

## **MIA 91-16: X Libraries - Reference Manual**



# Table of Contents

<a href="#">Copyright Notice</a>	1
<a href="#">List of programs</a>	3
<a href="#">Introduction</a>	5
<a href="#">What is Xt?</a>	6
<a href="#">The BETA interface to Xt et. al.</a>	7
<a href="#">Demo programs</a>	9
<a href="#">Environment Variables</a>	10
<a href="#">XtEnv</a>	11
<a href="#">Using the XtEnv fragment</a>	12
<a href="#">Basic XtEnv widget patterns</a>	13
<a href="#">Core</a>	14
<a href="#">Composite</a>	18
<a href="#">Constraint</a>	19
<a href="#">Shell</a>	20
<a href="#">WMShell</a>	21
<a href="#">ToplevelShell</a>	22
<a href="#">TransientShell</a>	23
<a href="#">OverrideShell</a>	24
<a href="#">VendorShell</a>	25
<a href="#">Other XtEnv Objects and Patterns</a>	26
<a href="#">Toplevel</a>	27
<a href="#">Display</a>	28
<a href="#">Timer</a>	29
<a href="#">WorkProc</a>	30
<a href="#">TranslationTable</a>	31
<a href="#">AcceleratorTable</a>	32



# Table of Contents

<a href="#"><u>RegisteredAction</u></a> .....	33
<a href="#"><u>StringArray</u></a> .....	34
<a href="#"><u>FallbackResources</u></a> .....	35
<a href="#"><u>Options</u></a> .....	36
<a href="#"><u>ErrorHandler</u></a> .....	37
<a href="#"><u>WarningHandler</u></a> .....	38
<a href="#"><u>Examples using Xt widgets</u></a> .....	39
<a href="#"><u>generic.bet</u></a> .....	39
<a href="#"><u>draw.bet</u></a> .....	39
<a href="#"><u>drawWithRefresh.bet</u></a> .....	40
<a href="#"><u>AwEnv</u></a> .....	42
<a href="#"><u>What is the Athena Widget set?</u></a> .....	42
<a href="#"><u>Using the AwEnv fragment</u></a> .....	42
<a href="#"><u>Overview of AwEnv Patterns</u></a> .....	43
<a href="#"><u>Simple Athena Patterns</u></a> .....	45
<a href="#"><u>Simple</u></a> .....	46
<a href="#"><u>Label</u></a> .....	47
<a href="#"><u>Command</u></a> .....	48
<a href="#"><u>ListWidget</u></a> .....	49
<a href="#"><u>StripChart</u></a> .....	50
<a href="#"><u>Toggle</u></a> .....	51
<a href="#"><u>Grip</u></a> .....	52
<a href="#"><u>Scrollbar</u></a> .....	53
<a href="#"><u>Examples using simple Athena Widgets</u></a> .....	54
<a href="#"><u>Using Label</u></a> .....	55
<a href="#"><u>hello.bet</u></a> .....	55
<a href="#"><u>Using ListWidget</u></a> .....	56
<a href="#"><u>list.bet</u></a> .....	56
<a href="#"><u>Using Command</u></a> .....	57
<a href="#"><u>button.bet</u></a> .....	57



# Table of Contents

<a href="#"><u>Using StripChart</u></a>	<b>58</b>
<a href="#"><u>stripchart.bet</u></a>	58
<a href="#"><u>Using Toggle</u></a>	<b>59</b>
<a href="#"><u>toggle.bet</u></a>	59
<a href="#"><u>baud.bet</u></a>	59
<a href="#"><u>Using ScrollBar</u></a>	<b>61</b>
<a href="#"><u>scroll.bet</u></a>	61
<a href="#"><u>Composite Athena Widgets</u></a>	<b>63</b>
<a href="#"><u>Box</u></a>	<b>64</b>
<a href="#"><u>Form</u></a>	<b>65</b>
<a href="#"><u>Extra attributes for children of Form</u></a>	65
<a href="#"><u>Form's Layout semantics</u></a>	65
<a href="#"><u>Dialog</u></a>	<b>66</b>
<a href="#"><u>Extra attributes for children of Dialog</u></a>	66
<a href="#"><u>Paned</u></a>	<b>67</b>
<a href="#"><u>Extra attributes for children of Paned</u></a>	67
<a href="#"><u>ViewPort</u></a>	<b>68</b>
<a href="#"><u>Layout Semantics</u></a>	68
<a href="#"><u>Using composite Athena Widgets</u></a>	<b>69</b>
<a href="#"><u>Using Form</u></a>	<b>70</b>
<a href="#"><u>form.bet</u></a>	70
<a href="#"><u>Cursors, Fonts and PixelMaps</u></a>	<b>72</b>
<a href="#"><u>fancyhello.bet</u></a>	72
<a href="#"><u>Menus</u></a>	<b>73</b>
<a href="#"><u>SimpleMenu</u></a>	<b>75</b>
<a href="#"><u>MenuButton</u></a>	<b>76</b>
<a href="#"><u>Sme</u></a>	<b>77</b>
<a href="#"><u>SmeBSB</u></a>	<b>78</b>
<a href="#"><u>SmeLine</u></a>	<b>79</b>
<a href="#"><u>SmeCascade</u></a>	<b>80</b>
<a href="#"><u>Examples of using Menus</u></a>	<b>81</b>



# Table of Contents

<a href="#"><u>A SimpleMenu example</u></a>	82
<a href="#"><u>menu.bet</u></a>	82
<a href="#"><u>Using cascading menus</u></a>	84
<a href="#"><u>cascademenu.bet</u></a>	84
<a href="#"><u>AsciiText - the Text Editor of Athena</u></a>	86
<a href="#"><u>AsciiText Widget for Users</u></a>	87
<a href="#"><u>Default Key Bindings</u></a>	88
<a href="#"><u>Example using AsciiText</u></a>	91
<a href="#"><u>text.bet</u></a>	91
<a href="#"><u>Prompts for Athena Widgets</u></a>	93
<a href="#"><u>Example using Athena Prompts</u></a>	94
<a href="#"><u>getstring.bet</u></a>	94
<a href="#"><u>MotifEnv</u></a>	97
<a href="#"><u>What is OSF/Motif?</u></a>	97
<a href="#"><u>Using the MotifEnv Fragment</u></a>	98
<a href="#"><u>Basic MotifEnv Patterns</u></a>	99
<a href="#"><u>Primitive Motif Widgets</u></a>	100
<a href="#"><u>Examples using Primitive Motif Widgets</u></a>	102
<a href="#"><u>hello.bet</u></a>	102
<a href="#"><u>list.bet</u></a>	102
<a href="#"><u>multi_font.bet</u></a>	104
<a href="#"><u>Motif Text Editor Widgets</u></a>	106
<a href="#"><u>Examples using Motif Text Editor Widgets</u></a>	107
<a href="#"><u>scrolledtext.bet</u></a>	107
<a href="#"><u>getpasswd.bet</u></a>	108
<a href="#"><u>Motif Gadgets</u></a>	110
<a href="#"><u>Examples using Motif Gadgets</u></a>	111
<a href="#"><u>Motif Manager Widgets</u></a>	112
<a href="#"><u>Examples using Motif Manager Widgets</u></a>	115
<a href="#"><u>rowcol.bet</u></a>	115
<a href="#"><u>bulletin.bet</u></a>	115
<a href="#"><u>form.bet</u></a>	116
<a href="#"><u>radiobox.bet</u></a>	117



# Table of Contents

<a href="#">mainwindow.bet</a>	118
<a href="#">scale.bet</a>	119
<b><a href="#">Motif Menus</a></b>	<b>121</b>
<b><a href="#">Examples using Motif Menus</a></b>	<b>123</b>
<a href="#">optionmenu.bet</a>	123
<a href="#">popupmenu.bet</a>	124
<a href="#">pulldownmenu.bet</a>	125
<b><a href="#">Motif Dialog Patterns</a></b>	<b>127</b>
<b><a href="#">Examples using Motif Dialog Patterns</a></b>	<b>130</b>
<a href="#">mainwindow.bet, continued</a>	130
<a href="#">filedialog.bet</a>	131
<b><a href="#">Other MotifEnv Patterns</a></b>	<b>134</b>
<b><a href="#">XSystemEnv</a></b>	<b>135</b>
<b><a href="#">Examples of using XSystemEnv</a></b>	<b>136</b>
<a href="#">XsimpleSleepDemo.bet</a>	136
<b><a href="#">Miscellaneous</a></b>	<b>137</b>
<a href="#">Xterm Interface</a>	137
<a href="#">EditRes Interface</a>	137
<b><a href="#">Bibliography</a></b>	<b>138</b>
<b><a href="#">Xtenv Interface</a></b>	<b>139</b>
<b><a href="#">Events Interface</a></b>	<b>161</b>
<b><a href="#">Xsystemenv Interface</a></b>	<b>174</b>
<b><a href="#">Basics Interface</a></b>	<b>175</b>
<b><a href="#">Callbackstruct Interface</a></b>	<b>183</b>
<b><a href="#">Primitive Interface</a></b>	<b>188</b>
<b><a href="#">Arrowbutton Interface</a></b>	<b>192</b>
<b><a href="#">Separator Interface</a></b>	<b>194</b>
<b><a href="#">Scrollbar Interface</a></b>	<b>196</b>
<b><a href="#">Label Interface</a></b>	<b>200</b>
<b><a href="#">Cascadebutton Interface</a></b>	<b>204</b>



# Table of Contents

<a href="#"><u>Drawnbutton Interface</u></a>	206
<a href="#"><u>Pushbutton Interface</u></a>	208
<a href="#"><u>Togglebutton Interface</u></a>	211
<a href="#"><u>Lists Interface</u></a>	215
<a href="#"><u>Texts Interface</u></a>	224
<a href="#"><u>Manager Interface</u></a>	239
<a href="#"><u>Bulletinboard Interface</u></a>	243
<a href="#"><u>Form Interface</u></a>	248
<a href="#"><u>Drawingarea Interface</u></a>	257
<a href="#"><u>Frame Interface</u></a>	259
<a href="#"><u>Rowcolumn Interface</u></a>	261
<a href="#"><u>Scrolledwindow Interface</u></a>	272
<a href="#"><u>Mainwindow Interface</u></a>	275
<a href="#"><u>Scale Interface</u></a>	278
<a href="#"><u>Panedwindow Interface</u></a>	281
<a href="#"><u>Selectionbox Interface</u></a>	283
<a href="#"><u>Fileselectionbox Interface</u></a>	288
<a href="#"><u>Command Interface</u></a>	296
<a href="#"><u>Messagebox Interface</u></a>	299
<a href="#"><u>Menushell Interface</u></a>	304
<a href="#"><u>Dialogshell Interface</u></a>	306
<a href="#"><u>Vendorshell Interface</u></a>	307
<a href="#"><u>Gadget Interface</u></a>	309
<a href="#"><u>Labelgadget Interface</u></a>	312
<a href="#"><u>Cascadebuttongadget Interface</u></a>	316
<a href="#"><u>Pushbuttongadget Interface</u></a>	318



# Table of Contents

<a href="#"><u>Togglebuttongadget Interface</u></a> .....	321
<a href="#"><u>Arrowbuttongadget Interface</u></a> .....	325
<a href="#"><u>Separator gadget Interface</u></a> .....	327
<a href="#"><u>Motifenv Interface</u></a> .....	329
<a href="#"><u>Allmotif Interface</u></a> .....	330
<a href="#"><u>Awenv Interface</u></a> .....	331
<a href="#"><u>Prompts Interface</u></a> .....	360
<a href="#"><u>Heapview Interface</u></a> .....	362
<a href="#"><u>Textactions Interface</u></a> .....	363
<a href="#"><u>Editres Interface</u></a> .....	371
<a href="#"><u>Xterm Interface</u></a> .....	372



# Copyright Notice

**Mjølner Informatics Report  
MIA 91-16  
August 1999**

Copyright © 1991-99 [Mjølner Informatics](#).  
All rights reserved.  
No part of this document may be copied or distributed  
without the prior written permission of Mjølner Informatics



## Legal Notice

---

The X Window System is a trademark of X Consortium, Inc.  
UNIX is a trademark of X/Open Company, Ltd.  
OPEN LOOK is a trademark of Novell, Inc.  
OSF/Motif is a trademark of Open Software Foundation, Inc.

Parts of this report are based on X Toolkit Intrinsic-C language Interface, by Joel McCormack, Paul Asente, and Ralph Swich, and X Toolkit Athene Widgets-C Language Interface, by Ralph Swich and Terry Weissman, both of which are copyrighted © 1985 - 1989 the Massachusetts Institute of Technology, Cambridge, Massachusetts, and Digital Equipment Corporation, Maynard, Massachusetts.

We have used this material under the terms of its copyright, which grants free use, subject to the following conditions:

Permission to use, copy, modify and distribute this documentation (i.e. the original MIT and DEC material) for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. or Digital not be used in in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T and Digital makes no representations about the suitability of the software described herein for any purpose. It is provided ``as is" without express or implied warranty.

---

Legal Notice

[Mjølner Informatics](#)

X Libraries - Reference Manual

---



# List of programs

[generic.bet](#)

[draw.bet](#)

[drawWithRefresh.bet](#)

[hello.bet](#)

[list.bet](#)

[button.bet](#)

[stripchart.bet](#)

[toggle.bet](#)

[baud.bet](#)

[scroll.bet](#)

[form.bet](#)

[fancyhello.bet](#)

[menu.bet](#)

[cascademenu.bet](#)

[text.bet](#)

[getstring.bet](#)

[hello.bet](#)



[list.bet](#)

[multi\\_font.bet](#)

[scrolledtext.bet](#)

[getpasswd.bet](#)

[rowcol.bet](#)

[bulletin.bet](#)

[form.bet](#)

[radiobox.bet](#)

[mainwindow.bet](#)

[scale.bet](#)

[optionmenu.bet](#)

[popupmenu.bet](#)

[pulldownmenu.bet](#)

[mainwindow.bet, continued](#)

[filedialog.bet](#)

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

X Libraries - Reference Manual

List of programs



# Introduction

This report describes the libraries available in the Mjølner System for programming X Window System applications.

The X Window System is a hardware-independent windowing system for workstations. It was developed by MIT and DEC and has been adopted as a standard platform for graphical applications. It is especially popular on UNIX systems.

An introduction to the X Window System seen from the users point of view is outside the scope of this report. If the reader is unfamiliar with using the X Window System, he is referred to one of the many available text-books on the subject (i.e. [\[Jones 89\]](#), [\[Mansfield 89\]](#), [\[Nye & O'Reilly 90a\]](#), [\[Nye & O'Reilly 90b\]](#), [\[Young 90\]](#)) or the on-line UNIX manual pages (man X).

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

.  
.

Introduction



# What is Xt?

For the C programmer the low-level interface used in programming X Window System applications is a library called Xlib. Xlib defines an extensive set of functions that provide complete access and control over the display, windows, and input devices. Although programmers can use Xlib to build applications, this rather low-level interface is tedious and hard to use correctly. It does little to make programming easy.

To simplify development of application programs, the development group behind the X Window System realized that the user-interface elements of a graphical application: scrollbars, commands buttons, menus, etc. should ideally be available ready-made for the programmer, and that the object-oriented style of programming is a desirable programming style when programming such applications. This resulted in a C-library called Xt on top of Xlib.

The purpose of Xt is to provide an object-oriented layer that supports user-interface abstractions called widgets. A widget is a reusable, configurable piece of C-code that operates independently of the application except through prearranged interactions.

Xt contains the basic functionality to support widgets, i.e. an architectural model for widgets that allow them to be written and used in an object-oriented fashion. Xt also contains a small core set of widgets.

One of the philosophies behind the X Window System is to be policy-free. It does not support any particular user interface style. The X Window System provides mechanisms to support many different interface styles rather than enforcing any one policy. Xt also attempts to be policy-free.

A widget set is a collection of widgets build on top of Xt that provide commonly used user-interface components tied together with a consistent appearance and user interface. Several different widget sets from various sources exists. The Athena widget set is one example. Others are Motif from Open Software Foundation (OSF) and OPEN WINDOWS from Sun and AT&T.

---



# The BETA interface to Xt et. al.

This report documents the BETA interfaces to the X Toolkit `XtEnv`, to the Athena widget set `AwEnv`, and to OSF/Motif `MotifEnv`. The interfaces consist of several fragments:

- `xlib`: a procedural interface to the low-level library Xlib. It contains an interface to nearly all the functions in Xlib. The interface is C-like: for most C-functions in Xlib there are corresponding BETA-patterns with the same names and enter/exit parameters corresponding to the C-parameters and return values.
- `events`: Contains BETA interface to X Events. Is included by `xlib`
- `xtlib`: a procedural interface to Xt. This fragment includes `xlib`.
- `awlib`: a procedural interface to Athena. Includes `xtlib`.
- `motiflib`: a procedural interface to OSF/Motif. Includes `xtlib`.
- `xtenv`: an interface to Xt where the abstractions of Xt is modelled in BETA in an object-oriented fashion. Includes `xtlib`.
- `awenv`: an object-oriented interface to Aw. Includes `awlib`.
- Various utility patterns for `awenv`.
- `motifenv`: an object-oriented interface to OSF/Motif. Includes `motiflib`. Actually `motifenv` is just the environment for Motif, the interfaces to the individual widgets and gadgets are placed in separate fragmentgroups, which must be included for use. The fragment `allmotif` includes them all.
- `xsystemenv`: When concurrency is used by means of the `SystemEnv` pattern described in [\[MIA 90-08\]](#), in conjunction with `XtEnv`, the `XSystemEnv` pattern located in the `xsystemenv` fragment should be used.

`xtenv`, `awenv`, `motifenv`, and `xsystemenv` are documented in the following chapters of this report.

`xlib`, `xtlib`, `awlib`, and `motiflib` are not documented as there is a one-one correspondence between these fragments and the corresponding C libraries. There should be no need for ordinary users to call any of the low-level routines in these fragments as there are corresponding documented higher-level operations for most of them.

`XtEnv`, `AwEnv`, and `MotifEnv` mostly contain user interface elements like menus, windows, dialog boxes, etc. The Mjølner System also contains a library called Bifrost with more graphical oriented elements like lines, splines, and circles based on an object-oriented stencil and paint model. This library is documented in [\[MIA 91-13\]](#). It Should also be noted, that a platform independent graphical user interface library - Lidskjalv - is now recommended instead of using `XtEnv`/`AwEnv`/`MotifEnv` directly. See [\[MIA 94-27\]](#) and [\[MIA 95-30\]](#).

---



## Introduction



# Demo programs

The demo programs presented in this report are also available in a machine readable form. They are located in the directory

`$BETALIB/demo/Xt`

that also contains other demos.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

·  
·

Introduction



# Environment Variables

The standard LD\_LIBRARY\_PATH environment variable is used in order to link X program correctly. This is given a default value in the Bourne Shell script \$BETALIB/configuration/env.sh, which are used by, e.g., the compiler. Check the LD\_LIBRARY\_PATH, and X specific variables like MOTIFHOME and MOTIFINC in this file. If they do not correspond to the organization of the libraries at your site, please have your system administrator correct the default values in this script. Notice, however, that normally you can control the values of these variables yourself, without editing the env.sh script: Values that are lists are just appended to by env.sh, and variables, that can have just one single value are only set by env.sh, if not set already.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

.  
.

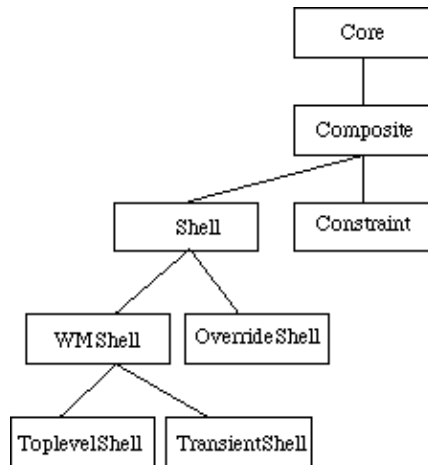
X Libraries - Reference Manual



# XtEnv

This chapter describes the BETA interface to the X toolkit (Xt). Xt contains the basic patterns common for many user-interface toolkits build on top of X, but does not contain any higher level user interface elements. It is typically used together with a widget set containing such user interface elements build on top of it. An example of such a widget set is the Athena Widget set. The BETA interface to the Athena Widget set is described in the next chapter.

The following figure illustrates the inheritance hierarchy among the XtEnv widgets:[\[1\]](#)



---

[1] Actually this figure is not complete: Above Core, two abstract superclasses called XtObject and RectObj exist. These classes should never be used directly by the user and will not be described in detail here. See the [Interface Descriptions](#) for details.

---



# Using the XtEnv fragment

The basic structure of the xtenv fragment is as follows:

```
ORIGIN 'xtlib';
-- LIB: attributes --
XtEnv: XtLib
  (# <<SLOT xtenvlib: attributes>>;
  ...(* xt widget patterns and other useful patterns *)
  do ...; INNER; ...
  #)
```

A typical BETA application using xtenv has the following outline:

```
ORIGIN '~beta/Xt/xtenv';
-- PROGRAM: descriptor --
XtEnv
  (#
  do ...
  #)
```

When xtenv starts executing it does some initializing. Afterwards it calls INNER allowing the application program to do its initialization. When the control returns to xtenv a global event handler is started. When an important user interaction event occurs, xtenv distributes the event to virtual patterns local for the user interface object in question of the application program.

---



## Basic XtEnv widget patterns



# Core

The core widget pattern is the most fundamental widget pattern, and serves as the superpattern for all other widget patterns. The core pattern defines characteristics common to all widgets, such as geometry, name, parent, sensitiveness, color, callback handling, translations and accelerators. It also defines various converters used to specify some of the other characteristics, e.g. it defines `textToFont`, used to specify font resources.

Core also defines an initialization method called `init` for the widget. This method must be called before anything can be done with the widget. It calls `INNER` such that specializations can add to the initialization behaviour.

`init` has an optional enter-part. The enter-part consists of two parameters: the father-widget and the name of the widget.

The father-widget is the widget, that shall contain this widget as a child. If the enter-part is not specified, the father-widget will be the enclosing widget of this widget according to BETA's scope-rules, see below. If this widget is not defined within the scope of the intended father-widget, the father-widget has to be specified in the enter-part.

