and drag out a rectangle encompassing the area to copy. Release the pointer button and move the selected area to the new location and click the pointer button to complete the copy. The image in the original location is unchanged and a copy of it is placed in the new location. To cancel a copy operation once the original area has been selected, select the copy icon again.



Move

The move operation is just like the copy operation except that after moving the area to a new location, the original area is removed by filling it with the currently selected color.



Flood Fill

To flood fill an area, choose a location, then click and release the select button. All contiguous pixels of the same color are changed to the current color.

Selecting and Adding Colors

Select the color used in each of the above functions with the color bar. The color bar, placed between the pixmap and the control panel, is a bar of rectangles which correspond to colors in the pixmap.

- The current color is denoted by a check mark. Selecting a color makes it the current color.
- Click the “+” pushbutton to add colors to the bar.
- Double-click on the newly created color to display the ColorSelector, which allows you to change that color in the pixmap.

Resizing the Pixmap

To resize the pixmap, click and hold on an edge of the small pixmap display and drag it to a new location. The size in pixels of the new image is displayed in the upper-left corner of the small pixmap. When the image is made bigger, all new pixels are set to the current color. When the image is made smaller, only the upper-left portion of the image that fits on the screen is saved.

Panning and Zooming

This area allows you to zoom and pan the pixmap. Grabbing the panner with the pointer button moves your current view of the pixmap to a new location. Dragging the panner to the extreme top right of its bounding area allows you to see the top-right corner of the pixmap. If the panner completely fills its area, the entire pixmap is currently shown, and the panner cannot be moved (see Figure 21). When the panner has the keyboard focus it can also be moved by using the arrow keys.

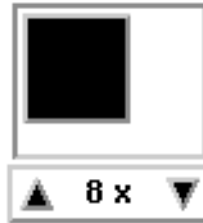


Figure 21. Pixmap Panner

To zoom the image in and out, use the up and down arrow buttons. This changes the magnification of the drawing pixmap. A magnification of “8x” means that each pixel in the real pixmap is represented by an 8x8 rectangle in the PixmapEditor. Zooming in and out does not change the image—it only affects your view of the image. The allowed magnifications are 1, 2, 4, 6, 8, 10, and 12.

Input and Output

To set the image in the PixmapEditor, either set the **XmNimage** resource to contain a valid **XImage** structure, or set **XmNdrawable** to point to an X drawable (either a window or pixmap). The **XmNdrawableInfo** resource can be used to have the pixmap load in a portion of a window. This can be used to extract a section of the root window. Changing either of these resources resets **XmNpixmapWidth** and **XmNpixmapHeight** and completely replaces the current pixmap.

The contents of the PixmapEditor can be retrieved using the same two resources, thus an application can retrieve the image as an **XImage** or an X Pixmap.

Classes and Inherited Resources

PixmapEditor inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XiPaned**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer’s Reference*.

Resources

Name Class	Default Type	Access
XmNbitmapMode XmCBitmapMode	False Boolean	CSG
XmNcellHeight XmCCellSize	8 int	CSG
XmNcellWidth XmCCellSize	8 int	CSG
XmNcurrentColor XmCColorIndex	-1 int	CSG
XmNdrawable XmCDrawable	XtUnspecifiedPixmap Drawable	CSG
XmNdrawableInfo XmCDrawableInfo	NULL XtPointer	CSG
XmNeditCursor XmCCursor	crosshair Cursor	CSG
XmNgridSize XmCGridSize	1 int	CSG
XmNgridTranslations XmCGridTranslations	see below XtTranslations	CSG
XmNimage XmCImage	NULL Pointer	CSG
XmNmaxHeight XmCMaxSize	32000 int	G
XmNmaxWidth XmCMaxSize	32000 int	G
XmNminHeight XmCMinSize	1 int	G

Name (continued) Class	Default Type	Access
XmNminWidth XmCMinSize	1 int	G
XmNmodified XmCModified	False Boolean	CSG
XmNpalette XmCPixelList	NULL Pixel *	CSG
XmNpaletteSize XmCColorIndex	0 int	CSG
XmNpixmapHeight XmCPixmapWidth	48 int	CSG
XmNpixmapWidth XmCPixmapHeight	48 int	CSG
XmNsetAllString XmCSetAllString	“set\nall” XmString	CSG
XmNshowNormalView XmCBoolean	True Boolean	CSG
XmNtile XmCTile	XtUnspecifiedPixmap Pixmap	CSG
XmNundoString XmCUndoString	“undo” XmString	CSG

XmNbitmapMode

If this resource is True, the PixmapEditor expects the image or drawable passed to it to be a bitmap, and only allows the user to change each cell in the bitmap to a 1 or a 0. Otherwise, full color editing is allowed. If a bitmap is loaded into a PixmapEditor that is not in bitmap mode, the 1’s are converted to black and the 0’s are converted to white.

XmNcellHeight

XmNcellWidth

Specifies the height and width in pixels of each cell in the big pixmap. Although these can be specified independently if the user zooms the application in or out, the average is used for both, forcing the aspect ratio back to 1.

XmNcurrentColor

Specifies the index in the **XmNpalette** array of the currently selected color.

XmNdrawable

When set, **XmNdrawable** can contain any valid drawable on this display. If **XmNdrawableInfo** is NULL, the entire drawable is loaded into the PixmapEditor. If **XmNdrawableInfo** is not NULL, the value of **XmNdrawableInfo** specifies which portion of the drawable is loaded into the PixmapEditor. The drawable is converted to an internal format at initialization time and is never referenced again. The programmer is free to destroy this drawable after the widget has been created or the XtSetValues() call returns.

When retrieved via XtGetValues(), **XmNdrawable** contains the image that has been created or edited with the PixmapEditor, returned as a Pixmap. This pixmap is only valid until the next time this resource is retrieved using XtGetValues() or set using XtSetValues(). Therefore, you should copy the pixmap if it will be used after the next call.

XmNdrawableInfo

A structure that specifies which portion of the drawable specified by **XmNdrawable** should be loaded into the PixmapEditor. The structure has the following format.

```
typedef struct _XiPixmapEditorDrawableInfo {
    short x, y;
    unsigned short width, height, depth;
} XiPixmapEditorDrawableInfo;
```

x, y Upper-left corner of the area to load.

width, height Size of the area to load.

depth Depth of the drawable being used.

XmNeditCursor

Specifies the cursor displayed when the pointer is over the active edit area.

XmNgridSize

Specifies the size of the grid (in pixels). The grid is constructed by letting the **XmNbackground** of the window show through. The grid size is *subtracted from* the **XmNcellWidth** and **XmNcellHeight**. Therefore, if the **XmNcellHeight** is 8 and the **XmNgridSize** is 1, the actual size of the drawn rectangle is 7 pixels. If the average of **XmNcellHeight** and **XmNcellWidth** is less than 3, **XmNgridSize** is forced to 0.

XmNgridTranslations

Specifies the translations that are active on the big pixmap. See “*Translations and Actions*” on page 111.

XmNimage

When set by using `XtSetValues()`, contains a pointer to an `XImage` that provides the initial data for the `PixmapEditor`. The entire `XImage` is always loaded into the `PixmapEditor`. The `XImage` is converted to an internal format at initialization time and is never referenced again. The programmer is free to destroy this image after the widget has been created.

When retrieved by using `XtGetValues()`, contains a pointer to the image that has been created or edited with the `PixmapEditor`. This `XImage` is only valid until the next time this resource is retrieved using `XtGetValues()` or set using `XtSetValues()`. Therefore, you should copy the `XImage` if it will be used after the next call.

XmNmaxHeight**XmNmaxWidth**

Specifies the maximum height and width the pixmap can ever become. If the image or drawable passed in exceeds this size, it is cropped to this size.

XmNminHeight**XmNminWidth**

Specifies the minimum height and width the pixmap can ever become. If the image or drawable passed in is smaller than this size it is placed in the upper-left corner of the area, and a convenient color is used to fill in the rest of the image.

XmNmodified

This resource is set to `True` when the user modifies the image. The value is set to `False` only when the **XmNdrawable** or **XmNimage** is changed by a call to `XtSetValues()`. This value is otherwise not used internally.

XmNpalette

Specifies an array of pixel values that is the palette for PixmapEditor widgets created with XmUNSPECIFIED_PIXMAP. A comma-separated list of color names is valid in a resource file. A String to PixelList converter has been registered. This converts comma-separated lists of colors to a pixel array.

XmNpaletteSize

Specifies the number of colors in the palette. This must match the actual size of the list or an error will occur.

XmNpixmapHeight**XmNpixmapWidth**

If no drawable or image is passed into the PixmapEditor, these resources specify the initial height and width of the pixmap. These resources are subject to the constraints of the minimum and maximum bounds. If a drawable or image is passed in, these values are ignored and reset to correspond to the height and width of that image or drawable.

XmNsetAllString

Specifies the label of the *Set All* pushbutton.

XmNshowNormalView

If this resource is False, the actual size of the pixmap, which is normally displayed above the panner, is not shown.

XmNtile

Specifies the pixmap used to tile the unused area of the PixmapEditor Porthole's background.

XmNundoString

Specifies the label of the *Undo* pushbutton.

Translations and Actions

The following are the default translation bindings available in the pixmap editing window. These may be modified by setting the **XmNgridTranslations** resource.

<Btn1Down>:	XiStartPoint()
<Btn1Up>:	XiEndPoint()
<Btn1Motion>:	XiMotion(BtnDown)
<Motion>:	XiMotion(NoButton)
<Btn2Down>,<Btn2Up>:	XiNextColor()

The following actions are supported by the PixmapEditor widget:

XiStartPoint()

Specifies the start location of an action in the PixmapEditor.

XiEndPoint()

Specifies the end location of an action in the PixmapEditor.

XiMotion()

Specifies an intermediate point of an action in the PixmapEditor. The argument determines whether the PixmapEditor should perform actions that only happen on a drag, or those that happen whenever the pointer is moved.

XiNextColor()

Selects the next available color from the list of color buttons along the bottom of the window. If the last button is currently selected, this action wraps to the first one.

Compound Widget Hierarchy

The PixmapEditor is composed of several sub-widgets. Most resource values that are passed to the PixmapEditor through the argument list at creation time or via XtSetValues() are then passed to each of the widget's children. An XtGetValues() request for a child widget's resource value must be made explicitly on the child. For more information on passing arguments to the EnhancementPak compound widgets and retrieving the widget ids of the child widgets, refer to "Compound Widgets" on page 12.

Consult the *OSF/Motif Programmer's Reference* for the list of any child widget's resources.

XiPixmapEditor	<named by application>
XiPorthole	porthole
XmDrawingArea	bigmap
XmForm	form
XmFrame	frame
XiButtonBox	colorBox
XiIconButton	newColor
XiIconButton	colorToggle
XiIconButton	colorToggle
XiPaned	panelPaned
XiToolbar	controlBox
XiIconButton	command0
XiIconButton	command1
.	.
.	.
.	.
XiIconButton	command11
XmForm	form
XiStretch	stretch
XmDrawingArea	littleMap
OverrideShell	labelShell
XmLabel	label
XiPanner	panner
XmFrame	frame
XiPaned	paned
XmArrowButton	topArrow
XmLabel	label
XmArrowButton	bottomArrow
Core	spacer
XmSash	sash

Convenience Routines

XiPixmapEditorGetPictureData

Returns pixmap status information without a callback. The picture data is an array of unsigned chars whose values represent offsets into the color array. Once the image has been modified call XiPixmapEditorRefresh() to update the display.

```
void XiPixmapEditorGetPictureData ( Widget w,  
                                   int reset_undo,  
                                   XColor **colors,  
                                   int *ncolors,  
                                   unsigned char **picture,  
                                   unsigned int *width,  
                                   unsigned int *height)
```

<i>w</i>	The PixmapEditor widget.
<i>reset_undo</i>	If True the undo buffer is set to the current image before the data is returned.
<i>colors, ncolors</i>	The colors used in this image are returned by this pointer. This list must not be modified.
<i>picture</i>	The picture data is returned here.
<i>width, height</i>	The size of the pixmap is placed in these variables.