The name of the widget is very important for the internal working of Xt as it is through this name, widgets are accessed. But the name is seldom important for the BETA-programmer. For some widgets the name is used as the default value for some of the widgets attributes. E.g. the name of a menu-item is used as the default value for the item-text presented to user when showing the menu. The name is also used if the programmer wants to change the default-values for some of attributes of some of the widgets used in the application. This can be done via a resourcefile (see [\[Nye & O'Reilly 90a\]](#), chapter 9). If the enter-part is not specified, the name will be the BETA-name of the descriptor for the widget.

Below some elaboration on the default father and default name of a widget is presented.

Consider the following program:

```
ORIGIN '~beta/Xt/awenv';
--- program: descriptor ---
AwEnv
(# faculty: label
  (# init:: (# do 2-> borderwidth #) #);
  University: @box
  (# Physics, Mathematics: @faculty;
    init:: (# do Physics.init; Mathematics.init #);
  #)
do University.init;
#)
```

The idea was that a window with two labels named `Physics` and `Mathematics` should appear. But executing it will give the error message

```
Xt Error: There must be only one non-shell widget which is son
of Toplevel. The widget causing the conflict is named faculty.
```

This is because the program uses the `init` pattern of the widgets without specifying the father and name of the widgets. To be precise, this is what happens: When the `init` pattern of a widget is invoked, it first checked to see if the father is `NONE`. This will be the case if no father is specified in the enter part of `init`.

If so, a search is started in the static environment of the widget pattern. If a specialization of a Core



widget is found, this widget is used as the father. This search is continued until a pattern with no enclosing pattern is found. In this case the widget named TopLevel (in xtenv) is used as the father. The widget TopLevel is an instance of the pattern TopLevelShell, which among its characteristics has the constraint that it wants to have exactly one non-shell child.

Now consider the example program: The first thing that happens is that the init attribute of University is invoked. Since no father is specified, a search for one is started from the University pattern. This search finds the pattern AwEnv(#...#), which is not a Core, and which has no enclosing pattern. Thus University will get the father widget TopLevel.

The final binding of University.init then invokes Physics.init. Physics is an instance of the pattern faculty, which is declared in the same scope as University. Thus the search for a father for Physics is identical to the search for the father of University, and Physics also gets TopLevel as its father. This is when the error occurs. The reason why the name reported in the error message is faculty is explained below.

Notice that it did not matter that the instantiation of the Physics object is done within University: the default father is searched for starting from the pattern declaration of the object.

In general there are three possible solutions to the problem in the example:

1. Supply the father and name when initializing the faculty widgets:

```
do ("Physics", University)->Physics.init;
  ("Mathematics", University)->Mathematics.init;
```

In this case, no search for a default father is needed for the faculty widgets.

2. Make (possibly empty) specializations of faculty inside University:

```
Physics: @faculty(##);
Mathematics: @faculty(##);
```

Now the search for a default father of Physics will start at the pattern faculty(##) inside University, so the University pattern will be the first found in this search, and hence the University widget will become the father of the Physics widget. Likewise for Mathematics.

3. Move the declaration of the faculty pattern inside the University pattern. This will give the same search path as in solution 2. (Conceptually, this might also be the best place to declare faculty in the first place.)

The above example was a simple one. In more complicated cases, the reason for an error of this kind can be trickier to spot. If your program uses the fragment system to move declarations of useful widgets into a library, this kind of error is likely to occur.

Remember that if an instance of an unspecialized widget is used, the widget pattern being declared in, say, the XtEnvLib attributes slot of xtenv, then the search for a default father is started at the XtEnv pattern, and therefore no father widget is found. In this case the widget will get TopLevel as father. Solutions 1 or 2 above will be appropriate in these cases.

The default name used for widgets sometimes also needs special attention: The following BETA program creates a window containing "Label"

```
ORIGIN '~beta/Xt/awenv'
--- program: descriptor ---
AwEnv
(# Hello: @Label;
do Hello.init;
#)
```

whereas the following program creates a window containing "Hello"

```
ORIGIN '~beta/Xt/awenv'
```



```

--- program: descriptor ---
AwEnv
(# Hello: @Label(##);
do Hello.init;
#)

```

Here is the reason why: The connection between the names used for widgets in BETA and the external names used in the external widgets interfaced to from BETA is that the pattern name of the BETA widget is used for the external widget name by default.

In the first example, the Hello widget is an instance of the pattern Label, and in the second example the widget is the only possible instance of the singular pattern Label(##), which is named Hello.

The appearance of the windows in this case comes from the fact that the Athena Label widget uses the external name of the widget as default label-string, if it is not specified otherwise.

A variant of this problem is the case where you specify a list of widgets using the same pattern:

```
hello1, hello2: @Label(##);
```

In this case the default name will always be the first name in the list, hello1. To avoid this behavior, use the scheme

```
hello1: @Label(##);
hello2: @Label(##);
```

or specify the name explicitly instead.

Every widget contains an instance of the pattern eventHandler, which allows the application program to do something when a particular event occurs. It is a low-level facility which is seldom needed as most widget patterns uses callbacks to notify the application-program when something interesting happens. eventhandler has local patterns keypress, keyrelease, buttonpress, ... , exposure, ... that can be used. If an instance of one of these patterns is enabled, it will be invoked, when the corresponding X-event occurs. The eventhandler has a local attribute called event, which is an instance of a virtual pattern, qualified with XAnyEvent (defined in [events.bet](#)). XAnyEvent is an abstract super-pattern for all X Events:

```

XAnyEvent: ExternalRecord (* Unspecified event *)
  (# type: @
    (* Type of THIS(XAnyEvent) *)
    long(# pos::< (# do 0->value #)#);
  serial: @
    (* number of last request processed by server *)
    long(# pos::< (# do 4->value #)#);
  send_event: @
    (* true if this came from a SendEvent request *)
    long(# pos::< (# do 8->value #)#);
  display: @
    (* Display the event was read from *)
    long(# pos::< (# do 12->value #)#);
  window: @
    (* window on which event was requested in event mask *)
    long(# pos::< (# do 16->value #)#);

  init:
    (* Used to allocate an XAnyEvent from within BETA *)
    (# do ... #);

  <<SLOT XAnyEventLib: attributes>>
enter ptr
exit ptr
#);

```



The different X events are then modelled as sub-patterns of XAnyEvent, e.g.

XButtonEvent: XAnyEvent

```
(# root: @
  (* root window that the event occurred on *)
  long(# pos::<(# do 20->value#)#);
subwindow: @
  (* child window *)
  long(# pos::<(# do 24->value#)#);
time: @
  (* milliseconds *)
  long(# pos::<(# do 28->value#)#);
x: @
  (* pointer x coordinate in event window *)
  long(# pos::<(# do 32->value#)#);
y: @
  (* pointer y coordinate in event window *)
  long(# pos::<(# do 36->value#)#);
x_root: @
  (* x coordinate relative to root *)
  long(# pos::<(# do 40->value#)#);
y_root: @
  (* y coordinate relative to root *)
  long(# pos::<(# do 44->value#)#);
state: @
  (* key or button mask *)
  long(# pos::<(# do 48->value#)#);
button: @
  (* detail *)
  long(# pos::<(# do 52->value#)#);
same_screen: @
  (* same screen flag *)
  long(# pos::<(# do 56->value#)#);

shiftModified:
  (# exit { true iff SHIFT was held down } #);
controlModified:
  (# exit { true iff CONTROL was held down } #);
...

<<SLOT XButtonEventLib: attributes>>
#);
```

Notice that some event types have some local utility patterns, in XButtonEvent, e.g., shiftModified, which is used to determine if the SHIFT modifier key was held down when the button-event occurred.

See the example-programs [draw.bet](#) and [drawWithRefresh.bet](#) later in this report for examples of using the eventhandler, and some of the attributes of X Events.



# Composite

Composite widgets are intended to be containers for other widgets. They have the ability to manage child widgets and they are responsible for handling the geometry for them.

The Composite widget pattern is an abstract superpattern. It is never instantiated directly in an application, only subpatterns are instantiated.



# Constraint

The Constraint widget pattern is a subpattern of the composite pattern, and does thus also manage the layout of children. Constraint widgets let the application provide layout information for each child. This information often takes the form of some constraints on the child's position and/or size. This is a more powerful way to arrange children because it allows you to provide different rules for how each child will be laid out.

The constraint widget is also an abstract superpattern.



# Shell

Widgets negotiate their size and positions with their parent widget (i.e. the widget that directly contain them). Widgets at the top of the hierarchy do not have any parent widget. Instead they must deal with the outside world. To provide for this, each top-level widget is encapsulated in a special widget, called a Shell widget.

The Shell widget pattern is a subpattern of the composite widget pattern, but can have only one child. Shell widgets are special purpose widgets that provide an interface between other widgets and the window manager. A shell widget negotiates the geometry of the widget with the window manager, and sets the properties required by the window manager.

The Shell widget pattern is also an abstract superpattern.



# WMShell

The WMShell is an abstract super pattern that contains attributes needed by the common window manager protocol, e.g. attributes used to specify icons, preferred size and aspect ratio etc.



# ToplevelShell

ToplevelShell widgets are used for normal top-level windows. xtenv creates the main toplevel widget `topLevel` of the applications. Some applications have multiple permanent top-level windows, and should use ToplevelShell to accomplish this.



# TransientShell

A TransientShell is similar to a toplevel shell except for the way it interacts with the window manager with respect to iconifying. If an applications toplevel window (transientFor) is iconified, the window manager normally iconifies all transient shells created by the application.



# OverrideShell

An OverrideShell instructs the window manager to completely ignore it. OverrideShell's completely bypass the window manager and therefore have no added window manager decorations. They are typically used for popup-menus.



# VendorShell

The VendorShell is an abstract super class with implementation dependent resources defined by the different widget sets using it.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

.  
.

XtEnv



## Other XtEnv Objects and Patterns



# Toplevel

TopLevel is the default toplevel widget. It is an instance of ToplevelShell and is automatically initialized by xtenv.



# Display

This pattern allows an application to have windows open on more than one workstation at the same time. After a display have been opened, the local item shell of the display can be used as a toplevel widget on that display. When children have been added to shell, the shell must be realized to show the window on screen.



# Timer

This pattern allow the application to be notified when a period of time have elapsed, while being able to do other things during that time.



# WorkProc

The Workproc-pattern is a means for doing background processing while the application is idle waiting for an event. After a Workproc has been started, the virtual method job is invoked by xtenv when no events are pending. It is invoked through a BETA component, such that a further binding of job may suspend itself in order not to let the user's response time suffer from time consuming background processing. Next time there are no events pending, job is resumed at the suspended place. More than one Workproc can be started at a time.



# TranslationTable

Can be used for the 'advanced' programmer to change the translations used by Xt. See [\[Nye & O'Reilly 90a\]](#), chapter 7 for more details.



# AcceleratorTable

Can be used for the 'advanced' programmer to change the accelerators used by Xt. See [\[Nye & O'Reilly 90a\]](#), chapter 7 for more details.



# RegisteredAction

Used to register so-called actions in Xt. These actions can be used in connection with translations and accelerators. See [\[Nye & O'Reilly 90a\]](#), chapter 7 for more details.



# StringArray

This is an array of externally allocated character strings. It is used for various purposes, e.g. to specify the contents of an Athena ListWidget.



# FallbackResources

Used to specify "default" resources for an application. These are set before the resource manager of Xt reads user-specified resources, and is thus used to "fall back" to, if no resources are specified by the user, but will still allow such user specified resources to be supplied



# Options

Used to specify additional resources to be readable from the command line of the application.



# ErrorHandler

An instance of ErrorHandler may be used to catch X errors as normal BETA runtime errors are caught. That is, a dump file is produced, specifying the dynamic call stack, when the error occurred, see [\[MIA 90-02\]](#). ErrorHandler contains two virtual exceptions xerror and xliberror that can be further bound as any other exception. That is, it is also possible to continue after an X error.



# WarningHandler

Like ErrorHandler, but for catching Xt warnings only. Contains the virtual xtwarning.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

.

XtEnv



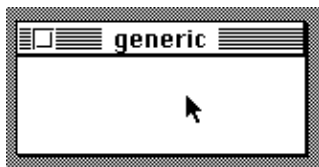
# Examples using Xt widgets

The following program illustrates the basic structure of an XtEnv application. It does nothing but popping up a window which is 50 pixels high and 150 pixels wide.

## **generic.bet**

```
ORIGIN '~beta/Xt/xtenv'
--- program : descriptor ---
XtEnv
(# widget: @Core
do widget.init;
    50 -> widget.height;
    150 -> widget.width
#)
```

In a specialization of XtEnv, a Core widget called widget is initialized. This will be a child of toplevel. Values are assigned to some of widget's attributes, whereafter the control returns to XtEnv. In the prefix-part toplevel is realized and XtEnv goes into the basic event loop. The realization of toplevel results in the popping up of the following window:



The decoration of the window - a titlebar with the title and a closebox - depends on the window manager running. The above dump of the window have been made on a Macintosh using the MacX X Window System. MacX contains its own window manager. Other window managers may make other decorations of the window like another type of titlebar or putting a resize-box on.

The name used in the titlebar is the name of the application.

Not very many interesting programs can be written using 'pure' XtEnv. The following program illustrates how the low-level Xlib calls can be used together with XtEnv, and it also illustrates how the low-level eventhandler facilities can be used. The demo program is thus for the advanced programmer. The demo program is an interactive drawing program. Every time the user presses the mouse button, a line is drawn.

## **draw.bet**

```
ORIGIN '~beta/Xt/xtenv'
--- PROGRAM: descriptor ---
XtEnv
(# widget: @Core
    (# px,py: @integer (* Used to hold the previous position *);
    gc: @integer
        (* The "Graphics Context" used in the drawings *);
    eventHandler::<
        (* Furtherbind to draw a line to the button press
        * position each time the mouse is pressed.
        *)
    (# bp: @ButtonPress
        (* Called each time the mouse is pressed *)
        (#
            do (display, window, gc, px, py,
                event.x, event.y) -> XDrawLine;
```

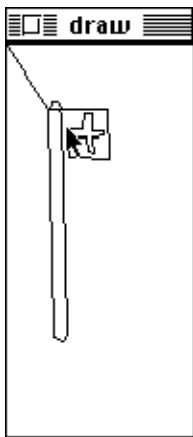


```

        (event.x,event.y) -> (px,py);
    #);
    init::< (# do bp.enable #);
    #);
    init::< (# do screenResource->XDefaultGCOfScreen->gc #);
    #);
do widget.init;
    200 -> widget.height;
    100 -> widget.width
#)

```

The bp object is called every time the user presses the mouse button. The Xlib functions XDrawLine and XDefaultGCOfScreen are used in the drawing of the line. The program may be used to make drawings like this:



The above program does not handle expose events. If another window is put on top of the above window and afterward moved, the contents of the 'draw' window is erased. The following demo-program is another version of the above program that handles expose-events. It is only necessary to worry about expose-events when drawing by means of low-level Xlib calls.

#### **drawWithRefresh.bet**

```

ORIGIN '~beta/Xt/xtenv';
INCLUDE '~beta/containers/list';
--- PROGRAM: descriptor ---
XtEnv
(# widget: @Core
    (# px,py: @integer; (* Previous position *)
        gc: @integer; (* Graphics Context *)

    Line:
        (* Definition of a simple "graphical object",
         * remembering its own state and with the ability
         * to redraw itself.
         *)
        (# x1,y1,x2,y2: @integer; (* End positions *)

        draw:
            (# do (display>window,gc,x1,y1,x2,y2)->XDrawLine #);
        init:
            (# enter ((x1,y1),(x2,y2))
                do THIS(line)[] -> lineList.append
                #);
        #);
    lineList: @List
        (* List of graphical objects to redraw in response to

```



```

    * expose events
    *)
    (# element::< line;
      redraw: scan(# do current.draw #)
    #);

eventHandler::<
  (* Furtherbind to create a new line object each time
   * the mouse is pressed, and to redraw the entire list
   * of lines in response to each expose event.
   *)
  (# bp: @ButtonPress
    (# l: ^Line
      do &Line[] -> l[];
        ((px,py),(event.x,event.y)) -> l.init;
        l.draw;
        (event.x,event.y) -> (px,py);
      #);
    ex: @Exposure(# do lineList.redraw #);
    init::< (# do bp.enable; ex.enable #);
  #);

init::<
  (#
    do 200 -> height;
    100 -> width;
    screenResource -> XDefaultGCOfScreen -> gc;
    lineList.init
  #);
#);
do widget.init;
#)

```

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

X Libraries - Reference Manual



# AwEnv

This chapter describes the BETA interface to the Athena widget set. The Athena widget set is build on top of Xt and it contains user interface elements like scrollbars, commands buttons, menus, etc.

## What is the Athena Widget set?

The Athena Widget Set (aw) is developed by MIT's Project Athena. It is included in the standard X Window System distribution and is thus universally available. It contains user-interface components like menus, scrollbars, command buttons and a variety of composite widgets.

The Athena widget set was not intended to be complete - it was built mainly for testing and demonstration of Xt. But it has matured into something quite useful, and it is at the time being the only non-commercial universally available widget set.

## Using the AwEnv fragment

The basic structure of the awenv fragment is as follows:

```
ORIGIN 'xtenv';
INCLUDE 'athena/awlib';
-- XtEnvLib: attributes --
... (* aw widget patterns and other useful patterns *)
AwEnv: XtEnv
  (# <<SLOT awenvlib: attributes>>;
   do INNER
   #)
```

A typical BETA application using awenv has the following outline:

```
ORIGIN '~beta/Xt/awenv'
-- program: descriptor --
AwEnv
  (#
   do ...
   #)
```

When xtenv (the prefix of awenv) starts executing, it does some initializing. Afterwards INNER is called allowing the application program to do its initialization. When the control returns to xtenv, a global event handler is started. When an important user interaction event occurs, xtenv distributes the event to virtual patterns local for the user interface object in question of the application program.

---



# Overview of AwEnv Patterns

AwEnv contains the following patterns:

- **Simple:** The base pattern for most of the simple widgets. Provides a rectangular area with a set-able mouse cursor and special border.
- **Label:** A rectangle that may contain one or more lines of text or a bitmap image.
- **Command:** A push button that, when selected, may cause a specific action to take place. This widget can display a multi-line string or a bitmap image.
- **ListWidget:** A list of text strings presented in row column format that may be individually selected. When an element is selected an action may take place.
- **Stripchart:** A real time data graph that will automatically update and scroll.
- **Toggle:** A push button (see Command) that contains state information. Toggles may also be used as radio buttons to implement a 'one of many' group of buttons.
- **Grip:** A rectangle that, when selected, will cause an action to take place.
- **Scrollbar:** A rectangular area containing a thumb that when slid along one dimension may cause a specific action to take place. The Scrollbar may be oriented horizontally or vertically.
- **SimpleMenu:** Provides support for menus.
- **Sme:** (simple menu entry) The base pattern of all menu entries. It may be used as a menu entry itself to provide a blank entry in a menu.
- **SmeBSB:** (simple menu entry with bitmap, string, bitmap). This menu entry provides a selectable entry containing a text string. Support is also provided that allows a bitmap to be placed in the left and right margins.
- **SmeLine:** This menu entry provides an unselectable entry containing a separator line.
- **SmeCascade:** This menu entry provides cascading menu entries.
- **MenuButton:** A push button (see Command) that pops up a SimpleMenu when a button is pressed.
- **Box:** This widget will pack its children as tightly as possible in non-overlapping rows.
- **Dialog:** An implementation of a commonly used interaction semantic to prompt for auxiliary input from the user, such as a filename.
- **Form:** A more sophisticated layout widget that allows the children to specify their positions relative to the other children, or to the edges of the Form.
- **Paned:** Allows children to be tiled vertically or horizontally. Controls are also provided to allow the user to dynamically resize the individual panes.



- `ViewPort`: Consists of a frame, one or two scrollbars, and an inner window. The inner window can contain all the data that is to be displayed. This inner window will be clipped by the frame with the scrollbars controlling which section of the inner window is currently visible.
  - `AsciiText`: The `AsciiText` widget provides a window that will allow an application to display and edit one or more lines of text. Options are provided to allow the user to add Scrollbars to its window, search for a specific string, and modify the text in the buffer.
- 

X Libraries - Reference Manual

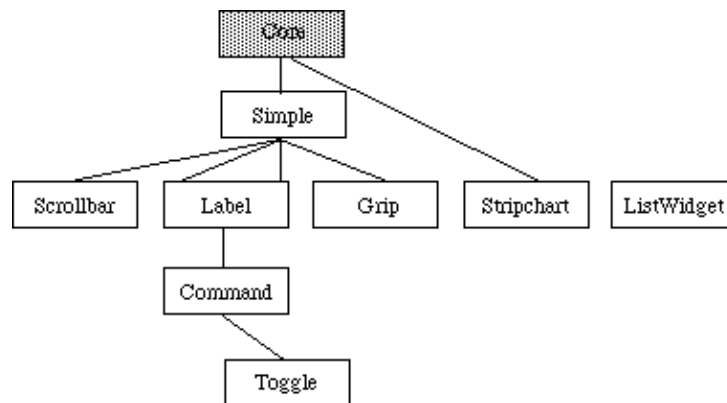
[Mjølner Informatics](#)

AwEnv



# Simple Athena Patterns

The following figure illustrates the inheritance hierarchy among the simple Athena widgets. Widgets from XtEnv are shaded gray:





# Simple

The Simple widget is not very useful by itself, as it has no semantics of its own. Its main purpose is to be used as a common superpattern for the other simple Athena widgets. Provides a rectangular area with a setable mouse cursor and special border.



# Label

A Label widget is a text string or bitmap displayed within a rectangular region of the screen. The label may contain multiple lines of characters. The Label widget will allow its string to be left, right, or center justified. Normally, this widget can be neither selected nor directly edited by the user. It is intended for use as an output device only.



# Command

The Command widget is an area, often rectangular, that contains a text label or bitmap image. Those selectable areas are often referred to as 'buttons'. When the pointer cursor is on a button, it becomes highlighted by drawing a rectangle around its perimeter. This highlighting indicates that the button is ready for selection. When pointer button 1 is pressed, the Command widget indicates that it has been selected by reversing its foreground and background colours. When the button is released, the Command widget's virtual pattern callback will be invoked. If the pointer is moved out of the widget before the button is released, the widget reverts to its normal foreground and background colours, and releasing the button has no effect. This behaviour allows the user to cancel an action.



# ListWidget

The ListWidget contains a list of strings formatted into rows and columns. When one of the strings is selected, it is highlighted, and the ListWidget's virtual pattern callback is invoked. Only one string may be selected at a time.



# StripChart

The StripChart widget is used to provide a real time graphical chart of a single value. This widget is used by the xload program to provide the load graph. It will read data from an application, and update the chart at the update interval specified.



# Toggle

The Toggle widget is an area, often rectangular, containing a text label or bitmap image. This widget maintains a Boolean state (e.g. True/False or On/Off) and changes state whenever it is selected. When the pointer is on the button, the button may become highlighted by drawing a rectangle around its perimeter. This highlighting indicates that the button is ready for selection. When pointer button 1 is pressed and released, the Toggle widget indicates that it has changed state by reversing its foreground and background colors, and its virtual method callback is invoked. If the pointer is moved out of the widget before the button is released, the widget reverts to its normal foreground and background colors, and releasing the button has no effect. This behavior allows the user to cancel an action.

Toggle buttons may also be part of a radio group. A radio group is a list of at least two Toggle buttons in which no more than one Toggle may be set at any time. A radio group is identified by any one of its members.



# Grip

The Grip widget provides a small rectangular region in which user input events (such as `ButtonPress` or `ButtonRelease`) may be handled. The most common use for the Grip widget is as an attachment point for visually repositioning an object, such as the pane border in a `Paned` widget.



# Scrollbar

The Scrollbar widget is a rectangular area containing a slide region and a thumb (also known as a slide bar). A Scrollbar can be used alone, as a value generator, or it can be used within a composite widget (for example, a Viewport). A Scrollbar can be oriented either vertically or horizontally.

When a Scrollbar is created, it is drawn with the thumb in a contrasting color. The thumb is normally used to scroll client data and to give visual feedback on the percentage of the client data that is visible.

Each pointer button invokes a specific action. That is, given either a vertical or horizontal orientation, the pointer button actions will scroll or return data as appropriate for that orientation. Pointer buttons 1 and 3 do not move the thumb automatically. Instead, they return the pixel position of the cursor on the scroll region. When pointer button 2 is clicked, the thumb moves to the current pointer position. When pointer button 2 is held down and the pointer is moved, the thumb follows the pointer.

The pointer cursor in the scroll region changes depending on the current action. When no pointer button is pressed, the cursor appears as a double-headed arrow that points in the direction that scrolling can occur. When pointer button 1 or 3 is pressed, the cursor appears as a single-headed arrow that points in the logical direction that the client will move the data. When pointer button 2 is pressed, the cursor appears as an arrow that points to the thumb.

While scrolling is in progress, the application receives notification through callback virtual patterns. For both discrete scrolling actions, the callback gives the pixel position of the pointer when the button was released. For continuous scrolling, the callback routine gives the current relative position of the thumb. When the thumb is moved using pointer button 2, the callback virtual is invoked continuously. When either button 1 or 3 is pressed, the callback virtual is invoked only when the button is released and the further binding of the callback virtual is responsible for moving the thumb.

---



## **Examples using simple Athena Widgets**



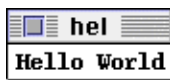
# Using Label

The following program illustrates how the 'standard' demo program which writes out the string 'Hello World' can be written using AwEnv. A Label widget is used to show the string to the user.

## **hello.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# hello: @Label;
do hello.init;
  'Hello World' -> hello.label
#)
```

When the program is run the following window is popped up:





# Using ListWidget

The following program illustrates how the ListWidget may be used. It is used to let the user select between four different strings.

## list.bet

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# alist: @ListWidget
  (# callback::<
    (#
      do 'Item number ' -> screen.putText;
      current.listIndex -> screen.putInt;
      ' have been selected: '' -> screen.putText;
      current.string -> screen.puttext;
      '' -> screen.putline;
    #);
    strings::<
      (# init::<
        (#
          do 'First Item ' -> addText;
          'Second Item ' -> addText;
          'Third Item ' -> addText;
          'Fourth Item ' -> addText;
        #)
      #);
  #)
do aList.init;
#)
```

When the program is run, the following window is popped up. When the user selects one of the list items, the callback-pattern in aList is invoked:





# Using Command

The following program illustrates the usage of Command. The program also uses a Box widget.

## **button.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# window: @Box
  (# commandButton: @Command
    (# callback::< (# do 'Thank you' -> please.label #)#);
    quit: @Command(# callback::< (# do stop #)#);
    please: @Label(##);

    init::<
      (#
        do commandButton.init;
        'Press here' -> commandButton.label;
        quit.init;
        please.init;
      #);
  #);
do window.init;
#)
```

The Box widget window contains three local widgets. When the program is run, the window shown to the left is popped up, and when the user clicks on the Press Here button, the window looks as shown to the right:





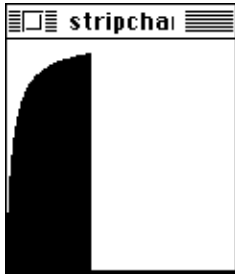
# Using StripChart

The following program illustrates the usage of StripChart.

## **stripchart.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# approxOne: @Stripchart
  (# n: @integer;
    getValue::< (# do n+1 -> n; (n,n+3) -> setQuotient #);
    init::< (# do 2 -> update #);
  #);
do approxOne.init;
#)
```

After the program has been running for about one minute, the window looks like this:





# Using Toggle

The following program illustrates the usage of one Toggle.

## **toggle.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# power: @Toggle
  (# callback::<
    (#
      do (if state then
        'on ' -> label
      else
        'off' -> label
      if)
    #);
  #);
do power.init;
'off' -> power.label
#)
```

When the program is run the user get a window which may switch between the following two states:



The following program illustrates how Toggles may be grouped into radio groups.

## **baud.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# baudRate: @Box
  (# baud: Toggle
    (# callback::<
      (#
        do radiodata -> screen.putInt;
        (if state
          // true then ' on' -> putLine
          // false then ' off' -> putLine
        if)
      #);
    #);
  b1,b2,b3,b4,b5: @baud;
  init::<
    (#
      do b1.init; '1200' -> b1.label; 1200 -> b1.radiodata;

      b2.init; '2400' -> b2.label; 2400 -> b2.radiodata;
      b1 -> b2.radioGroup; b2 -> b1.radioGroup;

      b3.init; '4800' -> b3.label; 4800 -> b3.radiodata;
      b2 -> b3.radioGroup;

      b4.init; '9600' -> b4.label; 9600 -> b4.radiodata;
      b3 -> b4.radioGroup;

      b5.init; '19200' -> b5.label; 19200 -> b5.radiodata;
```



```
        b4 -> b5.radioGroup;  
  
        false -> vertical;  
        4800 -> b1.setCurrent;  
    #);  
#);  
do baudRate.init;  
#)
```

When the program is run the user get a window where he may select one of the five buttons:





# Using ScrollBar

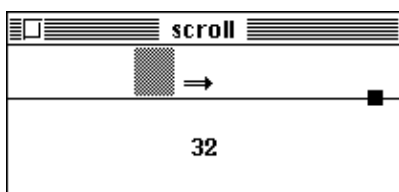
The following program illustrates the usage of a ScrollBar:

## **scroll.bet**

```
ORIGIN '~beta/Xt/awenv'
--- Program: descriptor ---
awenv
(# p: @paned
  (# s: @Scrollbar
    (# jumpProc::< (# do percent -> lab.putInt #);

    scrollProc::<
      (# current: @integer;
        do (topOfThumb*length-position*100) div length
          -> current;
        current -> screen.putInt; screen.newLine;
        (if true
          // (current<0) then 0 -> current
          // (current>(100-shown)) then
            100-shown -> current
          if);
        current -> topOfThumb -> lab.putInt;
      #);
  #);
lab: @Label
  (# putInt:
    (# t: @text; i: @integer
      enter i
      do i -> t.putInt; t[] -> label
    #)
  #);
init::<
  (#
    do s.init; false -> s.vertical; 100 -> s.length;
    10 -> s.shown; 40 -> s.thickness;
    lab.init; false -> lab.resize; 0 -> lab.putInt;
  #)
#);
do p.init;
#)
```

When the program is run the user get the following window where he can select an integer value by means of the scrollbar:



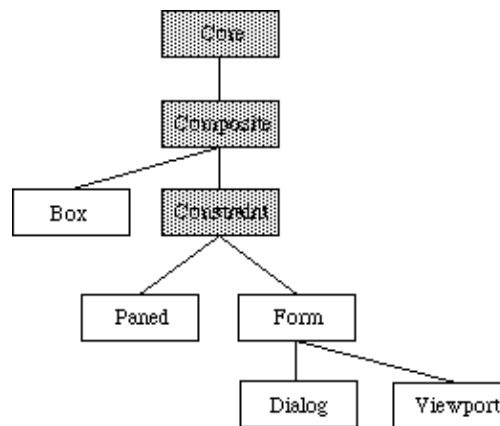


AwEnv



# Composite Athena Widgets

The following figure illustrates the inheritance hierarchy among the composite Athena widgets. Widgets from XtEnv are shaded gray:





# Box

The Box widget provides geometry management of arbitrary widgets in a box of a specified dimension. The children are rearranged when resizing events occur either on the Box or its children, or when children are managed or unmanaged. The Box widget always attempts to pack its children as tightly as possible within the geometry allowed by its parent.

Box widgets are commonly used to manage a related set of buttons and are often called ButtonBox widgets, but the children are not limited to buttons. The Box's children are arranged on a background that has its own specified dimensions and colour.



# Form

The Form widget can contain an arbitrary number of children or subwidgets. The Form provides geometry management for its children, which allows individual control of the position of each child. Any combination of children can be added to a Form. The initial positions of the children may be computed relative to the positions of other children. When the Form is resized, it computes new positions and sizes for its children. This computation is based upon information provided for each child.

The default width of the Form is the minimum width needed to enclose the children after computing their initial layout, with a margin of defaultDistance at the right and bottom edges. If a width and height is assigned to the Form that is too small for the layout, the children will be clipped by the right and bottom edges of the Form.

## Extra attributes for children of Form

Each child of the Form widget has a set of operations available to control the layout. These attributes allow the Form widget's children to specify individual layout requirements. Available attributes include the patterns fromHoriz, fromVert, horizDistance, vertDistance, resizable, bottom, left, right, top, see the [Interface Descriptions](#) for details.

The above mentioned patterns are actually attributes of the Core pattern. But they only function and should only be used for aggregation components of a Form.

## Form's Layout semantics

The Form widget uses two different sets of layout semantics. One is used when initially laying out the buttons. The other is used when the Form is resized.

The first layout method uses the fromVert and fromHoriz attributes to place the children of the Form. A single pass is made through the Form widget's children in the order that they were created. Each child is then placed in the Form widget below or to the right of the widget specified by the fromVert and fromHoriz attributes. The distance the new child is placed from its left or upper neighbour is determined by the horizDistance and vertDistance attributes. This implies some things about how the order of creation affects the possible placement of the children.

The second layout method is used when the Form is resized. It does not matter what causes this resize, and it is possible for a resize to happen before the widget becomes visible (due to constraints imposed by the parent of the Form). This layout method uses the bottom, top, left, and right attributes. These attributes are used to determine what will happen to each edge of the child when the Form is resized. If a value of Chain<something> is specified, the edge of the child will remain a fixed distance from the chain edge of the form. For example if ChainLeft is specified for the right attribute of a child then the right edge of that child will remain a fixed distance from the left edge of the widget. If a value of Rubber is specified, that edge will grow by the same percentage that the Form grew. For instance if the Form grows by 50% the left edge of the child (if specified as Rubber will be 50% farther from the left edge of the Form). One must be very careful when specifying these attributes, for when they are specified incorrectly children may overlap or completely occlude other children when the Form widget is resized.



# Dialog

The Dialog widget implements a commonly used interaction semantic to prompt for auxiliary input from a user. For example, you can use a Dialog widget when an application requires a small piece of information, such as a filename, from the user. A Dialog widget, which is simply a special case of the Form widget, provides a convenient way to create a preconfigured Form.

The typical Dialog widget contains three areas. The first line contains a description of the function of the Dialog widget, for example, the string 'Filename:'. The second line contains an area into which the user types input. The third line can contain buttons that let the user confirm or cancel the Dialog input. Any of these areas may be omitted by the application.

## Extra attributes for children of Dialog:

Children of Dialog have the same extra attributes as children of Form.



# Paned

The Paned widget manages children in a vertically or horizontally tiled fashion. The panes may be dynamically resized by the user by using the grips that appear near the right or bottom edge of the border between two panes.

The Paned widget may accept any widget pattern as a pane except Grip. Grip widgets have a special meaning for the Paned widget, and adding a Grip as its own pane will confuse the Paned widget.

The grips allow the panes to be resized by the user. The semantics of how these panes resize is somewhat complicated, and warrants further explanation here. When the mouse pointer is positioned on a grip and pressed, an arrow is displayed that indicates the pane that is to be to be resized. While keeping the mouse button down, the user can move the grip up and down (or left and right). This, in turn, changes the size of the pane. The size of the Paned widget will not change. Instead, it chooses another pane (or panes) to resize.

## Extra attributes for children of Paned

Each child of the Paned widget has a set of operations available to control the layout. These attributes allow the Paned widget's children to specify individual layout requirements.

Available attributes include the patterns `allowResize`, `maxSize`, `minSize`, `preferredPaneSize`, `resizeToPreferred`, `showGrip`, and `skipAdjust`. See the [Interface Descriptions](#) for details.

The above mentioned patterns are actually attributes of the Core pattern. But they only function and should only be used for aggregation components of a Paned widget.



# ViewPort

The Viewport widget consists of a frame window, one or two Scrollbars, and an inner window. The size of the frame window is determined by the viewing size of the data that is to be displayed and the dimensions to which the Viewport is created. The inner window is the full size of the data that is to be displayed and is clipped by the frame window. The Viewport widget controls the scrolling of the data directly. No application code are required for the scrolling.

When the geometry of the frame window is equal in size to the inner window, or when the data does not require scrolling, the ViewPort widget automatically removes any scrollbars. The forceBar option causes the Viewport widget to display all scrollbars permanently.

## Layout Semantics

The Viewport widget manages a single child widget. When the size of the child is larger than the size of the Viewport, the user can interactively move the child within the Viewport by repositioning the scrollbars.

The default size of the Viewport before it is realized is the width and/or height of the child. After it is realized, the Viewport will allow its child to grow vertically or horizontally if allowVert or allowHoriz are set, respectively. If the corresponding vertical or horizontal scrollbar is not enabled, the Viewport will propagate the geometry request to its own parent and the child will be allowed to change size only if the Viewport's parent allows it. Regardless of whether or not scrollbars are enabled in the corresponding direction, if the child requests a new size smaller than the Viewport size, the change will be allowed only if the parent of the Viewport allows the Viewport to shrink to the appropriate dimension.

Although the Viewport is a subpattern of the Form, none of the attributes for the children of Form mentioned on above must be supplied for any of the children of the Viewport. These attributes are managed internally, and are not meant for public consumption.

---



## Using composite Athena Widgets



# Using Form

The following program illustrates the usage of Form widgets and how layout are controlled for the children of the Form.

## **form.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# window: @Form
  (# top: @Command(##);
    middle1: @Command(##);
    middle2: @Command(##);
    middle3: @Command(##);
    bottom: @Command(##);

    init:<
      (#
        do top.init;
        middle1.init;
        middle2.init;
        middle3.init;
        bottom.init;

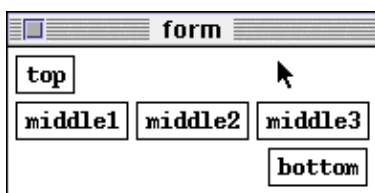
        top -> middle1.fromVert;

        top -> middle2.fromVert;
        middle1 -> middle2.fromHoriz;

        top -> middle3.fromVert;
        middle2 -> middle3.fromHoriz;

        middle1 -> bottom.fromVert;
        middle2 -> bottom.fromHoriz;
        10 -> bottom.horizDistance;
      #);
    #);
  do window.init;
  #)
```

Running the program results in the following window:





AwEnv



# Cursors, Fonts and PixelMaps

The following program illustrates how fonts, cursors, and pixelmaps can be used

## **fancyhello.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
awenv
(# main: @Box
  (# lab: @Label
    (# init::<
      (#
        do '*Helvetica-Bold-R-Normal--24-*' -> textToFont
          -> font;
        XCcoffeemug -> symbolToCursor -> cursor;
        'Hello World' -> label
      #)
    #);
  init::<
    (#
      do lab.init;
      '/usr/include/X11/bitmaps/escherknot'
        -> fileToPixmap -> backgroundPixmap
    #)
  #)
do main.init;
#)
```

Running the program gives the following window, notice the "Coffee mug" cursor:





# Menus

The Athena widget set provides support for single paned popup and pulldown menus. Menus are implemented as a Menu container (the SimpleMenu widget) and a collection of objects that will comprise the menu entries. The SimpleMenu widget is itself a direct subpattern of the OverrideShell widget pattern, therefore no other shell is necessary when creating a menu. The children of a SimpleMenu must be subpatterns of the Sme (Simple Menu Entry) object.

The Athena widget set provides four patterns of Sme objects that may be used to build menus.

- **Sme.** The base pattern of all menu entries. It may be used as a menu entry itself to provide a blank entry in a menu.
- **SmeBSB.** This menu entry provides a selectable entry containing a text string. Support is also provided that allows a bitmap to be placed in the left and right margins.
- **SmeLine.** This menu entry provides an unselectable entry containing a separator line.
- **SmeCascade.** This menu entry provides cascading menu entries.

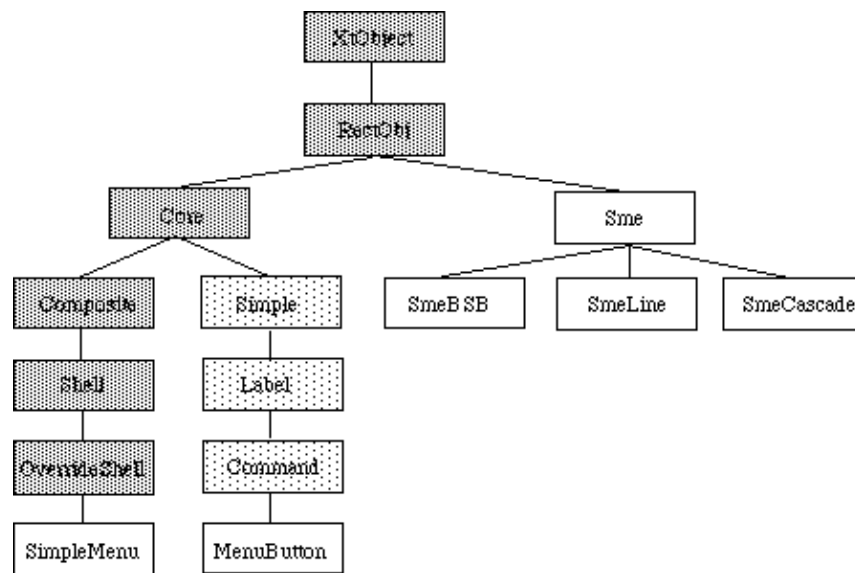
To allow easy creation of pulldown menus, a MenuButton widget is also provided.

The default configuration for the menus is click-move-release, and it is this interface that will be described. The menus will typically be activated by clicking a pointer button while the pointer is over a MenuButton, causing the menu to appear in a fixed location relative to that button; this is a pulldown menu. Menus may also be activated when a specific pointer and key sequence is used anywhere in the application; this is a popup menu. In this case the menu will position itself under the cursor. Typically menu will be placed so the pointer cursor is on the first menu entry, or the last entry selected by the user.

The menu will remain on the screen as long as the pointer button is held down. Moving the pointer will highlight different menu items. If the pointer leaves the menu, or moves over an entry that cannot be selected then no menu entry will be highlighted. When the desired menu entry has been highlighted, release the pointer button to remove the menu, and cause the callback associated with this entry to be invoked.

The following figure illustrates the inheritance hierarchy among the menu-related Athena widgets. Widgets from xtenv are shaded gray and other Athena widgets are dotted. Notice that Sme is a specialization of the abstract superclass RectObj of Core. This means that Sme is not a widget, but what is called a "gadget", i.e. a "windowless widget" that uses its parent window to draw in. For normal use this distinction is of no concern.







# SimpleMenu

The SimpleMenu widget is a container for the menu entries. It is a direct subpattern of OverrideShell. This is the only part of the menu that actually contains a window. The SimpleMenu serves as the glue to bind the individual menu entries together into a menu.



# MenuButton

The MenuButton widget is an area, often rectangular, that contains a text label or bitmap image. When the pointer cursor is on the button, the button becomes highlighted by drawing a rectangle around its perimeter. This highlighting indicates that the button is ready for selection. When a pointer button is pressed, the MenuButton widget will pop up a menu.



# Sme

The Sme (simple menu entry) object is the base pattern for all menu entries. While this object is mainly intended to be specialized, it may be used in a menu to add blank space between menu entries. The name Blank may be used as an alias for an Sme without specialization.



## SmeBSB

The SmeBSB (simple menu entry composed of a bitmap, a string and a bitmap) object is used to create a menu entry that contains a string, and optional bitmaps in its left and right margins. Since each menu entry is an independent object, the application is able to change the font, colour, height, and other attributes of the menu entries, on an entry by entry basis. The name Item may be used as an alias for an SmeBSB without specialization.



# SmeLine

The SmeLine object is used to add a horizontal line or menu separator to a menu. Since each menu entry is an independent object, the application is able to change the colour, height, and other attributes of the menu entries, on an entry by entry basis. This entry is not selectable, and will not highlight when the pointer cursor is over it. The name Line may be used as an alias for an SmeLine without specialization.



# SmeCascade

The SmeCascade object is used to add a cascading submenu to another menu. When the user points inside this item, the submenu is popped of to the right of the item. The user can then select among the items of the cascaded menu. The name Cascade may be used as an alias for an SmeCascade without specialization.