XiPixmapEditorGetRelevantData

Returns pixmap status information without a callback; a pointer to static data.

```
XiPixmapEditorCallbackStruct* XiPixmapEditorGetRelevantData (Widget w)
```

<i>w</i>	The PixmapEditor widget.
----------	--------------------------

XiPixmapEditorRefresh

Tells the PixmapEditor to refresh its display. If you call XiPixmapEditorGetPictureData and modify the picture data, be sure to call this routine to redisplay the image.

```
void XiPixmapEditorRefresh (Widget w)
```

<i>w</i>	The PixmapEditor widget.
----------	--------------------------

XiPorthole

UNIX Application Header File	<code>Xi/Porthole.h</code>
UNIX Class Header File	<code>Xi/PortholeP.h</code>
Class Name	<code>XiPorthole</code>
Class Pointer	<code>xiPortholeWidgetClass</code>
Superclass Name	<code>XmManager</code>
Creation Routine	<code>XiCreatePorthole</code>

The Porthole widget allows a single child to be managed and scrolled by the application. Unlike the Motif Scrolled Window, no scrollbars are provided.

The application is free to provide scrolling by attaching its own scrollbars or connecting the Porthole to another type of scrolling device, typically the Panner widget.

The Porthole handles `XtSetValues()` and geometry management requests from the child, allowing the application to call `XtSetValues()` with a new X and Y location for the child.

Figure 22 shows two views of a Panner with the slider in different positions:



Figure 22. Panner and Porthole Widgets

By default, the Porthole widget is sized to have its child completely fill the clip area when that child has its preferred geometry. This allows applications to place a Porthole around a child while still allowing the child to display all of its data by default. When the child is unable to show everything in the current screen space, scrolling is available.

The Porthole's clip window has its **XmNbackground** set to None when **XmNforceChildToFill** is True. This allows the child to be mapped and unmapped during complex resize operations without a screen flash. This technique can be used to increase application performance by reducing the number of graphics operations the X server must perform during geometry negotiations.

Geometry Management

The Porthole widget allows its managed child to request any size, unless **XmNforceChildToFill** is True (in which case the child must be at least as large as the Porthole's clip window). Any location can be specified so long as the child still obscures all of the clip window. If the child only covers part of the window, it must be placed in the upper-left corner of the window. This widget is typically connected by the **XmNreportCallback** with a scrolling device such as the Panner widget.

As specified by the Xt Ininsics, the programmer should make a `XtSetValues()` call on the child widget specifying the new **XmNx**, **XmNy**, **XmNwidth**, and **XmNheight** of the Porthole's child.

Classes and Inherited Resources

Porthole inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer's Reference*.

Resources

Name Class	Default Type	Access
XmNforceChildToFill XmCBoolean	True Boolean	CSG
XmNmarginHeight XmCMargin	0 Dimension	CSG
XmNmarginWidth XmCMargin	0 Dimension	CSG
XmNreportCallback XmCReportCallback	NULL XtCallbackList	CSG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG

XmNforceChildToFill

Forces the child to fill the entire space provided by the Porthole. Thus, the minimum size the Porthole will ever allow the child to be is the current size of the Porthole's clip window.

XmNmarginHeight

Specifies the vertical margin around the Porthole widget's child

XmNmarginWidth

Specifies the horizontal margin around the Porthole widget's child

XmNreportCallback

Specifies a list of functions to invoke whenever the managed child widget changes size or position.

XmNshadowThickness

Specifies thickness of the shadow border around the Porthole widget's child.

Callback Routine

All routines on the **XmNreportCallback** list are called with an **XiScrollReport** structure as `call_data`.

XiScrollReport

The **XiScrollReport** structure is a generic structure used to communicate scrolling or panning information between widgets, or from widget to application and back again. This structure is defined in the header file `<Xi/Reports.h>`.

```
typedef struct _XiScrollReport {
    unsigned int changed;
    Position slider_x, slider_y;
    Dimension slider_width, slider_height;
    Dimension canvas_width, canvas_height;
} XiScrollReport;
```

changed

Bitmask containing a list of the elements in the report callback structure that have changed, causing this report to be sent. All fields are valid with every call, this mask simply informs the application which fields have been modified. The acceptable values are `XiPRSliderX`, `XiPRSliderY`, `XiPRSliderWidth`, `XiPRSliderHeight`, `XiPRCanvasWidth`, and `XiPRCanvasHeight`.

slider_x,
slider_y

Location of the clip window in the coordinates of the child. This is backwards from the way X deals with these objects, but much more useful for scrolling devices.

slider_width,
slider_height

Width and height of the clip window in pixels.

canvas_width,
canvas_height

Width and height of the Porthole's child widget in pixels.

Convenience Routines

XiPortholeConnectPanner()

Connects a Porthole widget to a Panner widget.

```
void XiPortholeConnectPanner(  Widget porthole,  
                               Widget panner)
```

porthole Porthole widget to connect.

panner Panner widget to which the Porthole connects.

This routine connects the Porthole and Panner widget together to allow scrolling the Porthole's child by moving the Panner. In addition, the appropriate actions are taken when the user resizes the Porthole, makes a XtSetValues() call to move its child, or moves the Panner. There is no restriction on the relative location of Porthole and Panner. The location and size of each widget are completely customizable.

XiPortholeVisible()

Makes an obscured or partially obscured widget descendant of the XiPorthole centered within the porthole, if possible.

```
void XiPortholeVisible (Widget porthole, Widget descendant)
```

porthole The Porthole widget.

descendant Descendant widget to center within Porthole.

XiStretch

UNIX Application Header File	<code>Xi/Stretch.h</code>
UNIX Class Header File	<code>Xi/StretchP.h</code>
Class Name	<code>XiStretch</code>
Class Pointer	<code>xiStretchWidgetClass</code>
Superclass Name	<code>XmFrame</code>
Creation Routine	<code>XiCreateStretch</code>

The Stretch widget puts a border around its single managed child that the user can grab. The X cursor changes to a resize cursor similar to those used by the Motif Window Manager when over a section of the widget that can be used to resize the child. By selecting and dragging the border around the Stretch widget's single child, both the Stretch and the child resize dynamically.

The application can set maximum and minimum sizes to which the user can resize the Stretch widget's child. The size of the increment can also be specified, providing functionality similar to the *width_inc* and *height_inc* window manager hints. A callback for resize events is available.

Classes and Inherited Resources

Stretch inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XmFrame**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer's Reference*.

Resources

Name Class	Default Type	Access
XmNheightInc XmCIncrement	1 Dimension	CSG
XmNmaxHeight XmCMaxSize	32000 int	CSG
XmNmaxWidth XmCMaxSize	32000 int	CSG
XmNminWidth XmCMinSize	1 int	CSG
XmNminHeight XmCMinSize	1 int	CSG
XmNresizeCallback XmCCallback	NULL XtCallbackList	CSG
XmNshadowThickness XmCShadowThickness	5 short	CSG
XmNwidthInc XmCIncrement	1 Dimension	CSG

XmNheightInc

XmNwidthInc

Specifies the height and width of one cell. The Stretch widget allows the child to be resized only in increments of the height and width of a cell. When resizing the widget, a window pops up to show the current height and width of the child in cells, rather than pixels.

XmNmaxHeight

XmNmaxWidth

Specifies the maximum height and width, in cells, that the child can become. The height and width of a cell are specified by the **XmNheightInc** and **XmNwidthInc** resources.

XmNminHeight

XmNminWidth

Specifies the minimum height and width, in cells, that the child can become. The height and width of a cell are specified by the **XmNheightInc** and **XmNwidthInc** resources.

XmNresizeCallback

Specifies the list of callback routines called whenever the Stretch widget is resized.

XmNshadowThickness

Specifies the thickness of the shadow placed around the child. This area is grabbed and dragged to resize the child.

Translations and Actions

The following are the default translation bindings available in the Stretch widget:

<Btn1Down>:	XiStart()
<Btn1Up>:	XiEnd()
<Btn1Motion>:	XiMovement()

The following actions are supported by the Stretch widget:

XiStart()

Specifies the start location of Stretch.

XiEnd()

Specifies the end location of Stretch.

XiMotion()

Specifies an intermediate point of the Stretch.

Callback Routine

When the Stretch widget attempts to resize, all routines on the **XmNresizeCallback** are called to inform the application that a change has occurred. The Stretch widget automatically uses the geometry request of its parent. This callback informs the application what change has occurred. In most cases, this information is not necessary, since the child of the Stretch widget has already had its resize procedure called.

All routines on the **XmNresizeCallback** list are passed a pointer to the `XiStretchWidgetInfo` structure as `client_data`.

XiStretchWidgetInfo Structure

The XiStretchWidgetInfo data structure contains information about the new height and width and location of the Stretch widget's child.

```
typedef struct _XiStretchWidgetInfo {
    Boolean success;
    Dimension width, height;
    Position x, y;
} XiStretchWidgetInfo;
```

<i>success</i>	Whether the Stretch widget and its parent agreed to grant the user's request. If True, the widget has been resized. If False, the resize was disallowed.
<i>width, height</i>	The width and height of the Stretch widget, in units of XmNwidthInc and XmNheightInc respectively, as requested by the user. If <i>success</i> is True, the real dimensions of the widget will equal these values since the widget has been resized.
<i>x, y</i>	The values, in pixels, of XmNx and XmNy for the Stretch widget as requested by user. Note that the Stretch widget <i>does not</i> automatically change its XmNx and XmNy values in response to an upward or leftward resize request. If the application wants to change the location of the widget, it can use these values to perform the move itself.

XiTabStack

UNIX Application Header File	<code>Xi/TabStack.h</code>
UNIX Class Header File	<code>Xi/TabStackP.h</code>
Class Name	<code>XiTabStack</code>
Class Pointer	<code>xiTabStackWidgetClass</code>
Superclass Name	<code>XmBulletinBoard</code>
Creation Routine	<code>XiCreateTabStack</code>

The `TabStack` widget manages a group of widgets such that only one widget in the group is visible at a time. Each child is associated with a tab that displays text and/or a pixmap. The user selects the tab, interactively determining which child is displayed. Tabs can be configured to appear above, below, to the right, or to the left of a work area with the text oriented in any of the four cardinal directions. A sample `TabStack` is shown in Figure 23.

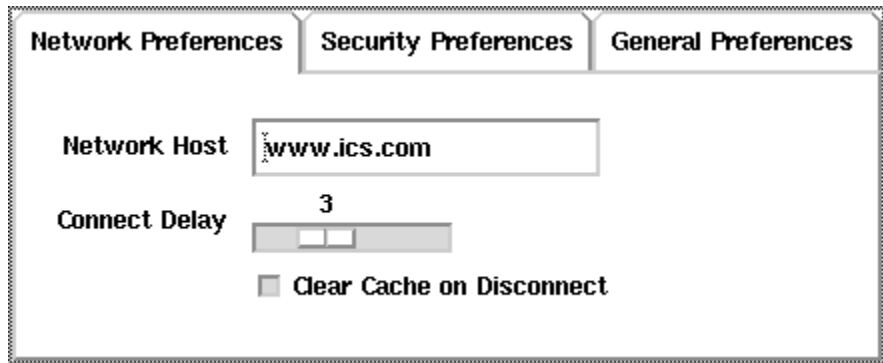


Figure 23. *TabStack* Widget

The `TabStack` allows the user to select tabs, either by pointer or keyboard traversal. When a tab is selected, the tab appears to be raised above the other tabs and the child associated with the tab is made visible. One tab is selected at all times.

Note: This widget behaves similarly to Microsoft Windows™ Tab Control.

Classes and Inherited Resources

`TabStack` inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XmBulletinBoard** classes.

The resources inherited from each Motif superclass are listed in the *Appendix* at the back of this book. For a complete description of each resource, refer to the *OSF/Motif Programmer's Reference*.

Resources.

Name Class	Default Type	Access
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNstackedEffect XmCStackedEffect	True Boolean	CSG
XmNtabAutoSelect XmCTabAutoSelect	True Boolean	CG
XmNtabCornerPercent XmCTabCornerPercent	40 int	CSG
XmNtabLabelSpacing XmCTabLabelSpacing	2 Dimension	CSG
XmNtabMarginHeight XmCTabMarginHeight	3 Dimension	CSG
XmNtabMarginWidth XmCTabMarginWidth	3 Dimension	CSG
XmNtabMode XmCTabMode	XiTABS_BASIC int	CSG
XmNtabOffset XmCTabOffset	10 Dimension	CSG
XmNtabOrientation XmCTabOrientation	XiTAB_ORIENTATION_DYNAMIC int	CSG
XmNtabSelectColor XmCTabSelectColor	dynamic Pixel	CSG
XmNtabSelectedCallback XmCTabSelectedCallback	NULL XtCallbackList	CSG
XmNtabSelectPixmap XmCTabSelectPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNXmNtabSide XmCTabSide	XiTABS_ON_TOP int	CSG

Name (continued) Class	Default Type	Access
XmNtabStyle XmCTabStyle	XiTABS_BEVELED int	CSG
XmNuniformTabSize XmCUniformTabSize	True Boolean	CSG
XmNuseImageCache XmCUseImageCache	True Boolean	CSG

XmNfontList

Specifies the XmFontList to use when drawing the label strings for the tabs.

XmNhighlightThickness

Specifies the thickness of the rectangle drawn around the label string and the pixmap of the tab that owns keyboard traversal.

XmNstackedEffect

Specifies whether visuals should depict a stack of folders (True), or TabStack should use all available space for its children (False).

XmNtabAutoSelect

Specifies whether a tab is automatically selected when receiving keyboard traversal.

XmNtabCornerPercent

Specifies the percent of font height to use for the corner visual.

XmNtabLabelSpacing

Specifies the amount of space between a text label and pixmap in the tab area.

XmNtabMarginHeight

Specifies the size of the vertical border placed around the label area of a tab.

XmNtabMarginWidth

Specifies the size of the horizontal border placed around the label area of a tab.

XmNtabMode

Specifies the mode in which the TabStack distributes the tabs. The following values are valid for this resource:

XmNXiTABS_BASIC	Distributes the tabs in either a vertical or horizontal row and clips the tabs if there is not enough room to display all tabs.
XmNXiTABS_STACKED	Adds additional tabs if there is not enough room to display all the tabs. The row of a selected tab is moved next to the children of the stack.
XmNXiTABS_STACKED_STATIC	Distributes the tabs in either a vertical or horizontal row. Adds additional tabs if there is not enough room to display all the tabs. The positions of rows are not changed when tabs are selected.

XmNtabOffset

Specifies the amount of indentation used to stagger the tab rows when displaying tabs in either the XiTABS_STACKED or XiTABS_STACKED_STATIC mode.

XmNtabOrientation

Specifies the orientation of the tab and the rotation factor of the tab label. The following values are valid for this resource:

XiTAB_ORIENTATION_DYNAMIC	The orientation of the tabs is calculated dynamically based on XmNtabSide resource.
XiTABS_LEFT_TO_RIGHT	Text appears at the default rotation, US English.
XiTABS_RIGHT_TO_LEFT	Text appears upside down.
XiTABS_TOP_TO_BOTTOM	Rotation of text to the vertical position with the first character drawn at the lowest y-position, and the bottom of the text should face the lowest x-position.
XiTABS_BOTTOM_TO_TOP	Rotation of text to the vertical position with the first character drawn at the highest y-position, and the bottom of the text should face the highest x-position.

XmNselectColor

Specifies the color of the selected tab.

XmNtabSelectedCallback

Specifies the list of callbacks to call when a child becomes the selected tab.

XmNselectPixmap

Specifies the pixmap of the selected tab.

XmNtabSide

Specifies the location of the tab with respect to the children of the TabStack widget. The following values are valid for this resource:

XiTABS_ON_TOP	Places the tabs above the children.
XiTABS_ON_BOTTOM	Places the tabs below the children.
XiTABS_ON_RIGHT	Places the tabs to the right of the children.
XiTABS_ON_LEFT	Places the tabs to the left of the children.

XmNtabStyle

Specifies the appearance of the tabs associated with the children of the TabStack widget. The following values are valid for this resource:

XiTABS_BEVELED	Draws the corners of the tabs as an angled line.
XiTABS_ROUNDED	Draws the corners of the tabs as a quarter of a circle.
XiTABS_SQUARED	Draws the tabs as rectangles.

XmNuniformTabSize

Gives all tabs a uniform major dimension, with the major dimension as width if the tab orientation is `XiTABS_LEFT_TO_RIGHT`, or `XiTABS_RIGHT_TO_LEFT`, or height if the tab orientation is `XiTABS_TOP_TO_BOTTOM`, or `XiTABS_BOTTOM_TO_TOP`.

XmNuseImageCache

Caches the XImages used for rotating text and pixmaps. This increases performance, but uses more memory.

Constraint Resources

Name Class	Default Type	Access
XmNfreeTabPixmap XmCFreeTabPixmap	False Boolean	CSG
XmNtabAlignment XmCAlignment	XmALIGNMENT_CENTER unsigned char	CSG
XmNtabBackground XmCBackground	dynamic Pixel	CSG
XmNtabBackgroundPixmap XmCBackgroundPixmap	dynamic Pixmap	CSG
XmNtabForeground XmCForeground	dynamic Pixel	CSG
XmNtabLabelPixmap XmCTabLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNtabLabelString XmCTabLabelString	NULL XmString	CSG
XmNtabPixmapPlacement XmCTabPixmapPlacement	XiPIXMAP_RIGHT XiPixmapPlacement	CSG
XmNtabStringDirection XmCTabStringDirection	XmSTRING_DIRECTION_DEFAULT unsigned char	CSG

XmNfreeTabPixmap

Frees the pixmaps assigned to the **XmNtabLabelPixmap** resources when the widget is destroyed.

XmNtabAlignment

Specifies the alignment of the tab label. The following values are valid for this resource:

XmALIGNMENT_BEGINNING Aligns label to left side of the available space.
G

XmALIGNMENT_CENTER Aligns label in center of the available space.

XmALIGNMENT_END Aligns label to right side of available space.

XmNtabBackground

Specifies the background color for the tab.

XmNtabBackgroundPixmap

Specifies the background pixmap for the tab.

XmNtabForeground

Specifies the foreground color for the tab.

XmNtabLabelPixmap

Specifies the Pixmap to display in the tab label.

XmNtabLabelString

The XmString to display as the textual portion of the tab label. This is copied when set on the widget. The value fetched by XtGetValues should not be freed, because it returns a pointer to the widget's value. If you want to use the value returned from XtGetValues, use XmStringCopy to copy the value.

XmNtabPixmapPlacement

Specifies the location of the pixmap with respect to the text in the tab label. The following values are valid for this resource:

XiPIXMAP_TOP	Places the pixmap above the XmString.
XiPIXMAP_BOTTOM	Places the pixmap below the XmString.
XiPIXMAP_RIGHT	Places the pixmap to the right of the XmString.
XiPIXMAP_LEFT	Places the pixmap to the left of the XmString.
XiPIXMAP_ONLY	Displays the pixmap portion only.
XiPIXMAP_NONE	Displays the XmString portion only.

XmNtabStringDirection

Specifies the string direction for the XmString portion of the tab label. The following values are valid for this resource:

XmSTRING_DIRECTION_L_TO_R	Left to right.
XmSTRING_DIRECTION_R_TO_L	Right to left.

Translations and Actions

TabStack includes the translations from the XmManager.

XiTabStackCallbackStruct Structure

A pointer to the following structure is passed to each callback as `client_data`:

```
typedef struct_XiTabStackCallbackStruct{
    int reason;
    XEvent *event;
    Widget selected_child;
} XiTabStackCallbackStruct;
```

reason Indicates why the callback was invoked. Valid callback reasons include `XiCR_TAB_SELECTED`, which indicates that a child became the selected widget.

event The XEvent that triggered the callback.

selected_child The widget ID of the selected child.

Convenience Routines

XiTabStackGetSelectedTab

Returns the widget ID of the currently selected tab.

```
Widget XiTabStackGetSelectedTab(Widget tab_stack)
    tab_stack          The TabStack widget.
```

XiTabStackSelectTab

Sets the currently displayed child of the TabStack.

```
void XiTabStackSelectTab(Widget tab, Boolean notify)
```

tab The child of the TabStack to be selected.

notify When True, **XmNtabSelectedCallback** will be called as usual in response to the change of state. Set False to suppress this callback.

XiTabStackXYToWidget

Converts an X/Y pixel coordinate (in the TabStack's window) to the widget ID of the tab occupying that space.

```
Widget XiTabStackXYToWidget(Widget tab_stack,
                             int x,
                             int y)
```

tab_stack The TabStack widget.

x, y The pixel coordinates to convert.

XiToolbar

UNIX Application Header File	<code>Xi/Toolbar.h</code>
UNIX Class Header File	<code>Xi/ToolbarP.h</code>
Class Name	<code>XiToolbar</code>
Class Pointer	<code>xiToolbarWidgetClass</code>
Superclass Name	<code>XmManager</code>
Creation Routine	<code>XiCreateToolbar</code>

The Toolbar widget manages groups of child widgets in a single row or column. Any type of widget can be a Toolbar item, but the most common type of widget is usually the `XiIconButton`. You can group a set of child widgets by setting the **XmNtoolbarGroup** constraint resource on each item. You can specify the order of items within a group by setting the **XmNgroupPosition** constraint resource on each item.

Toolbar Popup Labels

The Toolbar widget supports the display of a popup label over each child widget. The text in this label is specified in the **XmNentryLabelString** constraint resource. When the pointer first enters a Toolbar item, the popup label is shown after a delay specified in the **XmNpopupDelay** resource. During subsequent movement of the pointer within the Toolbar widget, popup labels are displayed with no delay. If the pointer leaves the Toolbar widget, the delay again becomes active.

Two callback lists, **XmNenterChildCallback** and **XmNleaveChildCallback**, are available for programmers who wish to be notified when the pointer enters and leaves Toolbar children. Figure 24 shows a typical Toolbar:



Figure 24. Toolbar Widget

Specifying Groups and Positions

Numbering of groups begins at 0 (the default), with no upper limit. Group numbers can be skipped; no extra space is left for empty groups. Numbering of items within groups begins at 0, with no upper limit. Position numbers can be skipped; empty space is left for each empty position. If you assign a position already filled with an item, Toolbar positions the item at the next unfilled position.

Classes and Inherited Resources

Toolbar inherits behavior and resources from **Core**, **Composite**, **Constraint**, and **XmManager**. Refer to “*Appendix*” on page 153 for a list of resources inherited from each Motif superclass. Refer to the *OSF/Motif Programmer’s Reference* for a complete description of each resource.

Resources

Name Class	Default Type	Access
XmNenterChildCallback XmCCallback	NULL XtCallbackList	CSG
XmNgroupSpacing XmCGroupSpacing	0 Dimension	CSG
XmNhorizontalMargin XmCMargin	0 Dimension	CSG
XmNleaveChildCallback XmCCallback	NULL XtCallbackList	CSG
XmNorientation XmCOrientation	XmHORIZONTAL unsigned char	CSG
XmNpopupBackground XmCBackground	dynamic Pixel	CSG
XmNpopupDelay XmCPopupDelay	500 Cardinal	CSG
XmNpopupFontList XmCFontList	dynamic XmFontList	CSG
XmNpopupForeground XmCForeground	dynamic Pixel	CSG
XmNpopupLabelEnabled XmCPopupLabelEnabled	False Boolean	CSG
XmNverticalMargin XmCMargin	0 Dimension	CSG

XmNenterChildCallback

Specifies the list of callbacks called when the pointer enters a Toolbar item. The same callback is set on all the Toolbar items, so the programmer must verify the item of interest, using the `XiToolbarCallbackStruct` that is passed as `call_data`.