Since each menu entry is an independent object, the application is able to change the colour, height, and other attributes of the menu entries, on an entry by entry basis.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

.  
.

AwEnv



## Examples of using Menus



# A SimpleMenu example

The following program demonstrates how the SimpleMenu and the Sme widgets can be used. The menu is activated from a MenuButton widget.

## menu.bet

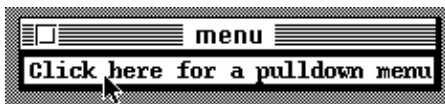
```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# command: @MenuButton
  (# theMenu: @SimpleMenu
    (# item1,item2,item3,item4: @SmeBSB
      (# callback::<
        (# do label -> putText; ' selected'->putLine #);
      #);

    quit: @Item(# callback::< (# do stop #)#);

    init::<
      (# bl: ^Blank;
        do quit.init;
        item1.init;
        'item1' -> item1.label;
        item2.init;
        'item2' -> item2.label;
        &blank[] -> bl[]; bl.init;
        20 -> bl.height;
        item3.init;
        'item3' -> item3.label;
        item4.init;
        'item4' -> item4.label;
      #);
    #);

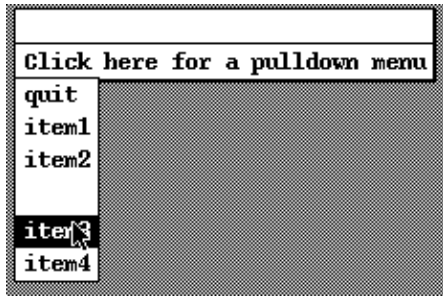
  init::<
    (#
      do 'Click here for a pulldown menu' -> label;
      theMenu.init;
      theMenu[] -> setMenu;
    #);
  #);
do command.init;
#)
```

When the program is run, the following window is popped up:



when the users clicks with the left mouse-button, the following menu is popped up:







# Using cascading menus

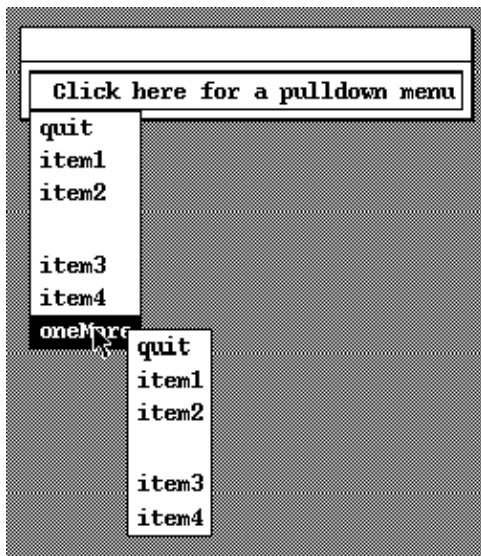
The following program is an extension of the previous, where a cascade menu item has been added.

## **cascademenu.bet**

```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# main: @Box
  (# theMenuButton: @MenuButton
    (# init::<
      (#
        do firstMenu.init;
        firstMenu[] -> setmenu;
        'Click here for a pulldown menu' -> label;
      #)
    #);
  Menu: SimpleMenu
    (# item1,item2,item3,item4: @SmeBSB
      (# callback::<
        (#
          do name -> putText; ' selected' -> putLine
        #)
      #);
    quit: @SmeBSB(# callback::< (# do stop #));
    init::<
      (# bl: ^blank;
        do quit.init;
        ('item1', THIS(Menu)) -> item1.init;
        ('item2', THIS(Menu)) -> item2.init;
        &blank[] -> bl[]; bl.init;
        20 -> bl.height;
        ('item3', THIS(Menu)) -> item3.init;
        ('item4', THIS(Menu)) -> item4.init;
        INNER;
      #);
    #);
  firstMenu: @Menu
    (# oneMore: @SmeCascade
      (# theMenu: @Menu;
        init::<
          (# do theMenu.init; theMenu -> subMenu #)
        #);
      init::< (# do oneMore.init #)
    #);
  init::<
    (# do theMenuButton.init; #);
  #);
do main.init;
#)
```

When the cascading menu entry is selected, the window looks like this:





---

X Libraries - Reference Manual

[Mjølner Informatics](#)

AwEnv



# AsciiText - the Text Editor of Athena

The AsciiText widget provides a window that will allow an application to display and edit one or more lines of text. Options are provided to allow the user to add Scrollbars to its window, search for a specific string, and modify the text in the buffer.

The word insert point is used in this chapter to refer to the text caret. This is the caret that is displayed between two characters in the text. The insert point marks the location where any new characters will be added to the text, i.e. the i'th position in an AsciiText is after the caret at position i. To avoid confusion the pointer cursor will always be referred to as the pointer.

The AsciiText widget supports three edit modes, controlling the types of modifications a user is allowed to make:

- Append-only
- Editable
- Read-only

Read-only mode does not allow the user or the programmer to modify the text in the widget. While the programmer may reset the entire string in read-only mode, it may not modify parts of the text with Replace. Append-only and editable modes allow the text at the insert point to be modified. The only difference is that text may only be added to or removed from the end of a buffer in append-only mode.



# AsciiText Widget for Users

The AsciiText widget provides many of the common keyboard editing commands. These commands allow users to move around and edit the buffer. If an illegal operation is attempted, (such as deleting characters in a read-only text widget), the terminal bell will be rung.

The default key bindings are patterned after those in the Emacs text editor:



# Default Key Bindings

Ctrl-a  
Beginning Of Line  
Meta-b  
Backward Word  
Ctrl-b  
Backward Character  
Meta-f  
Forward Word  
Ctrl-d  
Delete Next Character  
Meta-i  
Insert File  
Ctrl-e  
End Of Line  
Meta-k  
Kill To End Of Paragraph  
Ctrl-f  
Forward Character  
Meta-q  
Form Paragraph  
Ctrl-g  
Multiply Reset  
Meta-v  
Previous Page  
Ctrl-h  
Delete Previous Character  
Meta-y  
Insert Current Selection  
Ctrl-j  
Newline And Indent  
Meta-z  
Scroll One Line Down  
Ctrl-k  
Kill To End Of Line  
Meta-d  
Delete Next Word  
Ctrl-l  
Redraw Display  
Meta-D  
Kill Word  
Ctrl-m  
Newline  
Meta-h  
Delete Previous Word  
Ctrl-n  
Next Line  
Meta-H  
Backward Kill Word



Ctrl-o  
Newline And Backup  
Meta-<  
Beginning Of File  
Ctrl-p  
Previous Line  
Meta->  
End Of File  
Ctrl-r  
Search/Replace Backward  
Meta-]  
Forward Paragraph  
Ctrl-s  
Search/Replace Forward  
Meta-[  
Backward Paragraph  
Ctrl-t  
Transpose Characters

Ctrl-u  
Multiply by 4  
Meta-Delete  
Delete Previous Word  
Ctrl-v  
Next Page  
Meta-Shift Delete  
Kill Previous Word  
Ctrl-w  
Kill Selection  
Meta-Backspace  
Delete Previous Word  
Ctrl-y  
Unkill  
Meta-Shift Backspace  
Kill Previous Word  
Ctrl-z  
Scroll One Line Up

In addition, the pointer may be used to cut and paste text:

Button 1 Down  
Start Selection  
Button 1 Motion  
Adjust Selection  
Button 1 Up  
End Selection (cut)



Button 2 Down  
Insert Current Selection (paste)  
Button 3 Down  
Extend Current Selection  
Button 3 Motion  
Adjust Selection  
Button 3 Up  
End Selection (cut)



## Example using AsciiText

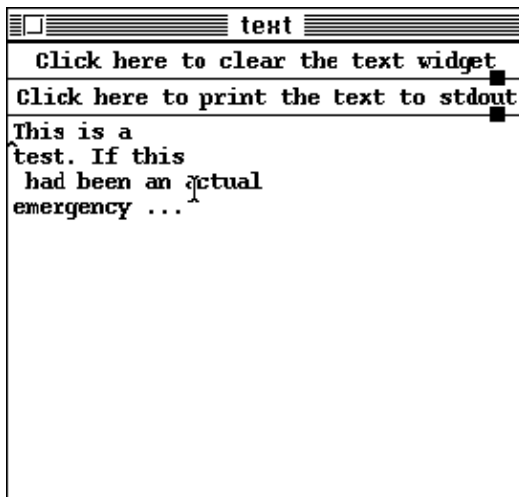
The following program illustrates the usage of AsciiText:

### **text.bet**

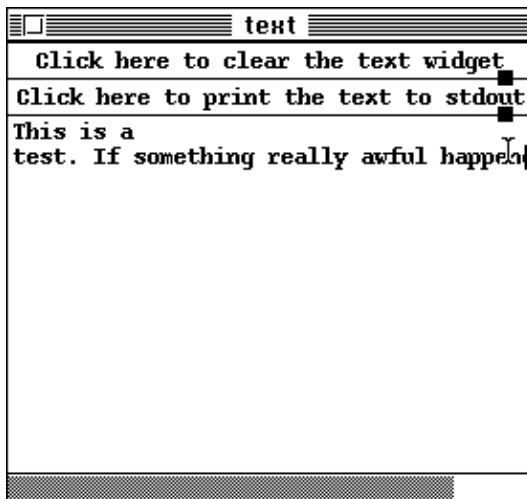
```
ORIGIN '~beta/Xt/awenv'
--- PROGRAM: descriptor ---
AwEnv
(# aPane: @Paned
  (# clear: @Command
    (# callback::< (# do ' ' -> txt.string #)#);
    print: @Command
    (# callback::<
      (#
        do 'The text is "' -> screen.putText;
        txt.string -> screen.putText;
        '" ' -> screen.putLine;
      #);
    #);
  txt: @AsciiText
  (#
    init::<
      (#
        do 200 -> preferredPaneSize;
        edit -> editType;
        ScrollWhenNeeded -> scrollVertical;
        ScrollWhenNeeded -> scrollHorizontal;
        false -> autoFill;
        'This is a \ntest. If this\n had been an
actual\nemergency ...' -> string
      #)#);
    init::<
      (#
        do clear.init;
        'Click here to clear the text widget'
        -> clear.label;
        print.init;
        'Click here to print the text to stdout'
        -> print.label;
        txt.init;
      #);
  #);
do aPane.init;
#)
```

When the program text is run, the following window pops up on the screen:





The user may edit the contents of the last window to i.e.:





# Prompts for Athena Widgets

The Dialog Widget provided by the Athena Widget set is not very useful in itself. It just contains an OK button, an optional Cancel button and an optional text entry field. The user of it must supply a Shell widget for father, and have to set various attributes to get even the simplest well known dialog boxes to work. Therefore three often used dialog boxes have been included in the BETA interface to the Athena Widgets. These are build up by using various other widgets from the Athena Widget set, including, of course, the Dialog widget.

These dialogs are:

- **Prompt:** A pattern which sets up a dialogbox containing a message and an OK button, which when clicked, dismisses the Prompt. Typing <RETURN> is the same as clicking on the OK button. The message and the text of the OK button may be set by the programmer. Also a small icon can be displayed in the Prompt if supplied by the programmer.
- **PromptForString:** A specialization of Prompt which sets up a dialogbox that asks the user to enter a text. It contains a message, a text entry field an OK and optionally a Cancel button. The pointer need not be inside the text entry field when typing. Typing <RETURN> is the same as clicking on OK, and typing <ESCAPE> is the same as clicking on Cancel. The message and the text of the OK and Cancel buttons may be set by the programmer.
- **PromptForBoolean:** A specialization of Prompt which sets up a dialogbox that asks the user to enter a boolean. It contains a message, two buttons for the answer an optionally a Cancel button. Typing <RETURN> is the same as clicking on the OK button, and typing <ESCAPE> is the same as clicking on Cancel. Also the first letter (in either case) of the labels of the OK and No buttons, respectively, may be typed to choose that button. The message and the text of the three buttons may be set by the programmer.



# Example using Athena Prompts

The following program illustrates the use of `PromptForString`, `PromptForBoolean` and `Prompt`.

## **getstring.bet**

ORIGIN '~beta/Xt/athena/prompts'

--PROGRAM: descriptor--

AwEnv

```
(# window: @Form
  (# h,v: @integer; (* Used to position the dialogs *)
    commandButton: @Command
      (# callback::<
        (#
          do (x+10,y+10) -> translatecoords -> (h,v);
          (h, v, 0, 'New label text?',
            'OK', 'Cancel', string.label)
            -> PromptForString
              (# ok::<(# do value[]->string.label #)#)
        #)
      #);
  quit: @Command
    (# callback::<
      (#
        do (x+10,y+10) -> translatecoords -> (h,v);
        (h,v,0,'Print string before stopping?',
          'Yes','No','Cancel')
          -> PromptForBoolean
            (# ok::<
              (# do string.label->putline; stop #);
              no::<
                (# do stop #);
              cancel::<
                (#
                  do (h,v,0,'We'll continue!','OK')->
                    Prompt
                      (# doubleBorder::<trueObject#)
                #);
              doubleBorder::< trueObject;
            #);
      #)
    #);
  string: @Label;
  init::<
    (#
      do commandButton.init;
      'Press here' -> commandButton.label;
      quit.init; 'Quit' -> quit.label;
      (* Specify 'quit' to be below 'commandButton' *)
      commandButton -> quit.fromVert;
      string.init; 'Please' -> string.label;
      (* Specify that 'string' should be below 'quit' *)
      quit -> string.fromVert;
      (* Make 'window' accept that 'string' resizes itself *)
      true -> string.resizable;
      (* Toplevel is the shell used by 'window' by default.
        * Allow it to resize itself if its children do.
        *)
      true -> toplevel.allowShellResize;
    #);
  #);
do window.init
#)
```



When this program is run, the following window appears:



When the button with the label 'Press Here' is pressed, a `PromptForString` is instantiated and popped up. The `PromptForString` is popped up a distance (10,10) pixels inside the 'Press Here' button. This is obtained by translating these coordinates into global coordinates, using `translatecoords`. On the screen, it looks as follows:

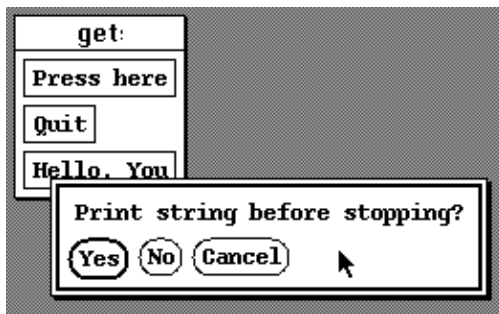


If the text 'Hello, You' is entered in the dialog, and the OK button is pressed (or `<RETURN>` is typed - this will invoke the OK button), the dialog disappears, and the Label named string will be changed. Then the main window will look as follows:



If then the Quit button is pressed, a `PromptForBoolean` is instantiated, asking if string should be printed before quitting:





If the Yes button is pressed, string.label is printed on the terminal, and the application stops. If the No button is pressed, the application just stops. But if the Cancel button is pressed, the application continues, i.e., quitting is cancelled. In this case a simple Prompt is used to inform the user of that:



---

X Libraries - Reference Manual

[Mjølner Informatics](#)

X Libraries - Reference Manual



# MotifEnv

This chapter describes the BETA interface to OSF/Motif. OSF/Motif is build on top of Xt and it contains user interface elements like scrollbars, buttons, standard dialogs, menus, etc. It also supports multi-font strings and gadgets.

## What is OSF/Motif?

The Motif widget set contains many user-interface components, including scroll bars, menus, buttons, dialogs, and a wide variety of composite widgets.

The most noticeable characteristics of the Motif Widget set is its three-dimensional (3-D) appearance. For instance buttons appear to be pushed in, when the user clicks on them. Most Motif widgets and gadgets draw a border around itself. The top and left sides of this border can be set to one color, and the right and bottom sides to another. By setting these sides to appropriate colors a 3-D shading effect can be obtained. These colors can be set directly by the programmer, but Motif will by default generate appropriate colors automatically based on the background color.

Motif has conventions about the use of its widgets and gadgets, that lead to a consistent look among all applications using Motif. Along with each Motif licence comes an OSF/Motif Style Guide with the documentation. This document contains recomandations for application design and layout.

Another notable feature of Motif is its resolution independence mechanism. All sizes and dimensions used by Motif can be specified in terms of pixels, multiples of 1/1000 of an inch, multiples of 1/100 of a point, multiples of 1/100 of a millimeter, or multiples of 1/100 of a font size. The default is to use pixels a the unit for sizes and dimensions.

OSF/Motif also supports some advanced features for internationalization such as language-independent so-called compound strings, keyboard traversal (moving around widgets and gadgets with keyboard keys rather than the mouse), and mnemonic key equivalents for invoking menu items.

OSF/Motif also supplies the mwm Motif Window Manager, which is ICCCM compliant; which decorates the windows with title bar, iconify button, help button, resize grips and which has even more features.



# Using the MotifEnv Fragment

The motifenv fragment group simply defines the MotifEnv prefix for applications using the BETA interface to Motif. For each widget/gadget wanted in the application, the fragment group in the sub-directory motif, defining the BETA interface to it, must be explicitly included.

```
ORIGIN 'xtenv';
INCLUDE 'motif/basics';
-- LIB: attributes --
MotifEnv: XtEnv
  (# <<SLOT MotifEnvLib: attributes>>
  do INNER
  #)
```

An application using motifenv thus have the following outline:

```
ORIGIN '~beta/Xt/MotifEnv'
INCLUDE '~beta/Xt/motif/rowcolumn'
INCLUDE '~beta/Xt/motif/pushbutton'
-- PROGRAM: descriptor --
MotifEnv
  (# ...
  do ...
  #)
```

In this case the program is using the RowBolumn- and PushButton widgets.

For ease of use, the fragment group allmotif includes the interface to all the widgets/gadgets, and may be used instead of motifenv fragment group at the price of a slightly bigger executable.

---



# Basic MotifEnv Patterns

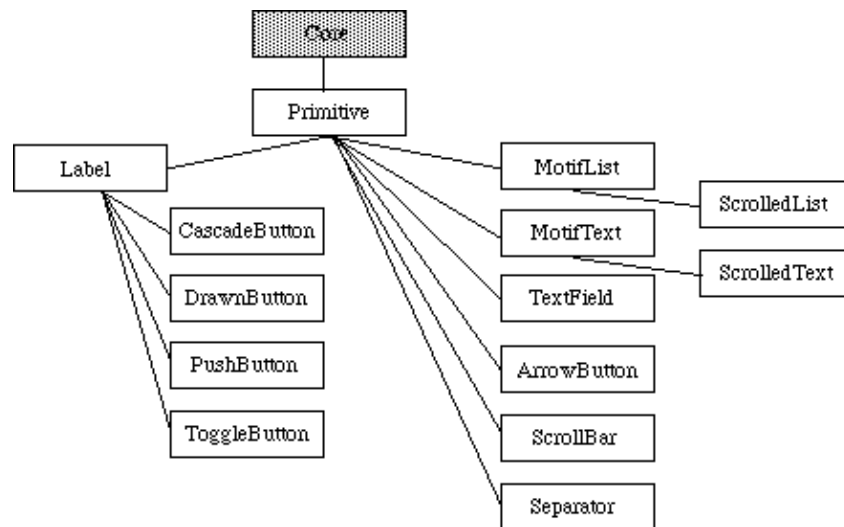
The following are some basic patterns defined in MotifEnv, which are used in most other MotifEnv patterns. They define various Motif specific extensions to XtEnv.

- **MotifCallback**: Unlike simple XtEnv callbacks, most Motif Callbacks returns data relevant for the callback along with the callback. The data returned is described using a specialization of the pattern called XmAnyCallbackStruct. The MotifCallback pattern defines a virtual pattern qualified by XmAnyCallbackStruct, and a static instance data MotifCallback is used as prefix for all the different Motif callbacks.
  - **MotifString**: This is the BETA interface to the so-called Compound Strings of OSF/Motif. A compound string is composed of text-segments, each having a Character Set (defining the font to use), a Direction (left-to-right or right-to-left), and the String consituting the text to show.
  - **MotifStringArray**: This is the BETA interface to an externally allocated array of MotifStrings, used in, e.g., MotifLists.
  - **MotifFontList**: Used, e.g., to define Character Sets for MotifStrings.
-



# Primitive Motif Widgets

The following figure shows the patterns constituting the primitive MotifEnv patterns. Patterns from XtEnv are shaded gray.



Notice that the Motif List and Text widgets are modelled by patterns called MotifList and MotifText, respectively, to avoid confusing them with corresponding patterns of the basic BETA libraries. The ScrolledList and ScrolledText patterns do not correspond directly to existing Motif widgets, but are supplied for convenience.

The following gives a brief overview of the Primitive widgets. The text editor widgets are described later:

- **Primitive**: used as a supporting superpattern for other widget patterns. It handles border drawing and highlighting, traversal activation and deactivation, and various callbacks needed by other widgets.
- **ArrowButton**: consists of a directional arrow surrounded by a border shadow. When it is selected, the shadow changes to give the appearance that the ArrowButton has been pressed in. When the ArrowButton is unselected, the shadow reverts to give the appearance that the ArrowButton is released, or out.
- **Separator**: a primitive widget that separates items in a display. Several different line drawing styles are provided, as well as horizontal or vertical orientation.
- **ScrollBar**: consists of two arrows placed at each end of a rectangle. The rectangle is called the scroll region. A smaller rectangle, called the slider, is placed within the scroll region. The data is scrolled by clicking either arrow, selecting on the scroll region, or dragging the slider. When an arrow is selected, the slider within the scroll region is moved in the direction of the arrow by an amount supplied by the application. If the mouse button is held down, the slider continues to move at a constant rate.
- **MotifList**: allows a user to select one or more items from a group of choices. Items are



selected from the list in a variety of ways, using both the pointer and the keyboard. MotifList operates on a StringArray that is defined by the application. Each string becomes an item in the MotifList, with the first string becoming the item in position 1, the second string becoming the item in position 2, and so on.

- **ScrolledList**: Utility pattern used to instantiate a MotifList within a ScrolledWindow.
- **Label**: can contain either a MotifString or a pixmap. When a Label is insensitive, its text is stippled, or the user-supplied insensitive pixmap is displayed.
- **CascadeButton**: links two MenuPanels or a MenuBar to a MenuPane. It is used in menu systems and must have a RowColumn parent with its rowColumnType resource set to XmMENU\_BAR, XmMENU\_POPUP or XmMENU\_PULLDOWN. It is the only widget that can have a Pulldown MenuPane attached to it as a submenu. The submenu is displayed when this widget is activated within a MenuBar, a PopupMenu, or a PulldownMenu. Its visuals can include a label or pixmap and a cascading indicator when it is in a Popup or Pulldown MenuPane; or, it can include only a label or a pixmap when it is in a MenuBar.
- **DrawnButton**: consists of an empty widget window surrounded by a shadow border. It provides the application developer with a graphics area that can have PushButton input semantics.
- **PushButton**: consists of a text label or pixmap surrounded by a border shadow. When a PushButton is selected, the shadow changes to give the appearance that it has been pressed in. When a PushButton is unselected, the shadow changes to give the appearance that it is out.
- **ToggleButton**: Usually this widget consists of an indicator (square or diamond) with either text or a pixmap on one side of it. However, it can also consist of just text or a pixmap without the indicator. The toggle graphics display a 1-of-many or N-of-many selection state. When a toggle indicator is displayed, a square indicator shows an N-of-many selection state and a diamond indicator shows a 1-of-many selection state. A ToggleButton implies a selected or unselected state. In the case of a label and an indicator, an empty indicator (square or diamond shaped) indicates that ToggleButton is unselected, and a filled indicator shows that it is selected. In the case of a pixmap toggle, different pixmaps are used to display the selected/unselected states.



# Examples using Primitive Motif Widgets

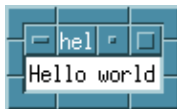
The following small program shows how to make the traditional "Hello world" program using a Motif Label widget:

## hello.bet

```
ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/label';

-- PROGRAM: descriptor --
MotifEnv
(# hello: @Label;
do hello.init;
  'Hello world' -> hello.labelString;
#)
```

When the program is run, the following window appears:



The border of the window, with grips for resizing the window, and the titlebar, with, e.g., iconify button, is added by the window manager, in this case mwm, the Motif Window Manager. The actual Label widget is the one showing the "Hello world" text. [\[2\]](#)

Notice that the Label widget has no 3-D appearance in itself. If that is wanted, a Frame widget could be used as father of the Label.

The following program shows how to construct a list of four items, that the user can select:

## list.bet

```
ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/lists';

-- PROGRAM: descriptor --
MotifEnv
(#
  lst: @MotifList
  (#
    browseSelectionCallback::<
      (* Called when an item is selected *)
      (# do (if data.item_position=itemCount then Stop if) #);
    defaultActionCallback::<
      (* Called when an item is double-clicked *)
      (# item: @MotifString;
      do 'The item ''' -> screen.putText;
        data.item -> item; item.getText -> screen.putText;
        '', at position ' -> screen.putText;
        data.item_position -> screen.putInt;
        ' has been double-clicked' -> screen.putLine;
      #);
    init::<
      (# strings: @MotifStringArray;
      do strings.init;
        'Item 1' -> strings.addText;
        'Item 2' -> strings.addText;
        'Item 3' -> strings.addText;
```



```

        'Item 4' -> strings.addText;
        'Quit  ' -> strings.addText;
        (strings, 1) -> additems;
        itemCount -> visibleItemCount;
    #)
#)
do lst.init;
#)

```

The list is constructed using the MotifString pattern, and the items are set up using the pattern MotifStringArray. MotifStringArray has almost the same attributes as the corresponding StringArray pattern of XtEnv. The difference is that a MotifStringArray is an array of so-called compound strings, that will be further explained in connection with the [multi\\_font](#) example below.

In the virtual init pattern of the ListWidget called lst, the MotifStringArray is initialized with five strings, and then added at position 1 in the list. The assignment of the itemCount resource (number of items) to the visibleItemCount resource, means that the window of the ListWidget will be big enough that all items are visible.

A ListWidget have four different "modes" of operation:

1. Single Selection mode - only one item can be selected at a time;
2. Multiple selection mode - multiple items may be selected, even non-adjacent items;
3. Extended select - ranges of adjacent items may be selected; and
4. Browse select - only one item may be selected at a time, but when moving the pointer with mouse button one pressed, the item under the mouse is highlighted, and when the mouse is released, the item is selected.

Browse select is the default, and when the above program is run, the window may look like this:



Item 2 is selected, and the mouse has been moved down to item 4 with the first mouse button pressed.

In all callbacks to Motif widgets and gadgets, an item called data is present. This is an instance of a virtual XmAnyCallbackStruct pattern. In all callbacks to MotifList widgets, the pattern data is an instance of is further bound to a pattern called XmListCallbackStruct. In the defaultActionCallback virtual of lst (which is called when an item is double-clicked) in the example, some of these attributes are used to print out information about the item selected. Likewise, in the browseSelectionCallback virtual (called when an item is selected in Browse selection mode), the data-object is used to determine if the last item (Quit) has been selected, and in that case the program is stopped.

The following small program shows how to use the Compound Strings of OSF/Motif, via the BETA abstraction called MotifString. It also shows how to define a fontlist for a Label widget. To



incorporate more than one font in a MotifString, you must create a font list, that specifies multiple character sets, for the widget that will be displaying the MotifString. This example is somewhat advanced, and may be skipped until multiple-font strings are needed.

### multi\_font.bet

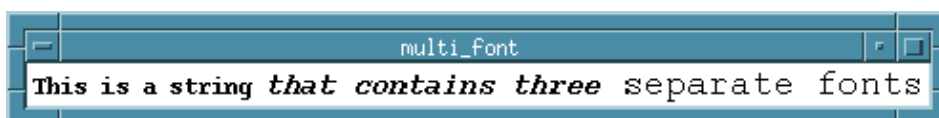
```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/label';
-- PROGRAM: descriptor --
MotifEnv
(
  multi: @Label
  (
    fontlst: @MotifFontList
    (
      (# init:<
        (
          do ('*-courier-*-r---12-*', 'charset1')
            -> addText;
          ('*-courier-bold-o---14-*', 'charset2')
            -> addText;
          ('*-courier-medium-r---18-*', 'charset3')
            -> addText;
        )
      );
    txt: @MotifString
    (
      (# init:<
        (
          do ('This is a string ', 'charset1',
            XmSTRING_DIRECTION_L_TO_R) -> setTextSegment;
          ('eerht sniatnoc taht', 'charset2',
            XmSTRING_DIRECTION_R_TO_L) -> appendSegment;
          (' separate fonts', 'charset3',
            XmSTRING_DIRECTION_L_TO_R) -> appendSegment;
        )
      );
    init:<
    (
      (# string: @labelString; (* The MotifStringResource *)
        do fontlst.init;
        fontlst -> fontList;
        txt.init;
        txt -> string.set;
      )
    );
  do multi.init;
)

```

Within the Label, two objects are used: A MotifFontList called fontlst, and a MotifString called txt. In fontlst, three standard X Windows font names are bound to three names, that may be used within the widget, the fontlist is used for. These three internal names denote so-called Character Sets. In the initialization of the Label called multi, this font list is assigned to the fontList resource of the Label. Then MotifStrings displayed by multi may use these character sets: In the initialization of the MotifString, three segments are added, using these three character sets. Note that the character sets are identified by the names in the font list. Notice also the specification of string direction in the three segments: The second segment is appended using right-to-left direction.

When the program is run, the following window appears:





Notice, that the second segment has been reversed in direction.

---

[2] Here the Label appears black and white. Normally the default color of Motif widgets is a bright blue color, but for documentation purposes, black and white looks better. The program was therefore invoked as `hello -bg white`, giving a white background, and consequently a black foreground. This is the case for most of the following screen snapshots too.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

MotifEnv



# Motif Text Editor Widgets

The three remaining widgets in the [figure of the primitive widget-hierarchy](#) are:

- **MotifText**: provides a single- or multiline text editor for customizing both user and programmatic interfaces.  
It can be used for single-line string entry, forms entry with verification procedures, and full-window editing.  
It provides an application with a consistent editing system for textual data. The screen's textual data adjusts to the application writer's needs.  
MotifText provides separate callbacks to verify movement of the insert cursor, modification of the text, and changes in input focus. Each of these callbacks provides the verification function with the widget instance, the event that caused the callback, and a data structure specific to the verification type. From this information the function can verify if the application considers this to be a legitimate state change and can signal the MotifText whether to continue with the action.  
A MotifText allows the user to select regions of text. Selection is based on the Interclient Communication Conventions (ICCC) selection model. A MotifText supports both primary and secondary selection.
- **ScrolledText**: Utility pattern for instantiating a MotifText within a ScrolledWindow.
- **TextField**: Like MotifText, but contains just one line of text. Used primarily for text entry fields in dialogs.



# Examples using Motif Text Editor Widgets

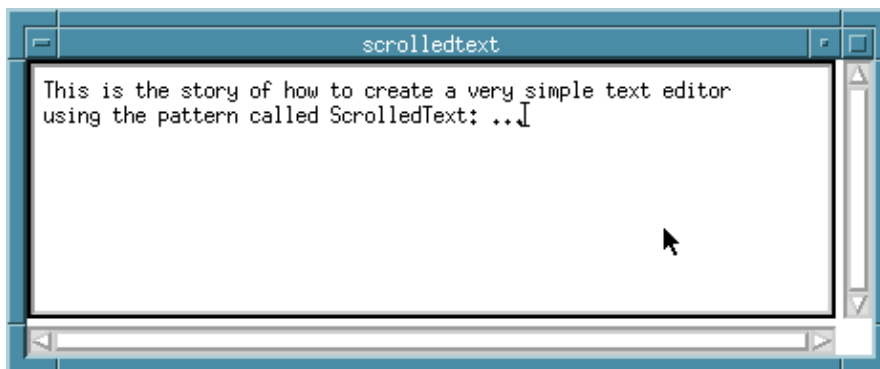
This is an example showing how to create a MotifText within a ScrolledWindow using the pattern ScrolledText.

## **scrolledtext.bet**

```
ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/texts';
-- PROGRAM: descriptor --
MotifEnv
(# txt: @ScrolledText
  (# init::<
    (#
      do 10 -> rows;
      80 -> columns;
      false -> resizeWidth;
      false -> resizeHeight;
    #)
  #);
do txt.init;
#)
```

This program creates a MotifText within a composite ScrolledWindow widget, which provides scroll bars for its child. The ScrolledText widget is conceptually a specialization of MotifText, that has some ScrolledWindow attributes too.

When the above program is run, a text editor with 10 lines, each of 80 characters appears:



The MotifText widget and the corresponding TextField widget have several callbacks installed: Virtual callbacks are called just before, and right after text is inserted into the buffer, when the widget gains and loses keyboard focus, when the widget gains or loses ownership of the X Window selection, when the insert cursor is moved within the buffer, etc.

The following example shows how to use the modifyVerifyCallback, which is called just before text is inserted into the buffer. The example shows how to implement the often used "enter password" field, where the characters typed is not visible on screen. To do this, within the callback, the character to be inserted into the buffer is changed to a '\*'. This is done by manipulating the callback-data, which contains pointers to externally allocated structures containing, among other things, the character(s) to be inserted to the text buffer, after the callback.

The example is somewhat complex, and may be skipped at first reading.



**getpasswd.bet**

```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/texts';
INCLUDE '~beta/Xt/motif/rowcolumn';
INCLUDE '~beta/Xt/motif/label';

-- PROGRAM: descriptor--
MotifEnv
(# rowcol: @RowColumn
  (# prompt: @Label
    (# init::< (# do 'Enter password:' -> labelString #));
    getpasswd: @MotifText
    (# passwd: @text; (* Used to save the password typed *)
      init::<
        (# do XmSINGLE_LINE_EDIT -> editMode #);
      modifyVerifyCallback::<
        (* Called before text is deleted or inserted *)
        (# typedtext: @XmTextBlockRec;
          string: @CString;
          inx: @integer;
          txt: @text;
          do data.text -> typedtext;
          (if typedtext.ptr=0 then
            (* BackSpace/Delete was typed:
              * Delete char in copy of typed password
              *)
            data.currInsert -> inx;
            (if inx>0 then
              (inx, inx) -> passwd.delete
            if);
          else
            (if typedtext.length = 1 then
              (* Allow simple access to the external
                * C string in the XmTextBlockRec struct
                *)
              typedtext.ptr -> string;
              (* Save the character typed *)
              0-> string.inxget -> txt.put;
              (txt[], data.currInsert+1)
                -> passwd.insert;
              (* Replace character with '*' *)
              (0, '*') -> string.inxput;
            else
              (* Disable multi-character insertions
                * (e.g. pasting)
                *)
              false -> data.doit;
            if);
          if)
        #);
      activateCallback::<
        (# (* The <Return> key was pressed *)
          do 'You have entered '' -> screen.puttext;
          passwd[] -> screen.puttext;
          ''.' -> screen.putline;
          Stop;
          #);
        #);
    init::< (# do prompt.init; getpasswd.init; #);
  #);
do rowcol.init;
#)

```

To access the external string, an instance of a pattern called `XmTextBlockRec` (declared in `MotifLib`) is used.



When the program is run, and after six characters has been typed, the window looks like this:



When the <RETURN> key is typed, the activateCallback is called. Within this, the unencrypted password is printed on the screen.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

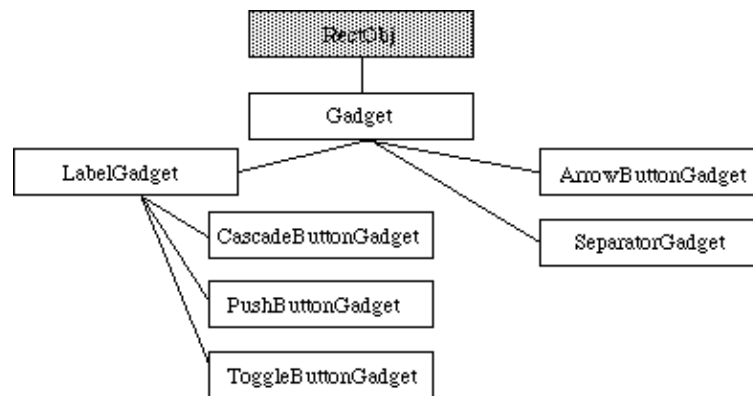
.

MotifEnv



# Motif Gadgets

The following figure shows the patterns constituting the MotifEnv gadget patterns. Patterns from XtEnv are shaded gray.



The following is a brief description of the Motif gadgets:

- **Gadget**: a pattern used as a supporting superpattern for other gadget patterns. It handles shadow-border drawing and highlighting, traversal activation and deactivation, and various callbacks needed by gadgets. The difference between a Primitive and a Gadget is, that a gadget is "window-less", i.e., it uses the window of its parent to draw into. The color and pixmap resources defined by Manager are directly used by gadgets. If one of these resources for is changed for a Manager widget, all of the gadget children within the Manager also change. Gadget is a specialization of RectObj; in contrast Primitive is a specialization of XtObject.
- **LabelGadget**: Like Label, but uses its parent window for drawing. Almost the same interface as Label. Requires less memory than Label.
- **CascadeButtonGadget**: Like CascadeButton, but uses its parent window for drawing. Almost the same interface as CascadeButton. Requires less memory than CascadeButton.
- **PushButtonGadget**: Like PushButton, but uses its parent window for drawing. Almost the same interface as PushButton. Requires less memory than PushButton.
- **ToggleButtonGadget**: Like ToggleButton, but uses its parent window for drawing. Almost the same interface as ToggleButton. Requires less memory than ToggleButton.
- **ArrowButtonGadget**: Like ArrowButton, but uses its parent window for drawing. Almost the same interface as ArrowButton. Requires less memory than ArrowButton.
- **SeparatorGadget**: Like Separator, but uses its parent window for drawing. Almost the same interface as Separator. Requires less memory than Separator.



# Examples using Motif Gadgets

With very few exceptions, a gadget can be used wherever, the corresponding widget would otherwise be used. The exceptions are: Gadgets do not support eventhandlers, translations or popup children. In several of the examples on the following pages, gadgets are used instead of widgets. Gadgets takes some of the load of the X Window server, since they do not create their own window, but instead draw in their parent window. The parents, however, must be prepared to support the gadgets in this sharing of the window. In general all Motif composites support gadgets.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

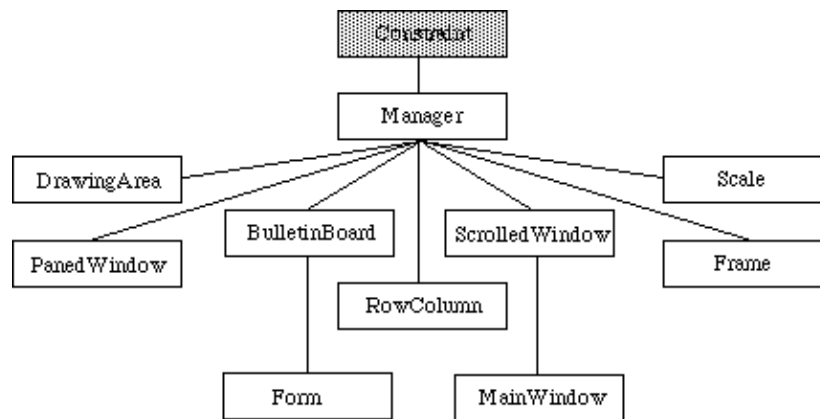
.

MotifEnv



# Motif Manager Widgets

The following figure shows the patterns constituting the MotifEnv manager patterns. Patterns from XtEnv are shaded gray.



The following is a brief description of most of the Motif Manager patterns. The Menu related patterns (special RowColumns, not shown in the figure) are described [later](#):

- **Manager**: a pattern used as a supporting superpattern for other composite patterns. It supports the visual resources, graphics contexts, and traversal resources necessary for the graphics and traversal mechanisms.
- **BulletinBoard**: a composite widget that provides simple geometry management for child widgets. It does not force positioning on its children, but can be set to reject geometry requests that result in overlapping children. BulletinBoard is the base widget for most dialog widgets and is also used as a general container widget. If its parent is a DialogShell, BulletinBoard passes title and input mode (based on dialog style) information to the parent, which is responsible for appropriate communication with the window manager.
- **Form**: a BulletinBoard with no input semantics of its own. Constraints are placed on children of the Form to define attachments for each of the child's four sides. These attachments can be to the Form, to another child widget or gadget, to a relative position within the Form, or to the initial position of the child. The attachments determine the layout behavior of the Form when resizing occurs.
- **DrawingArea**: an empty widget that is easily adaptable to a variety of purposes. It does no drawing and defines no behavior except for invoking callbacks. Callbacks notify the application when graphics need to be drawn (exposure events or widget resize) and when the widget receives input from the keyboard or mouse. Applications are responsible for defining appearance and behavior as needed in response to DrawingArea callbacks.
- **Frame**: a very simple manager used to enclose a single child in a border drawn by the Frame. It uses the Manager class resources for border drawing and performs geometry management so that its size always matches its child's size plus the margins defined for it. Frame is most often used to enclose other managers when the application developer desires the manager to have the same border appearance as the primitive widgets.



- **RowColumn:** a general purpose row/column manager capable of containing any widget type as a child. In general, it requires no special knowledge about how its children function and provides nothing beyond support for several different layout styles. However, it can be configured as a menu, in which case, it expects only certain children, and it configures to a particular layout. The menus supported are: MenuBar, Pulldown or Popup MenuPanels, and OptionMenu. The type of layout performed is controlled by how the application has set the various layout resources. Menus are described in detail later in this document. It can be configured to lay out its children in either rows or columns. In addition, the application can specify how the children are laid out, as follows:

1. the children are packed tightly together into either rows or columns,
2. each child is placed in an identically sized box (producing a symmetrical look), or
3. a specific layout (the current x and y positions of the children control their location).

In addition, the application has control over both the spacing that occurs between each row and column and the margin spacing present between the edges of the RowColumn widget and any children that are placed against it. By default the RowColumn widget has no 3-D visuals associated with it; if an application wishes to have a 3-D shadow placed around this widget, it can create the RowColumn as a child of a Frame widget.

- **Radiobox:** Utility pattern to instantiate a RowColumn widget of type XmWORK\_AREA. Typically, this is a composite widget that contains multiple ToggleButtonGadgets. The RadioBox arbitrates and ensures that at most one ToggleButtonGadget is on at any time. This pattern provides initial values for several RowColumn resources. It initializes packing to XmPACK\_COLUMN, radioBehavior to True, isHomogeneous to True, and entryClass to xmToggleButtonGadgetClass. In a RadioBox the ToggleButton or ToggleButtonGadget resource indicatorType defaults to XmONE\_OF\_MANY, and the ToggleButton or ToggleButtonGadget resource visibleWhenOff defaults to True.
- **ScrolledWindow:** combines one or two ScrollBar widgets and a viewing area to implement a visible window onto some other (usually larger) data display. The visible part of the window can be scrolled through the larger display by the use of ScrollBars.
- **MainWindow:** provides a standard layout for the primary window of an application. This layout includes a MenuBar, a CommandWindow, a work region, a MessageWindow, and ScrollBars. Any or all of these areas are optional. The work region and ScrollBars in the MainWindow behave identically to the work region and ScrollBars in the ScrolledWindow widget. The user can think of the MainWindow as an extended ScrolledWindow with an optional MenuBar and optional CommandWindow and MessageWindow. In a fully-loaded MainWindow, the MenuBar spans the top of the window horizontally. The CommandWindow spans the MainWindow horizontally just below the MenuBar, and the work region lies below the CommandWindow. The MessageWindow is below the work region. A MainWindow can also create three Separator widgets that provide a visual separation of MainWindow's four components.
- **Scale:** used by an application to indicate a value from within a range of values, and it allows the user to input or modify a value from the same range. A Scale has an elongated rectangular region similar to a ScrollBar. A slider inside this region indicates the current value along the Scale. The user can also modify the Scale's value by moving the slider within the rectangular region of the Scale. A Scale can also include a label set located outside the Scale region. These can indicate the relative value at various positions along the scale. A Scale can be either input/output or output only. An input/output Scale's value can be set by the application and also modified by the user with the slider. An output-only Scale is used strictly as an indicator of the current value of something and cannot be



modified interactively by the user.