XmNgroupSpacing

Specifies the number of pixels between groups. Spacing is vertical if **XmNOrientation** is `XmVERTICAL`, and horizontal if **XmNOrientation** is `XmHORIZONTAL`.

XmNhorizontalMargin

Specifies the horizontal spacing between the Toolbar items and the edge of the Toolbar.

XmNleaveChildCallback

Specifies the list of callbacks called when the pointer leaves a Toolbar item. The same callback is set on all the Toolbar items, so the programmer must verify the item of interest, using the `XiToolbarCallbackStruct` that is passed as `call_data`.

XmNOrientation

Specifies whether children are to be placed in a row or a column. The orientation can be either `XmHORIZONTAL` or `XmVERTICAL`. If the orientation is `XmHORIZONTAL`, the children are placed in a row with the major dimension being width and the minor dimension being height. If the value is `XmVERTICAL`, the children are placed in a column with the major dimension being height and the minor dimension being width. The default value is `XmHORIZONTAL`.

XmNpopupBackground

Specifies the background color for the popup label.

XmNpopupDelay

Specifies the amount of time, in msec, the Toolbar delays displaying the popup label when the pointer first enters a Toolbar item. If the pointer leaves the item before that amount of time, the label does not pop up.

XmNpopupFontList

Specifies the font used for the all text in the popup labels.

XmNpopupForeground

Specifies the foreground color for the popup label.

XmNpopupLabelEnabled

Specifies whether the Toolbar displays popup labels for each item.

XmNverticalMargin

Specifies the vertical spacing between the Toolbar items and the edge of the Toolbar.

Constraint Resources

Name Class	Default Type	Access
XmNentryLabelString XmCEntryLabelString	NULL XmString	CSG
XmNgroupPosition XmCGroupPosition	0 unsigned char	CSG
XmNtoolbarGroup XmCToolbarGroup	0 unsigned char	CSG
XmNtoolbarEntryData XmCToolbarEntryData	NULL XtPointer	CSG

XmNentryLabelString

Specifies the label displayed in the popup label for this item.

XmNgroupPosition

Specifies the position within the group for this item.

XmNtoolbarGroup

Specifies the group within the Toolbar for this item.

XmNtoolbarEntryData

Allows the application to attach any necessary application-specific data to the widget. This resource is not used internally by EnhancementPak.

XiToolbarCallbackStruct Structure

A pointer to the following structure is passed to each callback as `client_data`.

```
typedef struct _XiToolbarCallbackStruct {
    int reason;
    XEvent *event;
    Widget toolbar_item;
    unsigned char position;
    unsigned char group;
}XiToolbarCallbackStruct, *XiToolbarCallbackPtr;
```

<i>reason</i>	Either <code>XiCR_ENTER_CHILD</code> or <code>XiCR_LEAVE_CHILD</code> depending on which callback is invoked.
<i>event</i>	<code>XEvent</code> that caused the callback.
<i>toolbar_item</i>	Relevant child of the Toolbar.
<i>position</i>	Position within the group for this item.
<i>group</i>	Group within the Toolbar for this item.

Convenience Routines

XiToolbarMapGroup()

Maps all the widgets of a particular group.

```
void XiToolbarMapGroup(Widget tbar,
    unsigned char group_num)
```

<i>tbar</i>	Parent of the Toolbar widget.
<i>group_num</i>	Number of the group to map.

XiToolbarUnmapGroup()

Unmaps all the widgets of a particular group.

```
void XiToolbarUnmapGroup(Widget tbar,
    unsigned char group_num)
```

<i>tbar</i>	Parent of the Toolbar widget.
<i>group_num</i>	Number of the group to unmap.

XiToolBarManageGroup()

Manages all the widgets of a particular group.

```
void XiToolBarManageGroup (Widget tbar,  
                           unsigned char group_num)
```

tbar Parent of the Toolbar widget.

group_num Number of the group to manage.

XiToolBarUnmanageGroup()

Unmanages all the widgets of a particular group.

```
void XiToolBarUnmanageGroup (Widget tbar,  
                             unsigned char group_num)
```

tbar Parent of the Toolbar widget.

group_num Number of the group to unmanage.

XiToolBarDestroyGroup()

Destroys all the widgets of a particular group.

```
void XiToolBarDestroyGroup (Widget tbar,  
                            unsigned char group_num)
```

tbar Parent of the Toolbar widget.

group_num Number of the group to destroy.

XiToolTip

UNIX Application Header File	Xi/ToolTip.h
Class Name	XiToolTip

The ToolTip is a library of functions that provides a generalized interface to add time-delayed, mouse activated "ToolTip" functionality to any X Toolkit widget. For any widget so enabled, the ToolTip library creates a "virtual XmLabel" to serve as the popped up tip. This "virtual widget" appears to the Xt resource manager to be a child widget for the real widget on top of which it appears. Its resources can be set in app-default files, or programmatically through the XiToolTipSetValues/XiToolTipGetValues functions. The widget's name appears as "toolTip" in app-default processing, with a class name of XiToolTip. Thus, a possible app-default configuration of the ToolTips for a toolbar might look like:

```
MyApp*toolBar*fileOpenButton.toolTip.labelString: Open Document
MyApp*toolBar*fileCloseButton.toolTip.labelString: Close Document
MyApp*toolBar*fileOpenButton.toolTip.labelString: Make New Document
```

Figure 25 shows a ToolTip widget used to enhance a typical data entry screen:

The figure consists of two side-by-side screenshots of a data entry form. The form has the following fields:

- Name: JCS
- Address: 201 Broadway
- 5th Floor
- City: Cambridge
- State: MA
- Zip Code: [empty]
- [Phone icon] +1 617 621 0060

In the right screenshot, a tooltip is displayed over the Zip Code field, containing the text "Zip or Zip+4".

Figure 25. ToolTip Widget in a Data Entry Screen

Resources

Name Class	Default Type	Access
XmNtipEnabled XmCtipEnabled	Boolean False	CSG
XmNtipXOffset XmCTipXOffset	Dimension 10	CSG
XmNtipYOffset XmCTipYOffset	Dimension 10	CSG
XmNunpostDelay XmCUnpostDelay	0 Cardinal	CSG

XmNtipEnabled

Controls whether or not the tip appears when the pointer enters the widget.

XmNtipXOffset

XmNtipYOffset

Specifies the offset, in pixels, from the current mouse position to the location where the ToolTip shell appears.

Note: Because of the implementation of the ToolTip mechanism, **XmNtipXOffset** and **XmNtipYOffset** are always in pixels, regardless of the **XmNunitType** of the interface.

Note: These resources cannot be set using `XtSetValues()` and related functions. You must use `XiToolTipSetValues` and `XiToolTipGetValues` (which provide a functionally equivalent interface) to set and retrieve these values. Processing in the `app-defaults` file, however, works as for normal widgets.

XmNunpostDelay

Specifies the amount of time, in milliseconds, to wait before unposting the Tip. A value of 0 indicates that the Tip should never be unposted.

The following resources defined in XmLabel can also be set on ToolTip widgets (behave in the same as in the XmLabel widget class):

Name Class	Default Type	Access
XmNbackground XmCBackground	#dada80 Pixel	CSG
XmNborderColor XmCBorderColor	dynamic Pixel	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNfontList XmCFontList	dynamic XmFontList	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCLabelString	dynamic XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNmarginHeight XmCMarginHeight	2 Dimension	CSG
XmNmarginWidth XmCMarginWidth	2 Dimension	CSG

Convenience Routines

The `ToolTip` library provides a complete set of API routines for creating `ToolTip`s for widgets and for manipulating those `ToolTip`s. There are several functions that can always be used for the `ToolTip`s—functions such as `XiToolTipEnable()` and `XiToolTipDisable()`, which determine whether the `ToolTip` for a widget will be displayed, and `XiToolTipSetValues()` and `XiToolTipGetValues()`, which set and retrieve resource values on the `ToolTip`s.

Creating ToolTip

`ToolTip`s for widgets can be created in two ways.

- The function `XiToolTipRegister()` can be used at any time to create a `ToolTip` for a particular widget that has already been created.
- The `ToolTip` library can be set up to automatically create and register `ToolTip`s for widgets that are newly created. After an initial call to `XiToolTipInitialize()`, newly-created widgets are automatically created with `ToolTip` functionality. Thereafter, this automatic addition of `ToolTip` functionality can be temporarily halted by using `XiToolTipSuspend()` and then restored with `XiToolTipResume()`.

Regardless of which functions are used to create the `ToolTip` initially, `XiToolTipDestroyAll()` removes all `ToolTip` functionality from widgets in use.

`XiToolTipAddCallback()`

The `ToolTip` library provides an `XmNtipCallback` list which, for convenience, is used in the same way as normal `Xt` callbacks. Note that this resource is not a conventional `XtCallbackList` and thus cannot be set manually using `XiToolTipSetValues`, nor can it be set using `XtAddCallback()` (which would try to add the callback to the real widget, not the callback). The `XmNtipCallback` is called immediately before the `ToolTip` is popped up.

```
void XiToolTipAddCallback (Widget w,  
                          String name,  
                          XtCallbackProc f,  
                          XtPointer client_data)
```

<i>w</i>	The widget augmented with <code>ToolTip</code> functionality.
<i>name</i>	The name of the callback procedure.
<i>f</i>	The callback procedure to be added.
<i>client_data</i>	The data passed to the added procedure when it is invoked.

XiToolTipHideTip()

If a ToolTip widget is currently showing for widget *w*, XiToolTipHideTip pops it down unconditionally. If no ToolTip is showing or no ToolTip is registered, XiToolTipHideTip has no effect.

```
void XiToolTipHideTip (Widget w)
```

w

The widget augmented with ToolTip functionality

XiToolTipInitialize()

Initializes the global ToolTip environment. After this function is called, all new widgets created inherit ToolTip functionality by default, just as if XiToolTipRegister() had been called for them. Typically, this function would be called immediately following XtAppInitialize()—before any widgets are created.

Note: By default the ToolTips are created disabled. Automatically created ToolTips must be enabled either by setting the **XmNtipEnabled** resource in the app-defaults file or by an explicit call to XiToolTipEnable(). A useful way to do this through the app-default file might be as follows:

```
MyApp*toolBar.?.toolTip.tipEnabled: True
```

Also note that while it is the simplest and most accessible method for using ICS ToolTips, calling XiToolTipInitialize() adds a small space and performance overhead to the creation of new widgets. While it has been our experience that such overhead is negligible even for very large applications, applications that create widgets from within performance-critical code might better use XiToolTipRegister() explicitly for the widgets that need ToolTips.

```
void XiToolTipInitialize ()
```

XiToolTipRegister()

Registers widget *w* for ToolTip functionality, setting ToolTip arguments on it.

Note: This function merely adds the ToolTip behavior, it does not enable the ToolTip by default. That must be done through the `tipEnabled` resource in the app-defaults or `XiToolTipSetValues`, or by using `XiToolTipEnable()`.

```
void XiToolTipRegister(Widget w, ArgList args, Cardinal nargs)
```

<i>w</i>	The widget augmented with ToolTip functionality.
<i>args</i>	The argument list.
<i>nargs</i>	The number of attribute/value pairs in the argument list.

XiToolTipRemoveAllCallbacks()

Removes all registered ToolTip callbacks on widget *w*.

```
void XiToolTipRemoveAllCallbacks(Widget w, String name)
```

<i>w</i>	The widget augmented with ToolTip functionality.
<i>name</i>	The name of the callback procedure.