- **PanedWindow**: a Composite that lays out children in a vertically tiled format. Children appear in top-to-bottom fashion, with the first child inserted appearing at the top of the PanedWindow and the last child inserted appearing at the bottom. The user can adjust the size of the panes. To facilitate this adjustment, a pane control sash is created for most children. The sash appears as a square box positioned on the bottom of the pane that it controls. The user can adjust the size of a pane by using the mouse or keyboard. The PanedWindow is also a Constraint, which means that it creates and manages a set of constraints for each child.



# Examples using Motif Manager Widgets

The following example shows one usage of the very useful RowColumn widget. Here six PushButtonGadgets are placed in three horizontal rows.

## rowcol.bet

```
ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/rowcolumn';
INCLUDE '~beta/Xt/motif/pushbuttonwidget';

-- PROGRAM: descriptor --
MotifEnv
(# rowCol: @RowColumn
  (# buttons: [6]^PushButtonGadget;
    init::<
      (# t: @text;
        do (for i: 6 repeat
          &PushButtonGadget[] -> buttons[i][];
          buttons[i].init;
          t.clear; 'Button ' -> t.putText;
          i -> t.putInt;
          t[] -> buttons[i].labelString;
        for);
        XmHORIZONTAL -> orientation;
        XmPACK_COLUMN -> packing;
        3 -> numColumns;
      #);
    #);
  do rowCol.init;
  #)
```

Notice that when the orientation is horizontal, the numColumns resource specifies the number of rows. When the program is run, the following window appears:



The following example shows how the BulletinBoard can be used: Six children are placed at absolute (x,y) positions.

## bulletin.bet

```
ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/bulletinboard';
INCLUDE '~beta/Xt/motif/texts';
INCLUDE '~beta/Xt/motif/pushbutton';

-- PROGRAM: descriptor --
MotifEnv
(# board: @BulletinBoard
  (# button: PushButton
    (# activateCallback::<
      (#
```

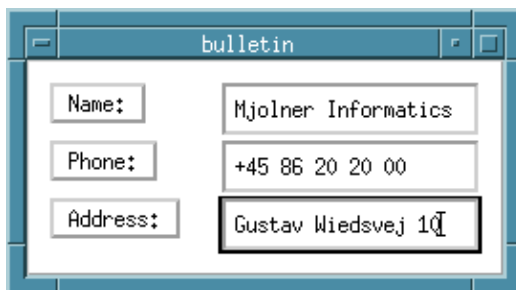


```

do (for i: buttons.range repeat
    (if buttons[i][]= this(button)[] then
        editors[i].value -> screen.putLine;
        if)
    for)
#);
#);
buttons: [3] ^button;
editors: [3] ^MotifText;
init::<
(#
do (for i: buttons.range repeat
    &button[] -> buttons[i][];
    buttons[i].init;
    10 -> buttons[i].x;
    10 + (30*(i-1)) -> buttons[i].y;
for);
(for i: editors.range repeat
    &MotifText[] -> editors[i][];
    editors[i].init;
    100 -> editors[i].x;
    10 + (30*(i-1)) -> editors[i].y;
for);
' Name: ' -> buttons[1].labelString;
' Phone: ' -> buttons[2].labelString;
' Address: ' -> buttons[3].labelString;
#);
#);
do board.init;
#)

```

When the program is run, the following window appears:



The three text entry fields have been filled out.

The next example shows one way to use the Form widget: Three PushButtons are aligned above each other using the Form constraint resources of the children. Notice that this could be done much easier using a RowColumn widget.

#### form.bet

```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/form';
INCLUDE '~beta/Xt/motif/pushbutton';

-- PROGRAM: descriptor --
MotifEnv
(# window: @Form
    (# button1: @PushButton(# #);
        button2: @PushButton(# #);

```



```

button3: @PushButton(# #);
init:<
(
do button1.init;
  button2.init;
  button3.init;
  XmATTACH_FORM -> button1.topAttachment;
  XmATTACH_FORM -> button1.leftAttachment;
  XmATTACH_FORM -> button1.rightAttachment;
  XmATTACH_WIDGET -> button2.topAttachment;
  button1 -> button2.topWidget;
  XmATTACH_FORM -> button2.leftAttachment;
  XmATTACH_FORM -> button2.rightAttachment;
  XmATTACH_WIDGET -> button3.topAttachment;
  button2 -> button3.topWidget;
  XmATTACH_FORM -> button3.leftAttachment;
  XmATTACH_FORM -> button3.rightAttachment;
  XmATTACH_FORM -> button3.bottomAttachment;
#);
do window.init
#)

```

In general specific constraints may be put on each edge of the children. Here one PushButton - button1 - is attached to the Form itself on three sides. This means that button1 will attempt to resize these three edges if the Form is resized. A second PushButton - button2 - is then attached vertically to button1: Its topAttachment resource is specified as the constant XmATTACH\_WIDGET, and button1 is then specified as the topWidget of button2. Likewise a third button - button3 - is attached vertically to button2. When the application is run, the following window appears:



Several special variants of the RowColumn widget are supplied by MotifEnv. This includes various menus described later, and it also includes the pattern RadioBox, used to create a traditional "radio panel" of buttons. The following example demonstrates the RadioBox pattern:

#### **radiobox.bet**

```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/rowcolumn';
INCLUDE '~beta/Xt/motif/togglebuttongadget';
INCLUDE '~beta/Xt/motif/pushbuttongadget';

-- PROGRAM: descriptor --
MotifEnv
(
  main: @RowColumn
    (
      panel: @RadioBox
        (
          r1, r2, r3, r4, r5, r6: @ToggleButtonGadget
            (
              valueChangedCallback:<
                (
                  do name -> screen.puttext;
                  (if data.set then 'set' -> putLine
                     else 'unset' -> putLine
                     if)
                )
            )
        )
    )
)

```

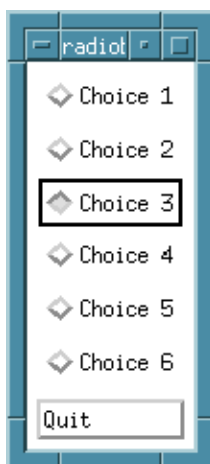


```

        #);
    #);
    init::<
    (#
    do ('Choice 1', panel) -> r1.init;
    ('Choice 2', panel) -> r2.init;
    ('Choice 3', panel) -> r3.init;
    ('Choice 4', panel) -> r4.init;
    ('Choice 5', panel) -> r5.init;
    ('Choice 6', panel) -> r6.init;
    #);
    #);
    quit: @PushButtonGadget
    (# init::< (# do 'Quit' -> labelString #);
    activateCallback::< (# do stop #)
    #);
    init::< (# do panel.init; quit.init #);
    #)
do main.init
#)

```

When this program is run, the window shown below with six `ToggleButtonGadgets` and one `PushButtonGadget` (Quit) appears. The window is shown in a state, where the third item is selected. Notice, that the button appears to be pushed inwards.



Many standard applications consists of one main window, with a menu bar with menus and a working area that can be scrolled. For this purpose, a `MainWindow` widget with support for this setup is provided. The following is an excerpt of an application using the `MainWindow` widget:

#### **mainwindow.bet**

```

ORIGIN '~beta/Xt/allmotif'
--PROGRAM: descriptor--
MotifEnv
(# main: @MainWindow
  (# work: @ArrowButtonGadget;
    mbar: @MenuBar
    (# specification of menu bar not shown ... #);
    init::<
    (#
    do mbar.init;      work.init;
    200 -> height; 300 -> work.height;
    200 -> width;  300 -> work.width;
    #);
  #);
#);

```



```

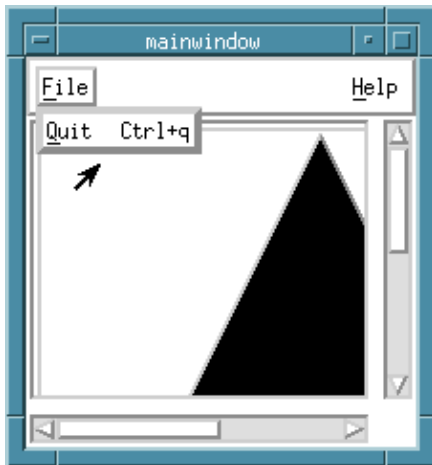
fallbackResources::<
  (# init::<
    (#
      do '*XmMainWindow.scrollingPolicy: XmAUTOMATIC'
        -> addtext
      #);
    #);
do main.init;
#)

```

For work area, an ArrowButtonGadget is used. This could be any other widget. In the initialization, the ArrowButtonGadget is made bigger than the working area of the MainWindow. This will force the scroll bars to appear.

The scrollingPolicy resource of the MainWindow cannot be changed on the fly, but only at creation time. Thus the easiest way to do it, is to specify it in a standard X Toolkit resource file, but the fallbackResources virtual of XtEnv can also be used as is done here. Setting the scrollingPolicy to XmAUTOMATIC will make scrollbars appear when needed.

When the program is run the following window appears:



The window is shown in a situation, where the File menu has been posted. It will later be shown how to program the menu bar with this File menu.

The next example shows how to use the special purpose Scale widget, useful for adding, e.g., a potentiometer-like control to a panel of controls.

#### **scale.bet**

```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/scale';
-- PROGRAM: descriptor --
MotifEnv
  (# volume: @Scale
    (# init::<
      (#
        do 0 -> minimum;
          100 -> maximum;
          XmVERTICAL -> orientation;
          'Volume' -> titleString;
          true -> showValue;
        #);
      valueChangedCallback::<

```

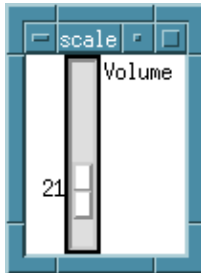


```

    (#
    do 'New volume: ' -> screen.puttext;
      data.value -> screen.putint;
      screen.newline;
    #);
  #);
do volume.init;
#)

```

The following window will appear when the program is run:



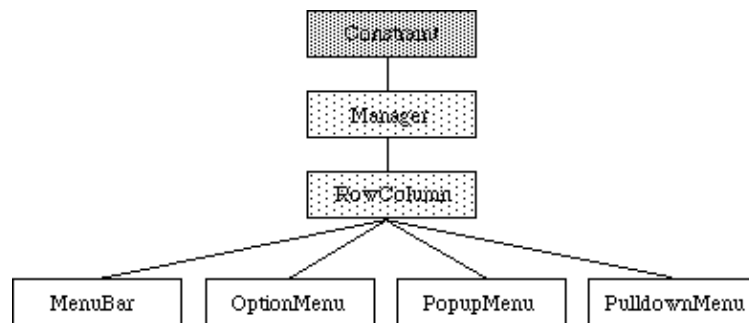
The `valueChangedCallback` is called after the scale has been changed. In this case the callback data is furtherbound to `XmScaleCallbackStruct`, that has an attribute called `value`, that contains the current value of the Scale.

---



# Motif Menus

The following figure shows the menu-related patterns of MotifEnv. Patterns from XtEnv are shaded gray, and other MotifEnv patterns are shaded light-gray:



The following is a brief presentation of the menu-related patterns:

- **PopupMenu**: Utility pattern used to instantiate a RowColumn widget of type XmMENU\_POPUP. When the Popup MenuPane is created, a MenuShell widget is automatically created as the parent of the MenuPane. The PopupMenu is used as the first MenuPane within a PopupMenu system; all other MenuPanes are of the Pulldown type. A Popup MenuPane displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the MenuPane. A PopupMenu is "popped up" using `manageChild`, and "popped down" using `unmanageChild`. The menu can be positioned before being managed using the local position pattern.
- **PulldownMenu**: Utility pattern to create a RowColumn of type XmMENU\_PULLDOWN. When creating a PulldownMenu, a MenuShell is automatically created as the parent of PulldownMenu. If the parent specified is a PopupMenu or a PulldownMenu, the MenuShell is created as a child of the parent's MenuShell; otherwise, it is created as a child of the specified parent widget. A PulldownMenu displays a 3-D shadow, unless the feature is disabled by the application. The shadow appears around the edge of the PulldownMenu. A PulldownMenu is used when creating submenus that are to be attached to a CascadeButton or a CascadeButtonGadget. This is the case for all MenuPanes that are part of a PulldownMenu system (a MenuBar), the MenuPane associated with an OptionMenu, and any MenuPanes that cascade from a Popup MenuPane.
- **PulldownMenus** that are to be associated with an OptionMenu must be created before the OptionMenu is created.
- The PulldownMenu must be attached to a CascadeButton or CascadeButtonGadget that resides in a MenuBar, a Popup MenuPane, a Pulldown MenuPane, or an OptionMenu. This is done by using the button resource `subMenuId`.
- To function correctly when incorporated into a menu, the PulldownMenu's hierarchy must be considered; this hierarchy depends on the type of menu system that is being built as follows:
  - ◆ If the PulldownMenu is to be pulled down from a MenuBar, its parent must be the MenuBar.



- ◆ If the PulldownMenu is to be pulled down from a PopupMenu or another PulldownMenu, its parent must be that PopupMenu or PulldownMenu.
  - ◆ If the PulldownMenu is to be pulled down from an OptionMenu, its parent must be the same as the OptionMenu parent.
- **OptionMenu:** Utility pattern for instantiating a RowColumn widget of type `XmMENU_OPTION`. An OptionMenu widget is a specialized RowColumn manager composed of a label, a selection area, and a single Pulldown MenuPane. When an application creates an OptionMenu widget, it supplies the label string and the PulldownMenu. In order to succeed, there must be a valid subMenuId resource set. When the OptionMenu is created, the PulldownMenu must have been created as a child of the OptionMenu's parent and must be specified. The LabelGadget and the selection area (a CascadeButtonGadget) are created by the OptionMenu. An OptionMenu is laid out with the label displayed on one side of the widget and the selection area on the other side. The selection area has a dual purpose; it displays the label of the last item selected from the associated PulldownMenu, and it provides the means for posting the PulldownMenu. The PulldownMenu is posted by moving the mouse pointer over the selection area and pressing a mouse button defined by OptionMenu's RowColumn parent. The PulldownMenu is posted and positioned so that the last selected item is directly over the selection area. The mouse is then used to arm the desired menu item. When the mouse button is released, the armed menu item is selected and the label within the selection area is changed to match that of the selected item. The OptionMenu also operates by using the keyboard interface mechanism. If the application has established a mnemonic with the OptionMenu, typing Alt with the mnemonic causes the PulldownMenu to be posted with traversal enabled. The standard traversal keys can then be used to move within the Menu. Selection can occur as the result of pressing the Return key or typing a mnemonic or accelerator for one of the menu items. An application may use the menuHistory resource to indicate which item in the PulldownMenu should be treated as the current choice and have its label displayed in the selection area. By default, the first item in the PulldownMenu is used.
  - **MenuBar:** Utility pattern for instantiating a RowColumn widget of type `XmMENU_BAR`. A MenuBar is generally used for building a Pulldown menu system. Typically, a MenuBar is created and placed along the top of the application window, and several CascadeButtons are inserted as the children. Each of the CascadeButtons has a PulldownMenu associated with it. These PulldownMenus must have been created as children of the MenuBar. The user interacts with the MenuBar by using either the mouse or the keyboard. A MenuBar displays a 3-D shadow along its border. The application controls the shadow attributes using the visual-related resources supported by Manager. A MenuBar widget is homogeneous in that it accepts only children that are specializations of CascadeButton or CascadeButtonGadget. Attempting to insert a child of a different class results in a warning message. If a MenuBar does not have enough room to fit all of its subwidgets on a single line, the MenuBar attempts to wrap the remaining entries onto additional lines if allowed by the geometry manager of the parent widget.



# Examples using Motif Menus

The following is an example of using an option menu.

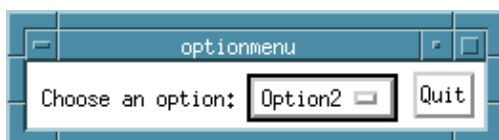
## **optionmenu.bet**

```
ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/rowcolumn';
INCLUDE '~beta/Xt/motif/pushbuttongadget';

-- PROGRAM: descriptor --
MotifEnv
(# main: @RowColumn
  (# menuPane: @PulldownMenu
    (# item1,item2,item3,item4: @PushButtonGadget
      (# activateCallback:<
        (# do name->puttext; ' selected'->putLine #);
      #);
    init:<
      (#
        do ('Option1', menuPane) ->item1.init;
        ('Option2', menuPane) ->item2.init;
        ('Option3', menuPane) ->item3.init;
        ('Option4', menuPane) ->item4.init;
      #);
    optMenu: @OptionMenu
      (# init:<
        (# subMenu:< (# do menuPane -> value #);
        do 'Choose an option:' -> labelString;
        menuPane.item2 -> menuHistory;
      #);
      #);
    quit: @PushButtonGadget
      (# init:< (# do 'Quit' -> labelString #);
      activateCallback:< (# do stop #)
      #);
    init:<
      (#
        do XmHORIZONTAL -> orientation;
        menuPane.init;
        optMenu.init;
        quit.init;
      #);
    #);
  do main.init;
#)
```

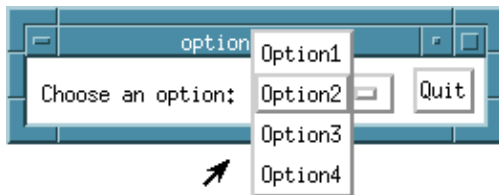
First a menu pane to use in the option menu is created: This is a PullDownMenu with four PushButtonGadgets as items. Then the option menu widget is created. This will glue the menu pane, a label and a button together. Notice, that specification of the menu pane for the option menu is done using the subMenu virtual local to the init virtual. This is because the OptionMenu widget needs to know what menu pane to use already when the OptionMenu is created.

When the program is run, the following window appears:





As can be seen, the OptionMenu starts out with Option2 selected. This was specified in the program using the menuHistory resource. When the OptionMenu button is clicked, the menu pops up, and a new option can be selected:



The following example shows how to create and use a PopupMenu:

### **popupmenu.bet**

```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/primitive';
INCLUDE '~beta/Xt/motif/rowcolumn';
INCLUDE '~beta/Xt/motif/separatorgadget';
INCLUDE '~beta/Xt/motif/pushbuttongadget';

-- PROGRAM: descriptor --
MotifEnv
(# theMenu: @PopupMenu
  (# item1,item2,item3,item4: @PushButtonGadget
    (# activateCallback:<
      (# do name -> puttext; ' selected' -> putLine #);
    #);
  quit: @PushButtonGadget
    (# activateCallback:< (# do stop #)#);
  init:<
    (# sep: ^SeparatorGadget;
    do ('Item1', theMenu) -> item1.init;
    ('Item2', theMenu) -> item2.init;
    ('Item3', theMenu) -> item3.init;
    ('Item4', theMenu) -> item4.init;
    &SeparatorGadget[] -> sep[]; sep.init;
    ('Quit', theMenu) -> quit.init;
    #);
  #);
main: @Primitive
  (# eventHandler:<
    (# bp: @ButtonPress
      (#
        do event -> theMenu.position;
        theMenu.manageChild
      #);
    init:< (# do bp.enable #);
    #);
  init:< (# do 100 -> width; 100 -> height;#);
  #);
do main.init;
('theMenu', main) -> theMenu.init;
#)

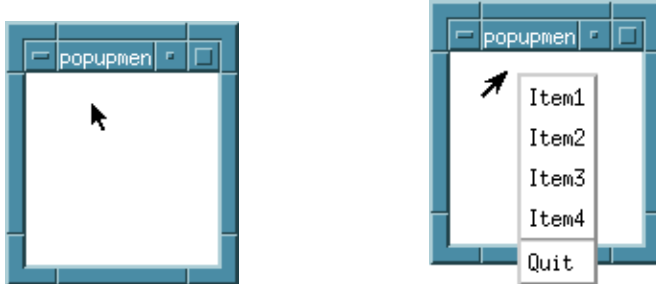
```

The PopupMenu is created having five PushButtonGadgets and one SeparatorGadget as children. In the example, this menu is to be popped up from a Primitive. This is done, by further binding the eventhandler of the Primitive: a ButtonPress eventprocessor is instantiated and enabled. When a button is pressed within the Primitive, the do-part of this eventprocessor is invoked. This will do two things:



1. The event causing the invocation is handled to the local position pattern of the PopupMenu. This will place the menu in a way so that the first menu item is just where the mouse was pressed. The position pattern uses informations in the event object for this.
2. The menu is popped up. This is done by using the manageChild attribute.

When the program is executed, the window shown left below appears. When a mouse button is then pressed, the menu pops up as shown in the right window below.



The next example shows how to create a pulldown menu system using a MenuBar, and also it shows how to create hierarchical menus using CascadeButtonGadgets.

#### **pulldownmenu.bet**

```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/rowcolumn';
INCLUDE '~beta/Xt/motif/cascadebutton';
INCLUDE '~beta/Xt/motif/pushbuttongadget';
INCLUDE '~beta/Xt/motif/cascadebuttongadget';

-- PROGRAM: descriptor --
MotifEnv
(# menus: @MenuBar
  (# menubutton: @CascadeButton
    (* The button that activates the menu *)
    (# init::< (# do 'Menu' -> labelString #)#);
    menu: @PullDownMenu
    (# item1: @PushButtonGadget
      (# activateCallback::<
        (# do 'Item1 selected' -> putLine #);
      #);
      item2: @CascadeButtonGadget
      (# submenu: @PullDownMenu
        (# cascade1: @PushButtonGadget
          (# activateCallback::<
            (# do 'Cascade 1 selected' -> putLine #);
          #);
          cascade2: @PushButtonGadget
          (# activateCallback::<
            (# do 'Cascade 2 selected' -> putLine #);
          #);
          init::<
            (# do cascade1.init; cascade2.init #);
          #);
          init::< (# do submenu.init; submenu->subMenuId #);
        #);
        quit: @PushButtonGadget
        (# activateCallback::< (# do stop #)#);
        init::< (# do item1.init; item2.init; quit.init #);
      #);
    init::<
      (#

```



```

do menubutton.init;
    menu.init;
    menu -> menubutton.subMenuId;
#)#);
do menus.init;
#)

```

A MenuBar can contain CascadeButtons, each having a PullDownMenu attached. In the example only one CascadeButton is created, thus the menu bar will only contain one menu. The CascadeButton - menubutton - is initialized first. Its label string is set to "Menu". Then the PullDownMenu is initialized. It contains three items, the first and last one being normal PushButtonGadgets. The second item, however, is a CascadeButtonGadget. This is used to bind a sub menu to the main menu. The submenu is a normal PullDownMenu, which is initialized, and then attached to the CascadeButtonGadget using the subMenuId resource of the CascadeButtonGadget. Finally the main menu is attached to the CascadeButton in the MenuBar, by using the submenuId resource of the CascadeButton in the MenuBar.