XiToolTipRemoveCallback()

Removes a callback added using `XiToolTipAddCallback`. As with `XtRemoveCallback`, both the callback procedure `f` and the closure data `client_data` must match.

```
void XiToolTipRemoveCallback(Widget w,  
                             String name,  
                             XtCallbackProc f,  
                             XtPointer client_data)
```

<i>w</i>	The widget augmented with ToolTip functionality.
<i>name</i>	The name of the callback procedure.
<i>f</i>	The callback procedure to be removed.
<i>client_data</i>	The data passed to the removed procedure.

XiToolTipResume()

XiToolTipResume() can be called after XiToolTipInitialize() and XiToolTipSuspend() to resume adding ToolTips for widgets that are newly created. Use of this function, along with XiToolTipSuspend(), lets the application programmer choose exactly which widgets are created with the ToolTips.

```
void XiToolTipResume()
```

XiToolTipSetValues()

XiToolTipSetValues behaves identically to XtSetValues except that it is used to set the ToolTip resources. All ToolTip resources (both ToolTip-specific and the XmLabel resources for the ToolTip widget) must be set using this function, and only resources specifiable for the ToolTip are affected.

```
void XiToolTipSetValues(Widget w,  
                        ArgList args,  
                        Cardinal nargs)
```

w The widget augmented with ToolTip functionality.

args The argument list.

nargs The number of attribute/value pairs in the argument list.

XiToolTipShowTip()

Pops up the ToolTip associated with the widget *w*. If another ToolTip for a widget within the same top-level shell is already showing, it is first popped down (only one ToolTip shell widget is allowed per top-level shell in the application). If the widget *w* has not been registered with the ToolTip system through either XiToolTipInitialize() or XiToolTipRegister(), this function has no effect. Note that the ToolTip shell pops up irrespective of the value of the tipEnabled resource. **XmNtipEnabled** controls only the automatic (that is, mouse movement initiated) popup of ToolTips.

```
void XiToolTipShowTip(Widget w)
```

w The widget augmented with ToolTip functionality.

XiToolTipSuspend()

XiToolTipSuspend() can be called after XiToolTipInitialize() to temporarily stop adding ToolTips for widgets that are newly created. Use of this function, along with XiToolTipResume(), lets the application programmer choose exactly which widgets are created with the ToolTips.

```
void XiToolTipSuspend()
```

XiTree

UNIX Application Header File	<code>Xi/Tree.h</code>
UNIX Class Header File	<code>Xi/TreeP.h</code>
Class Name	<code>XiTree</code>
Class Pointer	<code>xiTreeWidgetClass</code>
Superclass Name	<code>XiHierarchy</code>
Creation Routine	<code>XiCreateTree</code>

The Tree widget is a container that shows the relationship of its children in a tree-like format. Each child of the Tree widget is a node in the Tree. The hierarchy of nodes is created by specifying the Tree parent of each node as a constraint resource. If a node's parent is NULL, it is assumed to be a root of the Tree. Although each widget can have only one parent, the Tree widget supports adding more than one root node to a single Tree.

Note: The Tree widget assumes that it will be totally responsible for mapping and unmapping its children. Therefore no child of this widget should ever modify its **XmMappedWhenManaged** resource. If a child does modify this resource, the behavior is undefined.

Refer to “*XiHierarchy*” on page 66 for more information, including the convenience routines for *XiHierarchy* which can be used for the *XiTree*.

Figures 26 and 27 show two different forms of Tree widget:

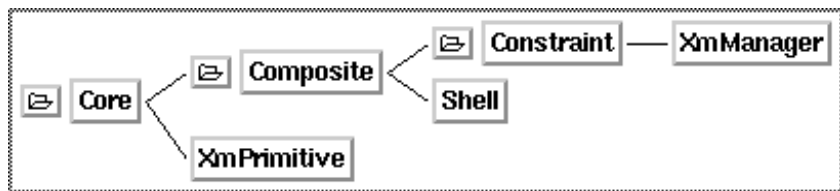


Figure 26. Tree Widget with *XmNconnectStyle* Set to *XiTreeDirect* and *XmNorientation* Set to *XmHORIZONTAL*

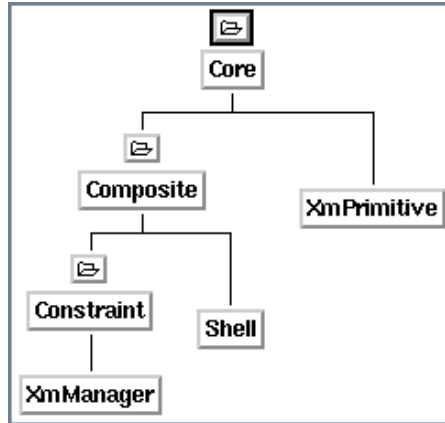


Figure 27. Tree Widget with `XmNconnectStyle` Set to `XiTreeLadder` and `XmNorientation` Set to `XmVERTICAL`

User Interaction

Each node in the Tree can be in four states: `XiOpen`, `XiClosed`, `XiAlwaysOpen`, or `XiHidden`. The state of a node changes how it appears to the user and what actions are available to the user.

XiOpen

This node has an open folder shown to its left; clicking on it closes the node. When a node is open, all of its children are visible.

XiClosed

This node has a closed folder shown to its left; clicking on it opens the node. When a node is closed, none of its children are visible.

XiAlwaysOpen

This node has no folder button associated with it. All of its children are visible.

Note: To maintain consistency of the user interface, it is best to use the node state `XiAlwaysOpen` for nodes with no children. This way the user will only see a folder button next to a node that has children to display. A folder button associated with a node that has no children has no defined behavior.

XiHidden

This node is not visible. All of its children appear and behave exactly as if they were children of the node's parent.

Geometry Management

The layout is performed by assigning each node a box that is just large enough to contain itself and all of its children. A recursive layout then places the node and the children. For an XmHORIZONTAL orientation, for example, the layout centers each node vertically in its box and at the extreme left horizontally. The children's boxes are placed to the right of the node separated by the **XmNhorizontalNodeSpace** from their parent, stacked above each other, and separated by **XmNverticalNodeSpace**. This process is repeated recursively for each child in the Tree.

The preferred size of the entire Tree will be just large enough to contain all nodes in the hierarchy. As the state of nodes change, the Tree will attempt to resize itself to contain its current configuration. If the Tree is forced larger than the desired size, the nodes will be centered vertically and flush to the left edge of the Tree widget (for a Tree with XmHORIZONTAL orientation). If the Tree is forced smaller, some nodes may be moved or drawn outside the end of the Tree. For this reason it is usually advisable to put the Tree into a Scrolled Window or Porthole widget.

Classes and Inherited Resources

Tree inherits behavior and resources from **Core**, **Composite**, **Constraint**, **XmManager**, and **XiHierarchy**.

Refer to “Appendix” on page 153 for a list of resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer's Reference*.

Resources

Name Class	Default Type	Access
XmNcompressStyle XmCCompressStyle	XiTreeCompressLeaves XiTreeCompressStyle	CSG
XmNconnectStyle XmCConnectStyle	XiTreeDirect XiTreeConnectStyle	CSG
XmNhorizontalDelta XmCHorizontalDelta	25 Dimension	CSG
XmNhorizontalNodeSpace XmCDimension	dynamic Dimension	CSG
XmNorientation XmCOrientation	XmHORIZONTAL unsigned char	CSG
XmNverticalDelta XmCVerticalDelta	30 Dimension	CSG
XmNverticalNodeSpace XmCDimension	dynamic Dimension	CSG

XmNcompressStyle

The XiTree can "compress" its children by staggering their placements when laying them out.

- When set to XiTreeCompressAll, all children off all nodes are alternately offset.
- When set to XiTreeCompressLeaves, only the leaves (nodes with no children) are treated in this manner.
- When set to XiTreeCompressNone, no nodes are treated in this manner.

This resource is valid only when **XmNorientation** is XmVERTICAL.

XmNconnectStyle

Specifies the style of connection lines, either XiTreeDirect or XiTreeLadder.

XmNhorizontalDelta

XmNverticalDelta

Specifies the vertical and horizontal offsets with which to alternate children when **XmNcompressStyle** is set to either XiTreeCompressLeaves or XiTreeCompressAll.

This resource is valid only when **XmNorientation** is XmVERTICAL.

XmNhorizontalNodeSpace**XmNverticalNodeSpace**

Specifies the number of pixels between each node in the Tree and its parent. The default is 20 for the direction of orientation and 2 otherwise.

XmNorientation

Specifies the orientation of the XiTree widget. When set to XmVERTICAL, the tree is displayed as a vertical tree with the root node at the top center of the widget.

Constraint Resources

Name Class	Default Type	Access
XmNlineBackgroundColor XmCBackground	dynamic Pixel	CSG
XmNlineColor XmCForeground	dynamic Pixel	CSG
XmNlineStyle XmCLineStyle	LineSolid int	CSG
XmNlineWidth XmCLineWidth	0 Dimension	CSG
XmNopenClosePadding XmCOpenClosePadding	0 int	CSG

XmNlineBackgroundColor

Specifies the background color of the line connecting a node to its parent. The default is the **XmNbackground** color of the Tree widget. This resource is effective only when the value of **XmNlineStyle** is LineDoubleDash.

XmNlineColor

Specifies the color of the line connecting a node to its parent. The default value for this resource is the **XmNforeground** color of the Tree widget.

XmNLineStyle

Specifies the style used to draw lines to the particular node. The valid values are those that are valid for the "line_style" element in a GC (graphics context)—LineSolid, LineOnOffDash, and LineDoubleDash. A string converter has been registered for quoted string equivalents of those values.

XmNLineWidth

Specifies the width of the connection line between a node and its parent.

XmNopenClosePadding

Specifies the number of pixels between the folder button and the node it is associated with.

Appendix



Classes and Inherited Resources

Many of the EnhancementPak widgets inherit behavior and resources from **Core**, **Composite**, **Constraint**, **XmBulletinBoard**, **XmFrame**, **XmManager**, **XmPrimitive**, and **XmTextField**.

This appendix lists the resources inherited from each Motif superclass. For a complete description of each resource, refer to the *OSF/Motif Programmer's Reference*.

Core Resources

CORE RESOURCE SET		
Name Class	Default Type	Access
XmNaccelerators XmCAccelerators	dynamic XtAccelerators	N/A
XmNancestorSensitive XmCSensitive	dynamic Boolean	G
XmNbackground XmCBackground	dynamic Pixel	CSG
XmNbackgroundPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderColor XmCBorderColor	XtDefaultForeground Pixel	CSG
XmNborderPixmap XmCPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNborderWidth XmCBorderWidth	1 Dimension	CSG
XmNcolormap XmCColormap	dynamic Colormap	CG
XmNdepth XmCDepth	dynamic int	CG
XmNdestroyCallback XmCCallback	NULL XtCallbackList	C

CORE RESOURCE SET (continued)		
Name Class	Default Type	Access
XmNheight XmCHeight	dynamic Dimension	CSG
XmNinitialResourcesPersistent XmCInitialResourcesPersistent	True Boolean	C
XmNmappedWhenManaged XmCMappedWhenManaged	True Boolean	CSG
XmNscreen XmCScreen	dynamic Screen*	CG
XmNsensitive XmCSensitive	True Boolean	CSG
XmNtranslations XmCTranslations	dynamic XtTranslations	CSG
XmNwidth XmCWidth	dynamic Dimension	CSG
XmNx XmCPosition	0 Position	CSG
XmNy XmCPosition	0 Position	CSG

Composite Resources

COMPOSITE RESOURCE SET		
Name Class	Default Type	Access
XmNchildren XmCReadOnly	NULL WidgetList	G
XmNinsertPosition XmCInsertPosition	NULL XtOrderProc	CSG
XmNnumChildren XmCReadOnly	0 Cardinal	G

XmBulletinBoard Resources

XMBULLETINBOARD RESOURCE SET		
Name Class	Default Type	Access
XmNallowOverlap XmCAllowOverlap	True Boolean	CSG
XmNautoUnmanage XmCAutoUnmanage	False Boolean	CG
XmNbuttonFontList XmCButtonFontList	dynamic XmFontList	CSG
XmNcancelButton XmCWidget	NULL Widget	SG
XmNdefaultButton XmCWidget	NULL Widget	SG
XmNdefaultPosition XmCDefaultPosition	True Boolean	CSG
XmNdialogStyle XmCDialogStyle	dynamic unsigned char	CSG
XmNdialogTitle XmCDialogTitle	NULL XmString	CSG
XmNfocusCallback XmCCallback	NULL XtCallbackList	C

XMBULLETINBOARD RESOURCE SET(continued)		
Name Class	Default Type	Access
XmNlabelFontList XmCLabelFontList	dynamic XmFontList	CSG
XmNmapCallback XmCCallback	NULL XtCallbackList	C
XmNmarginHeight XmCMarginHeight	10 Dimension	CSG
XmNmarginWidth XmCMarginWidth	10 Dimension	CSG
XmNnoResize XmCNoResize	False Boolean	CSG
XmNresizePolicy XmCResizePolicy	XmRESIZE_ANY unsigned char	CSG
XmNshadowType XmCShadowType	XmSHADOW_OUT unsigned char	N/A
XmNtextFontList XmCTextFontList	dynamic XmFontList	CSG
XmNtextTranslations XmCTextTranslations	NULL XtTranslations	C
XmNunmapCallback XmCCallback	NULL XtCallbackList	C

XmFrame Resources

XMFRAME RESOURCE SET		
Name Class	Default Type	Access
XmNmarginWidth XmCMarginWidth	0 Dimension	CSG
XmNmarginHeight XmCMarginHeight	0 Dimension	CSG
XmNshadowType XmCShadowType	dynamic unsigned char	CSG

XmManager Resources

XMMANAGER RESOURCE SET		
Name Class	Default Type	Access
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixmap	CSG
XmNhighlightPixmap XmCHighLightPixmap	dynamic Pixmap	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmTAB_GROUP XmNavigationType	CSG
XmNshadowThickness XmCShadowThickness	0 Dimension	CSG

XMMANAGER RESOURCE SET (continued)		
Name Class	Default Type	Access
XmNstringDirection XmCStringDirection	dynamic XmStringDirection	CG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG
XmNunitType XmCUnitType	dynamic unsigned Char	CSG
XmNuserData XmCUserData	NULL XtPointer	CSG

XmPrimitive Resources

XmPrimitive Resource Set		
Name Class	Default Type	Access
XmNbottomShadowColor XmCBottomShadowColor	dynamic Pixel	CSG
XmNbottomShadowPixmap XmCBottomShadowPixmap	dynamic Pixmap	CSG
XmNforeground XmCForeground	dynamic Pixel	CSG
XmNhelpCallback XmCCallback	NULL XtCallbackList	C
XmNhighlightColor XmCHighlightColor	dynamic Pixel	CSG
XmNhighlightOnEnter XmCHighlightOn Enter	False Boolean	CSG
XmNhighlightPixmap XmCHighlightPixmap	dynamic Pixmap	CSG
XmNhighlightThickness XmCHighlightThickness	2 Dimension	CSG
XmNinitialFocus XmCInitialFocus	NULL Widget	CSG
XmNnavigationType XmCNavigationType	XmNONE XmNavigation	CG
XmNshadowThickness XmCShadowThickness	2 Dimension	CSG
XmNtopShadowColor XmCTopShadowColor	dynamic Pixel	CSG
XmNtopShadowPixmap XmCTopShadowPixmap	dynamic Pixmap	CSG
XmNtraversalOn XmCTraversalOn	True Boolean	CSG

XmTextField Resources

XMTEXTFIELD RESOURCE SET		
Name Class	Default Type	Access
XmNXmNactivateCallback XmCXmCCallback	NULL XtCallbackList	C
XmNXmNblinkRate XmCXmCblinkRate	500 int	CSG
XmNXmNcolumns XmCXmCColumns	dynamic short	CSG
XmNXmNcursorPosition XmCXmCCursorPosition	0 XmTextPosition	CSG
XmNXmNcursorPositionVisible XmCXmCCursorPositionVisible	True Boolean	CSG
XmNXmNeditable XmCXmCEditable	True Boolean	CSG
XmNXmNfocusCallback XmCXmCCallback	NULL XtCallbackList	C
XmNXmNfontList XmCXmCFontList	dynamic XmFontList	CSG
XmNXmNgainPrimaryCallback XmCXmCCallback	NULL XtCallbackList	C
XmNXmNlosePrimaryCallback XmCXmCCallback	NULL XtCallbackList	C
XmNXmNlosingFocusCallback XmCXmCCallback	NULL XtCallbackList	C
XmNXmNmarginHeight XmCXmCMarginHeight	5 Dimension	CSG
XmNXmNmarginWidth XmCXmCMarginWidth	5 Dimension	CSG
XmNXmNmaxLength XmCXmCMaxLength	largest integer int	CSG
XmNXmNmodifyVerifyCallback XmCXmCCallback	NULL XtCallbackList	C

XMTEXTFIELD RESOURCE SET (continued)		
Name Class	Default Type	Access
XmNXmNmodifyVerifyCallbackWcs XmCXmCCallback	NULL XtCallbackList	C
XmNXmNmotionVerifyCallback XmCXmCCallback	NULL XtCallbackList	C
XmNXmNpendingDelete XmCXmCPendingDelete	True Boolean	CSG
XmNXmNresizeWidth XmCXmCResizeWidth	False Boolean	CSG
XmNXmNselectionArray XmCXmCSelectionArray	default array XtPointer	CSG
XmNXmNselectionArrayCount XmCXmCSelectionArrayCount	3 int	CSG
XmNXmNselectThreshold XmCXmCSelectThreshold	5 int	CSG
XmNXmNvalue XmCXmCValue	"" String	CSG
XmNXmNvalueChangedCallback XmCXmCCallback	NULL XtCallbackList	C
XmNXmNvalueWcs XmCXmCValueWcs	(wchar_t *)"" wchar_t *	CSG
XmNXmNverifyBell XmCXmCVerifyBell	dynamic Boolean	CSG

A

APPENDIX *XmTextField Resources*

Index

A

Actions

- Internationalized extended list widget 51

- Actions, button box widget 21

Always open node

- outline widget 85

- tree widget 147

B

Building applications

- UNIX 13

Button box widget

- actions 21

- classes 19, 27, 40

- inherited resources 19, 27, 40

- resources 19

- translations 21

- ButtonDown(type) 51

- ButtonUpOrLeave() 51

C

- call_data 9

Callback routines

- combination box widget 39

- hierarchy widget 70

- icon button widget 82

- internationalized extended list widget 54

- panner widget 101

- porthole widget 118

- stretch widget 122

- tab stack widget 131

- canvas 95

- Class methods, hierarchy widget 71

Classes

- button box widget 19, 27, 40

- color selector widget 23

- combination box widget 34

- font selector widget 60

- hierarchy widget 67

- icon box widget 74

- icon button widget 77, 78

- internationalized extended list widget 46

- outline widget 86

- paned widget 90

- panner widget 97

- pixmap editor widget 106

- porthole widget 117

- stretch widget 121

- tab stack widget 125

- toolbar widget 133

- tree widget 148, 149

classes

- icon box widget 75

Closed node

- outline widget 84

- tree widget 147

Color selector widget 22

- classes 23

- compound widget hierarchy 25

- inherited resources 23

- resources 23

- Colors, pixmap editor widget 104

Combination box widget 31

- callback routines 39

- classes 34

- compound widget hierarchy 38

- inherited resources 34

- resources 34

Index

- Composite resources 155
- Compound widget hierarchy
 - color selector 25
 - combination box 38
 - font selector widget 64
 - internationalized extended list 52
 - pixmap editor widget 111
- Compound widgets
 - constraint resources removed at creation 12
 - resources removed at creation 12
- Compound widgets, description 12
- Constraint resources
 - hierarchy widget 69
 - icon box widget 76
 - paned widget 92
 - removed at creation 12
 - tab stack widget 129
 - toolbar widget 135
 - tree widget 150
- Convenience routines
 - combination box widget 39
 - hierarchy widget 70
 - icon box widget 76
 - internationalized extended list widget 54
 - paned widget 94
 - panner widget 101
 - pixmap editor widget 113
 - porthole widget 119
 - toolbar widget 136
- Conventions, used in this document xi
- Copy, pixmap editor widget 104
- Core resources 153

D

- Draw circle, pixmap editor widget 103
- Draw line, pixmap editor widget 103
- Draw point, pixmap editor widget 103

E

- Encoding, font selector widget 59
- epak-talk mailing list xii
- Extended Internationalized List 44
- External symbols, description 12

F

- Filled circle, pixmap editor widget 103
- Filled rectangle, pixmap editor widget 103
- Flood fill, pixmap editor widget 104
- Font scaling, font selector widget 59
- Font selector widget
 - advanced features 58
 - basic features 57
 - classes 60
 - compound widget hierarchy 64
 - encoding 59
 - fixed width fonts 59
 - font scaling 59
 - inherited resources 60
 - non XLFD fonts 58
 - proportional fonts 59
 - resolution control 59
 - resources 60
 - XLFD name display 59

Fonts

- fixed, font selector widget 59
- proportional, font selector widget 59

G

- Geometry management 14
 - outline widget 85
 - paned widget 88
 - tree widget 148
- Groups, specifying with toolbar widget 133

H

- Help, for mailing list xiii
- Hidden node
 - outline widget 85
 - tree widget 148
- Hierarchy widget
 - build node table routine 72
 - callback routine 70
 - change node state routine 71
 - class methods 71
 - classes 66
 - constraint resources 69

- convenience routine 70
- inherited resources 66
- map node routine 71
- reset open closed button routine 72
- resources 67
- toggle node state routine 72
- unmap all extra nodes routine 71
- unmap node routine 71

I

Icon box widget

- classes 74
- constraint resources 76
- convenience routines 76
- inherited resources 74
- resources 75

Icon button widget

- callback routines 82
- classes 77
- resources 77, 78
- translations 81

Information, version 14

Inherited resources

- button box widget 19, 27, 40
- color selector widget 23
- combination box widget 34
- font selector widget 60
- hierarchy widget 66
- icon box widget 74
- icon button widget 77
- internationalized extended list widget 46
- outline widget 85
- paned widget 89
- panner widget 97
- pixmap editor widget 106
- porthole widget 116
- stretch widget 120
- tab stack widget 125
- toolbar widget 133
- tree widget 148

Input, pixmap editor widget 105

Internationalized extended list widget

- actions 51
- callback routine 54
- classes 46
- compound widget hierarchy 52
- convenience routines 54
- inherited resources 46
- resources 46
- translations 51
- using resource database with 46
- Xi18RowInfo structure 53

M

Mailing lists, epak-talk xii

Motif

- Epak resources inherited from 153
- superclasses 153

Motion() 51

Move, pixmap editor widget 104

N

Nodes

- always open, outline widget 85
- always open, tree widget 147
- build table routine, hierarchy widget 72
- change state routine, hierarchy widget 71
- closed, outline widget 84
- closed, tree widget 147
- hidden, outline widget 85
- hidden, tree widget 148
- map routine, hierarchy widget 71
- open, outline widget 84
- open, tree widget 147
- reset open closed button routine, hierarchy widget 72
- toggle state routine, hierarchy widget 72
- unmap all extra routine, hierarchy widget 71
- unmap routine, hierarchy widget 71