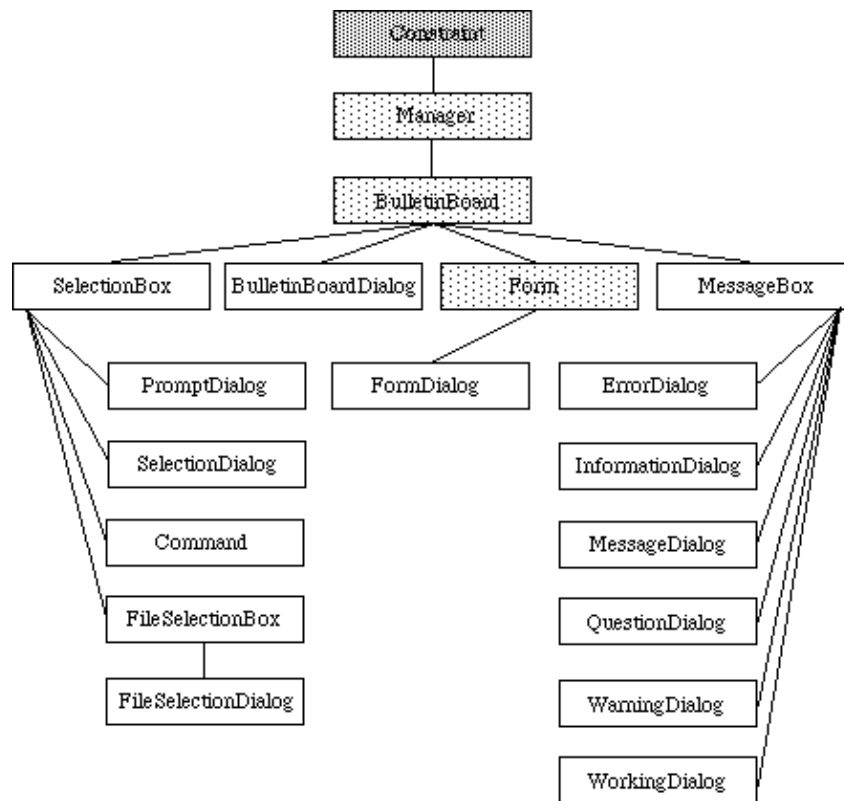
When the program is executed, a window containing a menu bar with just one button appears. Below in the window shown left, the menu button has been activated, and the menu has been posted. Notice the arrow in the second menu item, indicating that the item has a sub menu. If the second item is activated, the submenu will pop up, as shown in the right window below.





# Motif Dialog Patterns

The following figure shows the patterns constituting the MotifEnv dialog patterns. Patterns from XtEnv are shaded gray, and other MotifEnv patterns are shaded light-gray.



The following is a brief presentation of the different patterns:

- **SelectionBox**: a general dialog widget that allows the user to select one item from a list. A SelectionBox includes the following:
  1. a scrolling list of alternatives,
  2. an editable text field for the selected alternative,
  3. labels for the list and text field, and
  4. three or four buttons. The default button labels are OK, Cancel, and Help. By default an Apply button is also created; if the parent of the SelectionBox is a DialogShell it is managed, and otherwise it is unmanaged.

One additional WorkArea child may be added to the SelectionBox after creation. The user can select an item in two ways: by scrolling through the list and selecting the desired item or by entering the item name directly into the text edit area. Selecting an item from the list causes that item name to appear in the selection text edit area. The user may select a new item as many times as desired. The item is not actually selected until the user presses the OK PushButton.



- **PromptDialog:** Utility pattern to instantiate a DialogShell and an unmanaged SelectionBox child of the DialogShell. A PromptDialog prompts the user for text input. It includes a message, a text input region, and three managed buttons. The default button labels are OK, Cancel, and Help. An additional button, with Apply as the default label, is created unmanaged; it may be explicitly managed if needed.
- **SelectionDialog:** Utility pattern to instantiate a DialogShell and an unmanaged SelectionBox child of the DialogShell. A SelectionDialog offers the user a choice from a list of alternatives and gets a selection. It includes the following:
  1. a scrolling list of alternatives,
  2. an editable text field for the selected alternative,
  3. labels for the text field, and
  4. four buttons. The default button labels are OK, Cancel, Apply, and Help.
- **FileSelectionBox:** used to traverse through directories, view the files and subdirectories in them, and then select files. A FileSelectionBox has five main areas:
  1. a text input field for displaying and editing a directory mask used to select the files to be displayed,
  2. a scrollable list of filenames,
  3. a scrollable list of subdirectories,
  4. a text input field for displaying and editing a filename, and
  5. a group of PushButtons, labeled OK, Filter, Cancel, and Help.

The list of filenames, the list of subdirectories, or both can be removed from the FileSelectionBox after creation by using the patterns unmanageFileNames and unmanageSubdirectories.
- **FileSelectionDialog:** utility pattern to instantiate a DialogShell and an unmanaged FileSelectionBox child of the DialogShell.
- **Command:** a special-purpose composite widget for command entry that provides a built-in command-history mechanism. Command includes a command-line text-input field, a command-line prompt, and a command history list region. Whenever a command is entered, it is automatically added to the end of the command-history list and made visible. This does not change the selected item in the list, if there is one.
- **MessageBox:** MessageBox is a dialog pattern used for creating simple message dialogs. Specializations based on MessageBox are provided for several common interaction tasks, which include giving information, asking questions, and reporting errors. A MessageBox dialog is typically transient in nature, displayed for the duration of a single interaction. A MessageBox can contain a message symbol, a message, and up to three standard default PushButtons: OK, Cancel, and Help. It is laid out with the symbol and message on top and the PushButtons on the bottom. The Help button is positioned to the side of the other push buttons.
- **ErrorDialog:** Utility pattern to instantiate a DialogShell and an unmanaged MessageBox child of the DialogShell. An ErrorDialog warns the user of an invalid or potentially dangerous condition. It includes a symbol, a message, and three buttons. The default symbol is an octagon with a diagonal slash. The default button labels are OK, Cancel, and Help.
- **InformationDialog:** Utility pattern to instantiate a DialogShell and an unmanaged



MessageBox child of the DialogShell. An InformationDialog presents a short information to the user. It includes a symbol, a message, and three buttons. The default symbol is lower case i. The default button labels are OK, Cancel, and Help.

- **MessageDialog:** Utility pattern to instantiate a DialogShell and an unmanaged MessageBox child of the DialogShell. A MessageDialog gives the user some message. It includes a symbol, a message, and three buttons. By default there is no symbol. The default button labels are OK, Cancel, and Help.
- **QuestionDialog:** Utility pattern to instantiate a DialogShell and an unmanaged MessageBox child of the DialogShell. A QuestionDialog asks the user a question. It includes a symbol, a message, and three buttons. The default symbol is a question mark. The default button labels are OK, Cancel, and Help.
- **WarningDialog:** Utility pattern to instantiate a DialogShell and an unmanaged MessageBox child of the DialogShell. A WarningDialog warns the user of an invalid or potentially dangerous condition. It includes a symbol, a message, and three buttons. The default symbol is an exclamation point. The default button labels are OK, Cancel, and Help.
- **WorkingDialog:** Utility pattern to instantiate a DialogShell and an unmanaged MessageBox child of the DialogShell. A WorkingDialog informs the user, that some lengthy computations is going on, for instance. It includes a symbol, a message, and three buttons. The default symbol is an hourglass. The default button labels are OK, Cancel, and Help.
- **FormDialog:** Utility pattern to instantiate a DialogShell and an unmanaged Form child of the DialogShell. A FormDialog is used for interactions not supported by the standard dialog set. It does not include any labels, buttons, or other dialog components. Such components should be added to the FormDialog by the application.
- **BulletinBoardDialog:** Utility pattern to instantiate a DialogShell and an unmanaged BulletinBoard child of the DialogShell. A BulletinBoardDialog is used for interactions not supported by the standard dialog set. It does not include any labels, buttons, or other dialog components. Such components should be added to the BulletinBoardDialog by the application.



## Examples using Motif Dialog Patterns

The following example completes the [MainWindow example](#) shown earlier, by adding the menu bar, and creating a MessageDialog to pop up when the help button in the menu bar is activated. **mainwindow.bet, continued**

```
mbar: @MenuBar
  (# fileButton: @CascadeButton
    (# init::<
      (# do 'File' -> labelString; 'F' -> mnemonic #)
    #);
  fileMenu: @PullDownMenu
    (# quit: @PushButtonGadget
      (# init::<
        (#
          do 'Quit' -> labelString;
          'Q' -> mnemonic;
          'Ctrl<Key>q' -> accelerator;
          'Ctrl+q' -> acceleratorText;
        #);
        activateCallback::< (# do stop #);
      #);
      init::< (# do quit.init #);
    #);
  helpButton: @CascadeButton
    (* The button in the MenuBar that activates the HelpBox *)
    (# helpBox: @MessageDialog
      (* The dialog that displays the help message *)
      (# init::<
        (#
          do 'This "help" should be extended!'
            -> messageString;
          'Help' -> dialogTitle;
          unManageCancel; unManageHelp;
        #);
        #);
        activateCallback::< (# do helpBox.manageChild #);
        init::<
          (#
            do 'Help' -> labelString; 'H' -> mnemonic;
            helpBox.init
          #);
        #);
      init::<
        (#
          do fileButton.init; fileMenu.init;
          fileMenu -> fileButton.subMenuId;
          helpButton.init;
          helpButton -> menuHelpWidget;
        #);
    #);
```

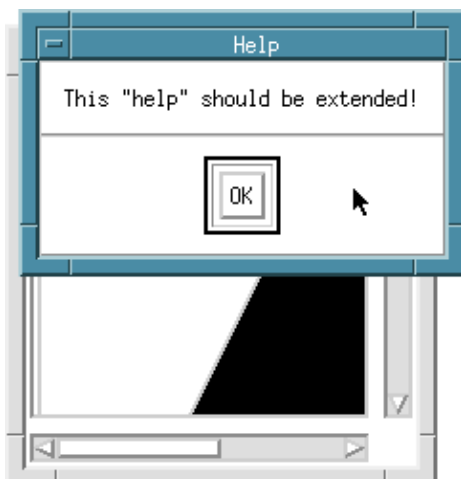
The example shows how to create the MenuBar and File menu for the MainWindow. Except for the use of mnemonics and accelerators, this has been explained in the section on [Menus](#) earlier in the manual. A mnemonic is a "keyboard equivalent", which can be used to invoke menus and menu items, when the menus are posted. Accelerators can be used to invoke menu items even when the menu containing the item is not posted.

The help dialog is invoked from a CascadeButton called helpButton in the MenuBar. Two special things happen in the init virtual of the MessageDialog:



1. The dialogTitle resource is set to the text "Help". This is because some window managers decorate the dialogs with title bar etc. The dialogTitle resource determines what will appear in the title bar in that case.
2. A MessageDialog by default contains three buttons: an OK button, a Cancel button, and a Help button. The Cancel and Help buttons do not make much sense within a help dialog, so these are removed using the unmanageCancel and unmanageHelp patterns. This is the easiest way to make a dialog with just an OK button, since, unfortunately, Motif does not include such a dialog.

The windows below show how the help dialog looks, when popped up from the mainwindow application.



The following example shows how to use the FileSelectionBox pattern to invoke a standard file specification dialog:

#### **filedialog.bet**

```

ORIGIN '~beta/Xt/motifenv';
INCLUDE '~beta/Xt/motif/fileselectionbox';
INCLUDE '~beta/Xt/motif/rowcolumn';
INCLUDE '~beta/Xt/motif/pushbutton';

-- PROGRAM: descriptor --
MotifEnv
(
  buttons: @RowColumn
    (
      (# fs: @FileSelectionDialog
        (# init::< (# do 'Enter File' -> dialogTitle #);
          OkCallback::<
            (# do dirSpec->putline; fs.unManageChild #);
          CancelCallback::<
            (# do 'Cancel'->putline; fs.unManageChild #);
          HelpCallback::<
            (# do 'Sorry, no help available'->putline #);
        #);
      getfile: @PushButton
        (# activateCallBack::< (# do fs.manageChild #));
      quit: @PushButton
        (# activateCallBack::< (# do stop #));
      init::<
        (
          (#
            do fs.init; getfile.init; quit.init;
          #);
    );

```



```

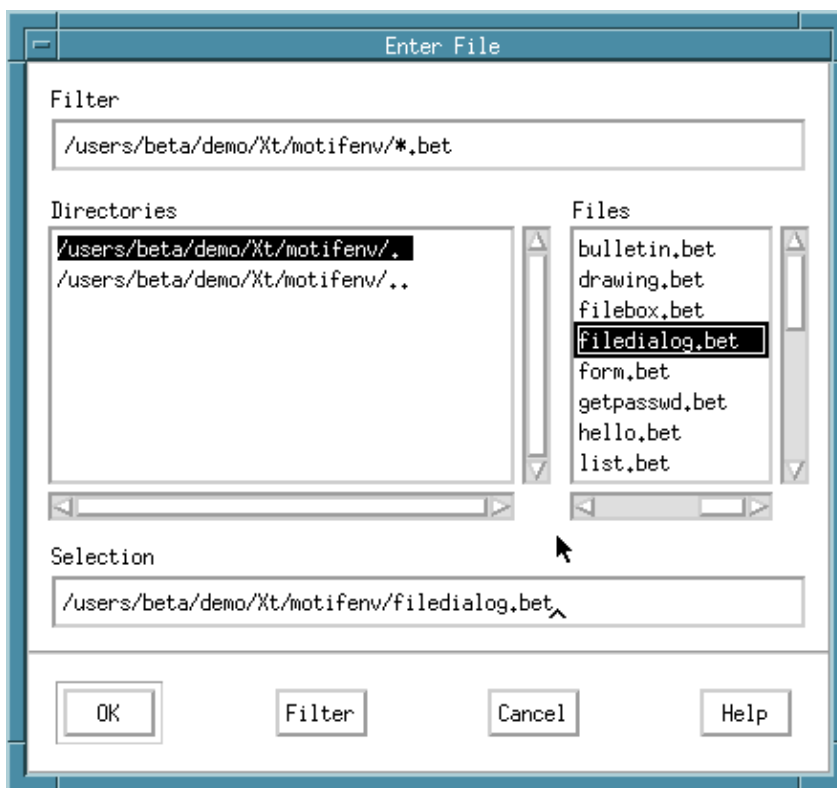
#);
do buttons.init;
#)

```

The main RowColumn contains two PushButtons:



The FileSelectionBox is popped up using the manageChild attribute. The dialog looks like this:



Three callbacks are further bound in the FileSelectionDialog: When OK is pushed, the file name selected is printed out. This is obtained using the dirSpec resource. Then the dialog is popped down using the unmanageChild attributes. If Cancel is pushed, "Cancel" is printed, and the dialog is popped down. If Help is pushed, a small disclaimer is printed out. This could easily be changed to pop up a real help dialog as in the previous example. The Filter button is used to apply the regular expression in the topmost text entry field to the list of files displayed in the dialog.



MotifEnv



## Other MotifEnv Patterns

The user may in some situations want to create a shell for a special menu or dialog type. This can be done using the MenuShell and DialogShell respectively. However, because of the wide variety of utility patterns to create menus and dialogs, normally the user will not need to use these two patterns directly.

- **MenuShell**: a custom OverrideShell widget. An OverrideShell widget bypasses the window manager when displaying itself. It is designed specifically to contain Popup or Pulldown MenuPanes. Most application writers never encounter this widget if they use the patterns PopupMenu or PulldownMenu, which automatically create a MenuShell widget as the parent of the MenuPane. However, if these patterns are not used, the application programmer must create the required MenuShell. In this case, it is important to note that the type of parent of the MenuShell depends on the type of menu system being built.
    - ◆ If the MenuShell is for the top-level Popup MenuPane, the MenuShell must be created as a child of the widget from which the Popup MenuPane is popped up.
    - ◆ If the MenuShell is for a MenuPane that is pulled down from a Popup or another Pulldown MenuPane, the MenuShell must be created as a child of the Popup or Pulldown MenuPane.
    - ◆ If the MenuShell is for a MenuPane that is pulled down from a MenuBar, the MenuShell must be created as a child of the MenuBar.
    - ◆ If the MenuShell is for a Pulldown MenuPane in an OptionMenu, the MenuShell's parent must be the OptionMenu.
  - **DialogShell**: Modal and modeless dialogs use DialogShell as the Shell parent. DialogShell widgets cannot be iconified. Instead, all secondary DialogShell widgets associated with an ApplicationShell widget are iconified and de-iconified as a group with the primary widget. A client indirectly manipulates a DialogShell via the different dialog patterns, and it can directly manipulate its BulletinBoard-derived child. Much of the functionality of DialogShell assumes that its child is a BulletinBoard, although it can potentially stand alone.
-



# XSystemEnv

When concurrency is used by means of the SystemEnv pattern described in [\[MIA 90-8\]](#), in conjunction with XtEnv, the XSystemEnv pattern located in the xsystemenv fragment should be used.

A program using xsystemenv should look something like:

```
ORIGIN 'xsystemenv';
-- program:descriptor --
systemEnv
  (# setWindowEnv::< (# do myWindowEnv[] -> theWindowEnv[] #);
    myWindowEnv: @MotifEnv (# ... #);
    ...
  #)
```

The setWindowEnv virtual and theWindowEnv reference are declared in basicsystemenv. The theWindowEnv reference does not have to be to a motifenv instance as long as it is at least an xtenv instance (e.g. awenv, motifenv, or xtenv).

The xtenv instance assigned to theWindowEnv is used for scheduling purposes to allow BETA coroutines to cooperate with the X event driven user interface.

For concurrency details, see BasicSystemEnv in [\[MIA 90-8\]](#).

---



# Examples of using XSystemEnv

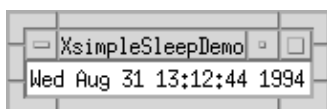
The following simple program uses a Motif Label widget for showing the current time and date in a window. It uses a System coroutine to update the clock once per second, by sleeping between each update of the clock.

## XsimpleSleepDemo.bet:

```
ORIGIN    '~beta/Xt/xsystemenv';
INCLUDE   '~beta/Xt/motifenv';
INCLUDE   '~beta/Xt/motif/label';
INCLUDE   '~beta/sysutils/time.bet';

--- program:descriptor ---
systemEnv
(# setWindowEnv::<
  (# do mymotif[] -> theWindowEnv[] #);
  updateClock: @|System
    (#
      do cycle
        (#
          do 1 -> sleep;
          systemTime -> formatTime -> mymotif.clock.labelString;
        #);
      #);
  mymotif: @motifEnv
    (# clock: @label
      (# init::< (# do systemTime->formatTime->labelString #)#);
      do clock.init;
      #);
do updateClock[] -> fork;
#)
```

When this program is run, the following window appears, in which the time is updated every second.



The same effect could be obtained by using a WorkProc.



# Miscellaneous

In the two directories `misc` and `demo/misc`, a few X related extras are placed.

## Xterm Interface

You can set the title and icon name of the xterm you are executing in, and you can start another UNIX process in a separate xterm window. The demo in

```
$BETALIB/demo/Xt/misc/xtermdemo.bet
```

shows examples of both.

## EditRes Interface

The `editres` program is a standard program distributed with the X Window System. It allows graphical browsing and changing of a program's X resources.

To support the `editres` program, the program must support a special "EditRes Protocol". To enable this protocol, the `enableEditRes` method in

```
$BETALIB/Xt/misc/editres.bet
```

should be invoked. After this, the program's resources can be manipulated by `editres`.

---

X Libraries - Reference Manual

[Mjølner Informatics](#)

.

X Libraries - Reference Manual



# Bibliography

[Nye & O'Reilly 90a]

Adrian Nye & Tim O'Reilly:  
*X Toolkit Intrinsic Programming Manual*,  
Volume Four of *The X Window System Series*,  
O'Reilly & Associates Inc, 1990, ISBN 0-937175-34-X

[Nye & O'Reilly 90b]

Adrian Nye & Tim O'Reilly:  
*X Toolkit Intrinsic Programming Manual, OSF/Motif 1.1 Edition*,  
Volume Four (Motif) of *The X Window System Series*,  
O'Reilly & Associates Inc, 1990, ISBN 0-937175-62-5

[Young 90]

Douglas A. Young:  
*The X Window System. Programming and Applications with Xt, OSF/Motif Edition*,  
Prentice Hall, 1990, ISBN 0-13-497074-8

[Jones 89]

Oliver Jones:  
*Introduction to the X Window System*,  
Prentice Hall, 1989, ISBN 0-13-499997-5

[Mansfield 89]

Niall Mansfield:  
*An X Window System User's Guide*,  
Addison-Wesley, Amsterdam, 1989, ISBN 0-201-51341-2.

[X WWW]

Kenton Lee:  
*Technical X Window System and Motif WWW Sites*,  
<http://www.rahul.net/kenton/xsites.html>.

[Motif WWW]

MW3:  
*Motif on the World Wide Web*,  
<http://www.cen.com/mw3>.