Non XLFD fonts, font selector widget 58

Notation, used in this document xi

O

Open node

Index

- outline widget 84
- tree widget 147
- Outline widget
 - always open node 85
 - classes 85
 - closed node 84
 - geometry management 85
 - hidden node 85
 - inherited resources 85
 - open node 84
- Output, pixmap editor widget 105
- P**
- Paned widget
 - classes 90
 - constraint resources 92
 - convenience routines 94
 - geometry management 88
 - inherited resources 89
 - resizing panes 88
 - search order 89
 - special considerations 89
 - translations 93
- Panner widget
 - callback routines 101
 - classes 97
 - convenience routines 101
 - inherited resources 97
 - resources 97
 - translations 99
- Panning, pixmap editor widget 105
- Pixmap editor widget
 - classes 106
 - colors, selecting and adding 104
 - compound widget hierarchy 111
 - convenience routines 113
 - copy 104
 - draw circle 103
 - draw line 103
 - draw point 103
 - filled circle 103
 - filled rectangle 103
 - flood fill 104
 - inherited resources 106
 - input 105
 - move 104
 - output 105
 - panning 105
 - rectangle 103
 - resizing pixmap 104
 - set all 103
 - translations 111
 - undo 103
 - zooming 105
- Pixmap, resizing 104
- Popup labels, toolbar widget 132
- Porthole widget
 - callback routine 118
 - classes 117
 - convenience routines 119
 - inherited resources 116
 - resources 117
- Positions, specifying with toolbar widget 133
- R**
- Rectangle, pixmap editor widget 103
- Resizing panes, paned widget 88
- Resolution control, font selector widget 59
- Resource database, using with internationalized
 - extended list widget 46
- Resource naming, description 12
- Resources
 - button box widget 19
 - color selector widget 23
 - composite 155
 - core 153
 - font selector widget 60
 - hierarchy widget 67
 - icon box widget 75
 - icon button widget 78
 - inherited, button box widget 19, 27, 40
 - inherited, hierarchy widget 66
 - internationalized extended list widget 46
 - porthole widget 117

- stretch widget 121
 - toolbar widget 133
 - tree widget 149
 - XmFrame 157
 - XmManager 157
 - XmNentryLabelAlignment 29
 - XmNentryLabelPixmap 29
 - XmNentryLabelString 29
 - XmNentryLabelType 30
 - XmNentryLabelFontList 29
 - XmNfillStyle 30
 - XmNpicture 41
 - XmNshowEntryLabel 30
 - XmNshowTitles 27
 - XmNstretchable 30
 - XmPrimitive 159
 - Resources, inherited
 - button box 19, 27, 40
 - color selector widget 23
 - combination box widget 34
 - font selector widget 60
 - stretch widget 120
 - toolbar widget 133
 - tree widget 148
 - Resources, panner widget 97
 - Routines
 - build node table, hierarchy widget 72
 - change node state, hierarchy widget 71
 - map node, hierarchy widget 71
 - reset open closed button, hierarchy widget 72
 - toggle node state, hierarchy widget 72
 - unmap all extra nodes, hierarchy widget 71
 - unmap node, hierarchy widget 71
- ## S
- SashAction(Commit) 94
 - SashAction(Key, Incr, Dir) 94
 - SashAction(Move) 94
 - SashAction(Start) 94
 - Search order, paned widget 89
 - Set all, pixmap editor widget 103
 - slider 95
 - Special considerations, paned widget 89
 - Stretch widget
 - callback routines 122
 - classes 121
 - inherited resources 120
 - resources 121
 - translations 122
 - Strings, version 14
 - Subscribing, to mailing lists xii
 - Superclasses, Motif 153
- ## T
- Tab stack widget
 - callback routines 131
 - classes 125
 - constraint resources 129
 - inherited resources 125
 - new resources 125
 - translations 130
 - templates
 - widget resources 4
 - Toolbar widget
 - classes 133
 - constraint resources 135
 - convenience routines 136
 - inherited resources 133
 - popup labels 132
 - resources 133
 - specifying groups 133
 - specifying positions 133

Index

Translations

- button box widget 21
- icon button widget 81
- Internationalized extended list widget 51
- paned widget 93
- panner widget 99
- pixmap editor widget 111
- stretch widget 122
- tab stack widget 130

Tree widget

- always open node 147
- classes 148, 149
- closed node 147
- constraint resources 150
- geometry management 148
- hidden node 148
- inherited resources 148
- open node 147
- resources 149
- user interaction 147

U

- Undo, pixmap editor widget 103
- Unsubscribing, mailing lists xii
- Using, resource database 46

V

- Version, information 14

W

- widget templates
 - resources 4

Widgets

- color selector, compound hierarchy 25
- combination box 31
- compound, description 12
- XiButtonBox 18
- XiColorSelector 22
- XiColumn 26
- XiCombinationBox 31
- XiDataField 40
- XiDBDateField 40
- XiExtended18List 44

- XiExtended18List, compound hierarchy 52
- XiFontSelector 57
- XiHierarchy 66
- XiIconBox 73
- XiIconButton 77
- XiOutline 83
- XiPaned 87
- XiPanner 95
- XiPixmapEditor 102
- XiPorthole 114
- XiStretch 120
- XiTabStack 124
- XiToolBar 132
- XiToolTip 138
- XiTree 146

X

- XCopArea 80
- XCopPlane 80
- Xi.h 14
- Xi18RowInfo structure 53
- Xi18SortFunction 54
- XiAbort() 99
- XiALIGNMENT_BEGINNING 27
- XiALIGNMENT_CENTER 27
- XiALIGNMENT_END 27
- XiALIGNMENT_UNSPECIFIED 27
- XiAlwaysOpen 69, 84, 147
- XiANY_COLUMN 54, 56
- XiArmAndActivate() 81
- XiButtonBox 6, 18
- xiButtonBoxWidgetClass 18
- XiButtonUp() 81
- XiClosed 69, 84, 147
- XiColorSelector 22
- xiColorSelectorWidgetClass 22
- XiColumn 26
- XiCombinationBox 31
- XiCombinationBoxGetValue() 39
- xiCombinationBoxWidgetClass 31
- XiComboCancel() 34
- XiComboListCancel 35

XiComboListDown() 33
 XiComboListPost 35
 XiComboListUnpost 35
 XiComboListUp() 34
 XiCR_TAB_SELECTED 131
 XiCR_UPDATE_SHELL 39
 XiCR_UPDATE_TEXT 39
 XiCR_VERIFY_TEXT 39
 XiDataField 40
 XiDBDataField 40
 XiDISTRIBUTE_SPREAD 28
 XiDISTRIBUTE_TIGHT 28
 XiEnd() 122
 XiEndPoint() 111
 XiEXT18LIST_FOUND 52
 XiEXT18LIST_NOT_FOUND 52
 XiExt18ListCallbackStruct 52
 XiExt18ListDeselectItems 54
 XiExt18ListDeselectRow 54
 XiExt18ListGetSelectedRowArray 55
 XiExt18ListGetSelectedRows() 55
 XiExt18ListMakeRowVisible 55
 XiExt18ListSelectAllItems 55
 XiExt18ListSelectItems 56
 XiExt18ListSelectRow 56
 XiExt18ListToggleRow() 56
 xiExt18ListWidgetClass 44
 XiExt18UnselectAllItems() 56
 XiExt18UnselectItems() 56
 XiExtended18List 44
 XiFillAll 21
 XiFillMajor 7, 20
 XiFillMinor 7, 20
 XiFillNone 20
 XiFontSelector 57
 xiFontSelectorWidgetClass 57
 XiGetFocus() 99
 XiHidden 69, 84, 147
 XiHierarchy 66
 XiHierarchyGetChildNodes 70
 XiHierarchyNodeStateData 68, 70
 XiHierarchyOpenAllAncestors 70
 XiHierarchyWidgetClass 66
 XilconBottom 79
 XilconBox 73
 XilconBoxAnyCell 76
 XilconBoxIsCellEmpty() 76
 xilconBoxWidgetClass 73
 XilconButton 77
 XilconButtonCallbackInfo 82
 xilconButtonWidgetClass 77
 XilconLeft 79
 XilconNone 79
 XilconOnly 79
 XilconRight 79
 XilconTop 79
 XiInheritBuildNodeTable 72
 XiInheritChangeNodeState 71
 XiInheritMapNode 71
 XiInheritResetOpenCloseButton 72
 XiInheritToggleNodeState 72
 XiInheritUnmapAllExtraNodes 71
 XiInheritUnmapNode 71
 XiMotion() 111, 122
 XiMove() 99
 XiNextColor() 111
 XiNotify() 81, 99
 XiOpen 69, 84, 147
 XiOutline 83
 xiOutlineWidgetClass 83
 XiPage() 100
 XiPaned 87
 XiPanedAskChild 92
 XiPanedGetPanels() 94
 xiPanedWidgetClass 87
 XiPanner 95
 xiPannerWidgetClass 95
 XiPIXMAP_BOTTOM 130
 XiPIXMAP_LEFT 130
 XiPIXMAP_NONE 130
 XiPIXMAP_ONLY 130
 XiPIXMAP_RIGHT 130
 XiPIXMAP_TOP 130
 XiPixmapEditor 102

Index

- XiPixmapEditorDrawableInfo 108
- XiPixmapEditorGetPictureData 113
- XiPixmapEditorGetRelevantData 113
- XiPixmapEditorRefresh 113
- XiPixmapEditorRefresh() 113
- xiPixmapEditorWidgetClass 102
- XiPorthole 114
- XiPortholeConnectPanner() 96, 101, 119
- XiPortholeVisible() 119
- xiPortholeWidgetClass 114
- XiPRCanvasHeight 100
- XiPRCanvasWidth 100
- XiPRSliderHeight 100
- XiPRSliderWidth 100
- XiPRSliderX 100
- XiPRSliderY 100
- XiScrollReport 100, 118
- XiSet() 100
- XiStart() 99, 122
- XiStartPoint() 111
- XiStop() 100
- XiStretch 120
- xiStretchWidgetClass 120
- XiStretchWidgetInfo 122, 123
- XiTABS_ORIENTATION_DYNAMIC 127
- XiTABS_BASIC 127
- XiTABS_BEVELED 128
- XiTABS_BOTTOM_TO_TOP 127, 128
- XiTABS_LEFT_TO_RIGHT 127, 128
- XiTABS_ON_BOTTOM 128
- XiTABS_ON_LEFT 128
- XiTABS_ON_RIGHT 128
- XiTABS_ON_TOP 128
- XiTABS_RIGHT_TO_LEFT 127, 128
- XiTABS_ROUNDED 128
- XiTABS_SQUARED 128
- XiTABS_STACKED 127
- XiTABS_STACKED_STATIC 127
- XiTABS_TOP_TO_BOTTOM 128
- XiTabStack 124
- XiTabStackCallbackStruct 131
- XiTabStackGetSelectedTab 131
- XiTabStackSelectTab 131
- xiTabStackWidgetClass 124
- XiTabStackXYToWidget 131
- XiToggle() 81
- XiToolbar 132
- XiToolbarCallbackStruct 134, 136
- XiToolbarDestroyGroup() 137
- XiToolbarManageGroup() 137
- XiToolbarMapGroup() 136
- XiToolbarUnmanageGroup() 137
- XiToolbarUnmapGroup() 136
- xiToolbarWidgetClass 132
- XiToolTip 138
- XiToolTipAddCallback() 141
- XiToolTipDestroyAll() 141, 142
- XiToolTipDisable() 141, 142
- XiToolTipEnable() 141, 142, 143, 144
- XiToolTipGetValues() 138, 139, 141, 142
- XiToolTipHideTip() 143
- XiToolTipInitialize() 141, 142, 143, 145
- XiToolTipRegister() 141, 143, 144
- XiToolTipRemoveAllCallbacks() 144
- XiToolTipRemoveCallback() 144
- XiToolTipResume() 141, 145
- XiToolTipSetValues() 138, 141, 142, 145
- XiToolTipShowTip() 145
- XiToolTipSuspend() 141, 145
- XiTree 146
- XiTreeCompressAll 149
- XiTreeCompressLeaves 149
- XiTreeCompressNone 149
- XiTreeDirect 146, 149
- XiTreeLadder 147, 149
- XiWarpTo() 100
- XLFD name display, font selector widget 59
- XmALIGNMENT_BEGINNING 48, 79, 129
- XmALIGNMENT_CENTER 48, 79, 129
- XmALIGNMENT_END 41, 48, 79, 129
- XmCMargin 7
- XmEXTENDED_SELECT 49, 50
- XmFrame resources 157

- XmHORIZONTAL 7, 28, 87, 89, 91, 94, 134, 146, 148
- XmLabel 13
- XmList 31
- XmManager 8
- XmManager resources 157
- XmN100DPIstring 61
- XmN75DPIstring 61
- XmNactivateCallback 77, 78, 81, 82
- XmNalignment 40, 41, 48, 79
- XmNallowResize 12, 92
- XmNanyLowerString 61
- XmNanyString 62
- XmNarmColor 79
- XmNautoClose 67
- XmNautoFill 41
- XmNbackground 12, 108, 116, 150
- XmNbitmapMode 107
- XmNblueSliderLabel 24
- XmNboldString 62
- XmNbothString 62
- XmNbottomAttachment 12
- XmNbottomOffset 12
- XmNbottomPosition 12
- XmNbottomWidget 12
- XmNcanvasHeight 97
- XmNcanvasWidth 97
- XmNcellHeight 107, 108
- XmNcellWidth 107, 108
- XmNcellX 74, 76
- XmNcellY 74, 76
- XmNchildren 94
- XmNcloseFolderPixmap 67, 69
- XmNcolorListTogLabel 24
- XmNcolorMode 24
- XmNcolumnName 24
- XmNcolumnTitles 48
- XmNcomboTranslations 33, 35
- XmNcompressStyle 149
- XmNconnectNodes 86
- XmNconnectStyle 146, 147, 149
- XmNconstrainWidth 86
- XmNcurrentColor 107
- XmNcurrentFont 62, 63
- XmNcursor 90
- XmNcustomizedCombinationBox 36, 38
- XmNcustomizedComboBox 37
- XmNdefaultEncodingString 62
- XmNdefaultEntryLabelAlignment 27, 29
- XmNdefaultEntryLabelFontList 28
- XmNdefaultFillStyle 28
- XmNdefaultScale 97, 98, 101
- XmNdistribution 28
- XmNdoubleClickCallback 45, 48, 54, 79, 81, 82
- XmNdrawable 105, 108, 109
- XmNdrawableInfo 102, 105, 108
- XmNeditable 36
- XmNeditCursor 108
- XmNencodingList 62
- XmNencodingString 62
- XmNenterChildCallback 132, 134
- XmNentryData 48
- XmNentryLabelAlignment 27, 29
- XmNentryLabelFontList 28, 29
- XmNentryLabelPixmap 29
- XmNentryLabelString 8, 29, 132, 135
- XmNentryLabelType 30
- XmNequalSize 19
- XmNfamilyString 62
- XmNfileReadError 24
- XmNfillOption 7, 19
- XmNfillStyle 28, 30
- XmNfindLabel 48
- XmNfirstColumn 48
- XmNfirstColumnPixmap 48, 53
- XmNfirstRow 48
- XmNfontList 49, 79, 126
- XmNforceChildToFill 116, 117
- XmNforeground 150
- XmNfreeTabPixmap 129
- XmNgreenSliderLabel 24
- XmNgridSize 108
- XmNgridTranslations 109, 111
- XmNgroupPosition 132, 135

Index

- XmNgroupSpacing 134
- XmNheight 12, 116
- XmNheightInc 121, 123
- XmNhighlightThickness 126
- XmNhorizontalDelta 149
- XmNhorizontalMargin 5, 36, 68, 75, 79, 134
- XmNhorizontalNodeSpace 148, 150
- XmNhorizontalScrollBar 49
- XmNiconPlacement 77
- XmNiconTextPadding 79
- XmNimage 105, 109
- XmNinsertBefore 69
- XmNinternalSpace 98
- XmNitalicString 62
- XmNitemCount 38
- XmNitemFoundCallback 49, 52
- XmNitemNotFoundCallback 49, 52
- XmNitems 38
- XmNitemSpacing 28
- XmNlabel 80
- XmNlabelFontList 28
- XmNlabelSpacing 28
- XmNlabelString 13, 39, 80
- XmNleaveChildCallback 132, 134
- XmNleftAttachment 12
- XmNleftOffset 12
- XmNleftPosition 12
- XmNleftWidget 12
- XmNlineBackgroundColor 150
- XmNlineColor 150
- XmNlineStyle 151
- XmNlineWidth 151
- XmNmappedWhenManaged 66, 84, 146
- XmNmargInHeight 7, 21, 24, 62, 90, 117
- XmNmargInWidth 2, 6, 7, 15, 21, 24, 90, 117
- XmNmaxHeight 109, 121
- XmNmaxWidth 109, 121
- XmNminHeight 109, 121
- XmNminimumCellHeight 75
- XmNminimumCellWidth 75
- XmNminimumHorizontalCells 75
- XmNminimumVerticalCells 75
- XmNminWidth 109, 121
- XmNmodified 109
- XmNmonoSpaceString 62
- XmNnavigationType 12
- XmNnewVisualStyle 36, 37, 49
- XmNnoCellError 24
- XmNnodeCloseFolderPixmap 67, 69
- XmNnodeOpenFolderPixmap 67, 69
- XmNnodeState 69, 70
- XmNnodeStateBeginEndCallback 68
- XmNnodeStateCallback 68, 70
- XmNnodeStateChangedCallback 68
- XmNnodeStates 84
- XmNnumChildren 94
- XmNnumColumns 48, 49, 53
- XmNnumRows 48, 49
- XmNopenClosePadding 151
- XmNopenFolderPixmap 67, 69
- XmNoptionString 63
- XmNorientation 7, 21, 28, 87, 89, 91, 94, 134, 146, 147, 149, 150
- XmNotherString 63
- XmNoverrideRedirect 36
- XmNpalette 107, 110
- XmNpaletteSize 110
- XmNpaneMaximum 12, 92
- XmNpaneMinimum 12, 92
- XmNparentNode 69, 70
- XmNpicture 40, 41, 42
- XmNpictureErrorCallback 42
- XmNpixmap 79, 80
- XmNpixmapHeight 105, 110
- XmNpixmapWidth 105, 110
- XmNpopupBackground 134
- XmNpopupCursor 36
- XmNpopupDelay 132, 134
- XmNpopupFontList 134
- XmNpopupForeground 134
- XmNpopupLabelEnabled 135
- XmNpopupOffset 33, 36
- XmNpopupShellWidget 36
- XmNpreferredPaneSize 12, 88, 92

- XmNpropSpaceString 63
- XmNrecomputeSize 80
- XmNredSliderLabel 24
- XmNrefigureMode 68, 91
- XmNreportCallback 98, 99, 101, 116, 117, 118
- XmNresize 98
- XmNresizeCallback 122
- XmNresizeToPreferred 12, 88, 92
- XmNrgbFile 25
- XmNrightAttachment 12
- XmNrightOffset 12
- XmNrightPosition 12
- XmNrightWidget 12
- XmNrubberBand 98, 99, 100
- XmNsampleText 63
- XmNsashHeight 91
- XmNsashIndent 91
- XmNsashShadowThickness 91
- XmNsashTranslations 91
- XmNsashWidth 91
- XmNscalingString 63
- XmNselectCallback 49
- XmNselectColor 128
- XmNselectedColumn 49
- XmNselectionPolicy 31, 39, 49, 50
- XmNselectPixmap 128
- XmNsensitive 12
- XmNseparatorOn 91
- XmNset 80
- XmNsetAllString 110
- XmNshadowThickness 37, 98, 117, 122
- XmNshowEntryLabel 30
- XmNshowFind 44, 50
- XmNshowFontName 63
- XmNshowLabel 37
- XmNshowNameString 63
- XmNshowNormalView 110
- XmNshowSash 12, 93
- XmNshowTitles 27
- XmNsingleSelectionCallback 49, 54, 55, 56
- XmNsizeString 63
- XmNskipAdjust 12, 88, 93
- XmNsliderHeight 98
- XmNsliderToggleLabel 25
- XmNsliderWidth 98
- XmNsliderX 98
- XmNsliderY 98
- XmNsortFunctions 50
- XmNspacing 63, 91
- XmNstackedEffect 126
- XmNstretchable 28, 30
- XmNSTRING_DIRECTION_L_TO_R 80
- XmNSTRING_DIRECTION_R_TO_L 80
- XmNstringDirection 50, 79, 80
- XmNtabAlignment 129
- XmNtabAutoSelect 126
- XmNtabBackground 129
- XmNtabBackgroundPixmap 130
- XmNtabCornerPercent 126
- XmNtabForeground 130
- XmNtabLabelPixmap 129, 130
- XmNtabLabelSpacing 126
- XmNtabLabelString 130
- XmNtabMarginHeight 126
- XmNtabMarginWidth 126
- XmNtabMode 127
- XmNtabOffset 127
- XmNtabOrientation 127
- XmNtabPixmapPlacement 130
- XmNtabSelectedCallback 128, 131
- XmNtabSide 127, 128
- XmNtabStringDirection 130
- XmNtabStyle 128
- XmNtextRows 63
- XmNtile 110
- XmNtipCallback 141
- XmNtipEnabled 139, 142, 143, 145
- XmNtipXOffset 139
- XmNtipYOffset 139
- XmNtitle 50
- XmNtitleLabelString 50
- XmNtoolbarEntryData 135
- XmNtoolbarGroup 132, 135
- XmNtopAttachment 12

Index

XmNtopOffset 12
XmNtopPosition 12
XmNtopWidget 12
XmNundoString 110
XmNuniformTabSize 128
XmNunitType 15, 139
XmNunpostDelay 140
XmNupdateShellCallback 36, 37, 39
XmNupdateTextCallback 36, 37, 39
XmNuseImageCache 128
XmNuserData 12
XmNuseScaling 63
XmNuseTextField 37
XmNvalidateCallback 42, 43
XmNvalue 39
XmNvalueChangedCallback 63
XmNverify 36, 37
XmNverifyTextCallback 36, 37, 39
XmNverticalDelta 149
XmNverticalMargin 5, 36, 68, 75, 79, 135
XmNverticalNodeSpace 148, 150
XmNverticalScrollBar 49
XmNvisibleItemCount 37, 50
XmNwidth 12, 116
XmNwidthInc 121, 123
XmNx 12, 116, 123
XmNxlfldString 63
XmNy 12, 116, 123
XmPrimitive resources 159
XmSINGLE_SELECT 50
XmString 15, 45, 77
XmSTRING_DIRECTION_L_TO_R 50, 80, 130
XmSTRING_DIRECTION_R_TO_L 50, 79, 130
XmStringCopy 130
XmStringFree 15
XmStringTable 15
XmText 37
XmTextField 37
XmUNSPECIFIED_PIXMAP 110
XmVERTICAL 7, 28, 89, 91, 94, 134, 147, 149,
150
XtAddCallback 5, 9
XtAddCallback() 141
XtAppInitialize() 143
XtCallbackList 6, 141
XtCallbackProc 9
XtConstraint 8
XtCreateManagedWidget 2
XtCreateWidget 2, 3
XtFree 10, 39
XtGeometryYes 14
XtGetMultiClickTime() 82
XtGetValues 5, 6, 13, 38
XtMakeGeometryRequest 14
XtMalloc 39
XtManageChild 3
XtManageChildren 3
XtNameToWidget 13
XtRemoveCallback 5, 9
XtSetValues 3, 5, 6, 8, 13, 15, 38
XtSetValues() 139
XtVaSetValues 3

Z

Zooming, pixmap editor widget 105