# XtEnv Interface

```
ORIGIN 'xtlib';
BODY 'private/xtenvbody';
BODY 'private/errorhandlerbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

--- LIB: attributes ---

XtEnv: XtLib
(* XtEnv is the BETA interface to the X Toolkit Intrinsics(Xt). Xt
 * contains the basic classes common for many user-interface
 * toolkits build on top of X windows, but does not contain any
 * higher level user interface elements. It is typically used
 * together with a widget set containing such user interface
 * elements build on top of the Intrinsics.
 *)
(# <<SLOT xtenvLib: attributes>>;

topLevel: @ToplevelShell;
(* The default top level shell created and initialized at
 * startup of THIS(XtEnv)
 *)
fallbackResources: <
(* Specification of resources being set at startup of
 * THIS(XtEnv). In init, resources may be specified like
 * '*button.label: Push Me' -> addText; i.e. very much the same
 * way as resources are specified in your .Xdefaults file.
 *)
StringArray(##);
options:< XrmOptionDescRecList
(* Furtherbinding of this virtual lets the resource manager of
 * Xt set values for additional resources from the command
 * line.
 * Example:
 *   init:<
 *     (#
 *       do ('-bp', '*borderPixmap', XrmoptionsSepArg, '')->add
 *       #)
 *   will allow the "borderPixmap" resource to be specified by
 *   the "-bp" option.
 *)
(# init:< object #);
appName:<
(* The name of the application executing THIS(XtEnv). If not
 * set by a further binding of this virtual, the first command
 * line argument - i.e. the program name - is used (leading
 * path component is stripped)
 *)
(# value: ^text;
do INNER; ...;
exit value[]
#);
appClass:<
(* The class name of the application executing THIS(XtEnv). If
 * not set by a further binding of this virtual, the first
 * command line argument - i.e. the program name - is used
 * (leading path component is stripped, and the first letter is
 * capitalized. If the first letter is 'x', the second letter
```



```

    * is capitalized too.
    *)
    (# value: ^text;
    do INNER; ...;
    exit value[]
    #);
appContext: (* The "Application Context" of THIS(XtEnv) *)
    (# value: @XtAppContext;
    ...
    exit value
    #);

(* X Toolkit Exceptions *)
XtException: Exception
    (# do ...; INNER #);
XtObjectInstantiation:< XtException
    (* Raised on attempt to instantiate an XtObject *)
    (# do ...; INNER #);
RectObjectInstantiation:< XtException
    (* Raised on attempt to instantiate a RectObject *)
    (# do ...; INNER #);
FatherNotCore:< XtException
    (* Raised from XtObject.init, if the father specified is not a
    * Core or specialization thereof.
    *)
    (# name: ^text
    enter name[]
    do ...; INNER
    #);
MultipleToplevelChildren:< XtException
    (* Raised if more than one non-shell widget has specified
    * Toplevel as father.
    *)
    (# name: ^text
    enter name[]
    do ...; INNER
    #);
NoToplevelSon:< XtException
    (* Raised if no non-shell widget uses Toplevel as father *)
    (# do ...; INNER #);
ResourceSetError:< XtException
    (* Raised on attempt to set a resource for an uninitialized
    * widget
    *)
    (# do ...; INNER #);

ErrorHandler:
    (* Errors arising from the X Toolkit itself can be caught by
    * instatiating and installing an instance of ErrorHandler
    * before any widget is initialized.
    *)
    (# xterror:< XtException
        (# message: ^text;
        enter message[]
        do '\nX Toolkit Error: ' -> msg.puttext;
        message[] -> msg.putline;
        INNER;
        #);
        xliberror:< Exception
        (# message: ^text;
        enter message[]
        do 'Xlib Error: ' -> msg.puttext;
        message[] -> msg.putline;
        INNER;
        #);
    install: ...

```



```
#);
```

#### WarningHandler:

```
(* Warnings arising from the X Toolkit itself can be caught by
 * instatiating and installing an instance of WarningHandler
 * before any widget is initialized.
 *)
(# xtwarning:< XtException
  (# message: ^text;
   enter message[]
   do '\nX Toolkit Warning: ' -> msg.puttext;
    message[] -> msg.putline;
    INNER;
  #);
install: ...
#);
```

#### XtObject:

```
(* Models widgets and gadgets, and defines callback handling.
 * Widgets are specializations of Core whereas gadges only
 * inherit from the XtObject specialization RectObj. XtObject
 * is an ABSTRACT class, do not try to instantiate directly
 * from it.
 *)
(# <<SLOT XtObjectLib: attributes>>;

thewidget: @widget;

callbacks: @...;

parent: IntegerValue
  (* Returns the parent of THIS(XtObject) *)
  (# do thewidget -> XtParent -> value #);
name:
  (* Returns the name of THIS(XtObject) *)
  (# n: ^text;
   ...
   exit n[]
  #);

init:<
  (* Initialization method for THIS(XtObject). This method
   * must be called before anything can be done with
   * THIS(XtObject). It calls INNER such that
   * specializations can add to the initialization
   * behaviour.
   *
   * init has an enter-part, but when calling init, it is
   * optional to specify the enter-part. The enter-part
   * consists of two parameters: the father-widget and the
   * name of THIS(XtObject).
   *
   * The father-widget is the widget, that shall contain
   * this widget as a child. If the enter-part is not
   * specified, the father-widget will be the enclosing
   * widget of this widget according to BETAs scope-rules,
   * i.e. the Core, which statically encloses the pattern,
   * THIS(XtObject) is an instance of. If no such enclosing
   * widget is found, topLevel is used. If THIS(XtObject)
   * wants to have a father-widget, that it is not defined
   * within the scope of, the father-widget has to be
   * specified in the enter-part.
   *
   * The name of the XtObject is very important for the
   * internal working of Xt as it is through this name,
   * XtObjects are accessed. But the name is seldom
```



```

* important for the BETA-programmer. For some widgets
* the name is used as the default value for some of the
* widgets attributes. E.g. the name of a menu-item is
* used as the default value for the item-text presented
* to user when showing the menu. The name is also used if
* the programmer wants to change the default-values for
* some of attributes of some of the widgets used in the
* application. This can be done via a resourcefile. If
* the enter-part is not specified, the name will be the
* BETA-name of the descriptor for THIS(XtObject).
*)
(# name: ^text;
fatherWidget: @widget;

widgetClass:< IntegerObject
  (#
    do INNER; ...;
  #);
getFatherWidget:< IntegerObject;
  (* Virtual pattern that can be used to specify a
  * father for THIS(XtObject). If no father-widget is
  * specified in the enter part, getFatherWidget is
  * tried first, and only if this exits 0, the
  * statically enclosing widget is used.
  *)
doNotManageChild: @boolean;
  (* If this boolean is set to true, THIS(XtObject) is
  * not managed after INNER
  *)
enter (name[],fatherWidget)
do ...;
  INNER; ...;
#);
(* BETA interface to Xt resources *)
RoIntegerResource:
  (* Read Only *)
  (# get:
    (# value: @integer
      ...
      exit value
    #);
    resourceName:< IntegerObject;
  exit get
  #);
IntegerResource: RoIntegerResource
  (# set:
    (# value: @integer
      enter value
      ...
    #);
  enter set
  #);
RoShortResource:
  (# get:
    (# value: @int16
      ...
      exit value
    #);
    resourceName:< IntegerObject;
  exit get
  #);
ShortResource: RoShortResource
  (# set:
    (# value: @int16
      enter value
      ...

```



```

        #);
    enter set
    #);
RoCharResource:
    (* Read Only *)
    (# get:
        (# value: @char;
        ...
        exit value
        #);
        resourceName:< IntegerObject;
    exit get
    #);
CharResource: RoCharResource
    (# set:
        (# value: @char
        enter value
        ...
        #);
    enter set
    #);
RoBooleanResource:
    (* Read Only *)
    (# get:
        (# value: @boolean;
        ...
        exit value
        #);
        resourceName:< IntegerObject;
    exit get
    #);
BooleanResource: RoBooleanResource
    (# set:
        (# value: @boolean
        enter value
        ...
        #);
    enter set
    #);
StringResource:
    (# set:
        (# t: ^text
        enter t[]
        ...
        #);
    get:
        (# t: ^text;
        ...
        exit t[]
        #);
        resourceName:< IntegerObject;
        strprivate: @...;
    enter set
    exit get
    #);
StringArrayResource:
    (# set:
        (# theString: ^StringArray;
        enter theString[]
        ...
        #);
    get:
        (# theString: ^StringArray;
        ...
        exit theString[]
        #);

```



```

        resourceName:< IntegerObject;
enter set
exit get
#);

destroy:<
(* Removes THIS(XtObject) completely. Nothing can be done
 * with it afterwards, and no attributes can be accessed.
 *)
(# do INNER; ... #);
destroyCallback:< callbackProc
(* Virtual pattern corresponding to the Xt
 * "destroyCallbacks" resource. This virtual pattern will
 * be called when the widget is destroyed.
 *)
(# do INNER; ...; #);
callback:< callbackProc;
(* Virtual pattern corresponding to the "callback" Xt
 * resource of some widgets. This is a hook provided by many
 * widgets to allow applications to add functionality when
 * some widget-specific condition occur. E.g. a menu-item
 * widget will call this virtual method when the item has
 * been selected by the user.
 *)

(* Private *)
installCallbacks:<
(* Install various callbacks *)
(# do ...; INNER #);

exit theWidget
#) (* XtObject *);

RectObj: XtObject
(* Models widgets and gadgets, and defines geometry resources
 * and sensitivity. RectObj is an ABSTRACT class, do not try
 * to instantiate directly from it.
 *)
(# <<SLOT RectObjLib: attributes>>;

init:<
(# widgetClass:< (# do INNER; ... #);
do INNER;
#);

x:
(* The x position of the top-left corner of this widget
 * in the father-widget. Normally it has no effect to set
 * this resource, except for Shell widgets in cooperation
 * with certain window managers.
 *)
ShortResource(# resourceName:< XtNx #);
y:
(* The y position of the top-left corner of this widget
 * in the father-widget. Normally it has no effect to set
 * this resource, except for Shell widgets in cooperation
 * with certain window managers.
 *)
ShortResource(# resourceName:< XtNy #);
width:
(* The width of this widget. *)
ShortResource(# resourceName:< XtNwidth #);
height:
(* The height of this widget. *)
ShortResource(# resourceName:< XtNheight #);
borderwidth:

```



```

    (* The width of the border of this widget. *)
    ShortResource(# resourceName::< XtNborderwidth #);
ancestorSensitive:
    (* The sensitivity state of the ancestors of
    * THIS(RectObj). A RectObj is insensitive if either it or
    * any of its ancestors is insensitive. This attribute
    * cannot be changed, although it may be queried.
    *)
    RoBooleanResource(# resourceName::< XtNancestorSensitive #);
sensitive:
    (* A boolean which controls whether this(RectObj)
    * responds to user input. Input events will not be given
    * if either ancestorSensitive or sensitive is False.
    *)
    BooleanResource(# resourceName::< XtNSensitive #);

translateCoords:
    (* Coordinates are typically given in the local
    * coordinate-system of the window. This pattern
    * translates local coordinates to global coordinates.
    *)
    (# x,y: @integer;
    x2,y2: @shortRef
    enter (x,y)
    do (thewidget,x,y,x2[],y2[]) -> xtTranslateCoords;
    exit (x2,y2)
    #);
#); (* RectObj *)

Core: RectObj
    (* The Core widget class is the most fundamental widget class,
    * and serves as the superclass for all other widget
    * classes. The Core widget class is an abstract super
    * pattern. It is not designed to be used directly in
    * applications. The Core class defines characteristics common
    * to all widgets, in addition to the attributes inherited from
    * XtObject and RectObj.
    *)
    (# <<SLOT coreLib: attributes>>;

init::<
    (* Initialization method for THIS(Core). This method must
    * be called before anything can be done with
    * THIS(Core). It calls INNER such that specializations
    * can add to the initialization behaviour. See the
    * explanation on default father bindings in XtObject.init
    *)
    (# widgetClass::<
    (#
    do INNER;
    (if value=0 then CoreWidgetClass -> value if)
    #);
    do ev.init; (* Initialization of the eventhandler for
    * THIS(Core)
    *)
    INNER;
    #);

    (* Core resources *)
depth:
    (* Holds the number of bits per pixel used for indexing
    * into the colormap (read only).
    *)
    IntegerResource(# resourceName::< XtNDepth #);
mappedWhenManaged:
    (* If this resource is True, then the window of

```



```

    * THIS(Core) will automatically be mapped when it is
    * realized and managed (after INNER in init unless
    * doNotManage is true)
    *)
    BooleanResource(# resourceName::< XtNManagedWhenManaged #);
borderColor:
    (* A pixel value which indexes THIS(Core)'s colormap to
    * derive the border colour of THIS(Core)'s window.
    *)
    IntegerResource(# resourceName::< XtNBorderColor #);
borderPixmap:
    (* The border pixmap of THIS(Core)'s window. *)
    IntegerResource(# resourceName::< XtNBorderPixmap #);
colormap:
    (* The colormap that THIS(Core) will use. *)
    IntegerResource(# resourceName::< XtNColormap #);
screenResource:
    (* The screen where THIS(Core) is placed. *)
    IntegerResource(# resourceName::< XtNScreen #);
backGround:
    (* A pixel value which indexes THIS(Core)'s colormap to
    * derive the background colour of THIS(Core)'s window. You
    * can set either background or backgroundPixmap, but not
    * both. Whichever is set later takes priority.
    *)
    IntegerResource(# resourceName::< XtNBackGround #);
backGroundPixmap:
    (* The background pixmap of THIS(Core)'s window. *)
    IntegerResource(# resourceName::< XtNBackGroundPixmap #);
accelerators:
    (* A list of event to action bindings to be executed by
    * THIS(Core), even though the event occurred in another
    * widget.
    *)
    IntegerResource(# resourceName::< XtNaccelerators #);
translations:
    (* The event bindings associated with THIS(Core). See
    * also the translation handling routines below
    *)
    IntegerResource(# resourceName::< XtNtranslations #);
display: IntegerValue
    (* The display where THIS(Core) is placed. *)
    (# do theWidget -> xtDisplayOfObject -> value #);
window: IntegerValue
    (* The window associated with THIS(Core) *)
    (# do theWidget -> xtWindowOfObject -> value #);
screenOfObject: IntegerValue
    (# do theWidget -> xtScreenOfObject -> value #);

    (* Patterns for handling realization, mapping and geometry
    * management
    *)
realize:
    (* Realize THIS(Core) and all its child widgets on the
    * screen if the father widget of THIS(Core) is realized
    * or if it is toplevel.
    *)
    (# do theWidget -> xtRealizeWidget #);
unrealize:
    (* Removes THIS(Core) from the screen. All children of
    * THIS(Core) is also unrealized. Unrealizing a widget
    * also unmanaged it. Thus the geometry of its siblings
    * may be affected by the unrealization. An unrealized
    * widget can be realized again later.
    *)
    (# do theWidget -> xtUnRealizeWidget #);

```



```

map:
    (* Makes THIS(Core) visible, if it is already realized *)
    (# do theWidget -> xtMapWidget #);

unMap:
    (* Makes THIS(Core) invisible on the screen. The
     * difference from unRealize is, that a widget that is
     * unmapped is still managed, and the geometry of the
     * siblings of THIS(Core) will thus not change by
     * unmapping THIS(Core).
     *)
    (# do theWidget -> xtUnMapWidget #);

manageChild:
    (* Makes THIS(Core)'s geometry be managed *)
    (# do theWidget -> xtManageChild #);

unManageChild:
    (* Makes THIS(Core)'s geometry be unmanaged *)
    (# do theWidget -> XtUnmanageChild #);

(* Translation handling routines. See also
 * XtEnv.translationTable
 *)

overrideTranslations:
    (* Merge the new translations t into the existing widget
     * translations, ignoring any #replace, #augment, or
     * #override directives that may have been specified in
     * the translation string. If the new translations contain
     * an event or event sequence that already existed in
     * THIS(Core)'s translations, the new translations
     * override the existing ones.
     *)
    (# t: @integer enter t ... #);

augmentTranslations:
    (* Merge the new translations t into the existing widget
     * translations, ignoring any #replace, #augment, or
     * #override directives that may have been specified in
     * the translation string. If the new translations contain
     * an event or event sequence that already existed in
     * THIS(Core)'s translations, the new translations are
     * ignored
     *)
    (# t: @integer enter t ... #);

uninstallTranslations:
    (* Remove the entire translation table for THIS(Core) *)
    (# do theWidget -> xtunInstallTranslations #);

(* Installing accelerators *)
installAccelerators:
    (* Merge the accelerators of source with the translations
     * of THIS(Core). This will allow events occurring in
     * THIS(Core) to invoke actions in source
     *)
    (# source: @widget;
     enter source
     do (thewidget, source) -> XtInstallAccelerators
     #);

installAllAccelerators:
    (* Merge the accelerators of source and all subwidgets of
     * source with the translations of THIS(Core). This will
     * allow events occurring in THIS(Core) to invoke actions
     * in source
     *)
    (# source: @widget;
     enter source
     do (thewidget, source) -> XtInstallAllAccelerators
     #);

```



```

(* Various convenience routines for conversions *)
textToFont:
(* A procedure that converts a text specifying a name of
 * a font to something which can be used in
 * font-assignments.
 *)
(# fn: ^text;
  f: @integer;
  enter fn[]
  do (display, fn) -> XLoadQueryFont -> f;
  exit f
  #);

BitmapFileToPixmap:
(* Takes the name of a file which contains a bitmap to be
 * used as a pixmap. Reads in this file and returns
 * something that can be used in pixmap-assignments.
 *)
(# filename: ^text;
  pixmap: @integer;
  enter filename[]
  do ...;
  exit pixmap
  #);

BitmapFileToBitmap:
(* Takes the name of a file which contains a
 * bitmap. Reads in this file and returns something that
 * can be used in bitmap-assignments.
 *)
(# filename: ^text;
  bitmap: @integer;
  enter filename[]
  do ...;
  exit bitmap
  #);

SymbolToCursor:
(* Reads in one of the predefined cursors and returns
 * something that can be used in cursor-assignments. The
 * enter parameter is a constant that defines the
 * cursor. The constants XCXcursor, XCArrow, etc. can be
 * used. See xlib.bet for the complete list.
 *)
(# symbol: @integer;
  cursor: @integer
  enter symbol
  do (display, symbol) -> XCreateFontCursor -> cursor
  exit cursor
  #);

eventHandler:<
(* This virtual pattern allows the application program to
 * do something when a particular event occurs. It is a
 * low-level facility which is seldom needed as most
 * widget classes uses callbacks (like "callback") to
 * notify the application-program when something
 * interesting happens. eventhandler has local patterns
 * keypress, keyrelease, buttonpress, exposure, etc., that
 * can be used. If an instance of one of these patterns
 * is enabled, it will be invoked, when the corresponding
 * X-event occurs.
 *)
(#
  eventProcessor:
    (* Prefix for event handling patterns *)
    (#
      EventDesc:< XAnyEvent;
      event:

```



```

        (* The event being processed *)
        @EventDesc;
mask:<
    (* Specify the mask for the X11 event to
       * process
       *)
    (# eventMask, nonmaskable: @integer;
    do INNER
    #);
enable:<
    (* Enables THIS(EventProcessor). After this,
       * THIS(EventProcessor) will be called everytime
       * the corresponding event occurs
       *)
    (# do ...; INNER #);
disable:<
    (* Disable THIS(EventProcessor). Afterwards
       * THIS(EventProcessor) will not be called in
       * response to the corresponding event anymore
       *)
    (# do ...; INNER #);
private: @...;
do INNER
#); (* eventProcessor *)
init:< object;

(* Predefined Eventprocessors *)
keyPress: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XKeyEvent;
    do INNER
    #);
keyRelease: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XKeyEvent;
    do INNER
    #);
buttonPress: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XButtonEvent;
    do INNER
    #);
buttonRelease: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XButtonEvent;
    do INNER
    #);
EnterWindow: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XEnterWindowEvent;
    do INNER
    #);
LeaveWindow: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XLeaveWindowEvent;
    do INNER
    #);
PointerMotion: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XMotionEvent;
    do INNER
    #);
PointerMotionHint: eventProcessor
    (# mask:< (# ... #);
    eventDesc:< XMotionEvent;
    do INNER

```



```

#);
Button1Motion: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XMotionEvent;
  do INNER
  #);
Button2Motion: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XMotionEvent;
  do INNER
  #);
Button3Motion: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XMotionEvent;
  do INNER
  #);
Button4Motion: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XMotionEvent;
  do INNER
  #);
Button5Motion: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XMotionEvent;
  do INNER
  #);
ButtonMotion: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XMotionEvent;
  do INNER
  #);
KeyMapState: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XKeymapEvent;
  do INNER
  #);
Exposure: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XExposeEvent;
  do INNER
  #);
VisibilityChange: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XVisibilityEvent;
  do INNER
  #);
StructureNotify: eventProcessor
  (# mask::<(# ... #);
  do INNER
  #);
ResizeRedirect: eventProcessor
  (# mask::<(# ... #);
    eventDesc::< XResizeRequestEvent;
  do INNER
  #);
SubstructureNotify: eventProcessor
  (# mask::< (# ... #);
  do INNER
  #);
SubstructureRedirect: eventProcessor
  (# mask::<(# ... #);
  do INNER
  #);
FocusChange: eventProcessor
  (# mask::<(# ... #);
  do INNER

```



```

    #);
PropertyChange: eventProcessor
    (# mask::<(# ... #);
      eventDesc::< XPropertyEvent;
    do INNER
    #);
ColormapChange: eventProcessor
    (# mask::<(# ... #);
      eventDesc::< XColormapEvent;
    do INNER
    #);
OwnerGrabButton: eventProcessor
    (# mask::<(# ... #);
    do INNER
    #);
#);
ev: @eventHandler; (* One static instance of eventHandler *)
#) (* core *);

```

**Composite:** Core

```

(* Composite widgets are intended to be containers for other
 * widgets. They have the ability to manage child widgets and
 * they are responsible for handling the geometry for them.
 *
 * The composite widgets pattern is an abstract
 * superpattern. It is never instantiated directly in an
 * application, only subclasses are instantiated.
 *)
(# <<SLOT compositeLib: attributes>>;

init::<
    (# widgetClass::<
        (#
            do INNER;
            (if value=0 then CompositeWidgetClass->value if)
        #)
    do INNER
    #);
NumChildren:
    (* Returns the number of children in this composite
     * widget.
     *)
    IntegerResource(# resourceName::< XtNnumChildren #);
#) (* composite *);

```

**Constraint:** Composite

```

(* The Constraint pattern is a subpattern of the composite,
 * and does thus also manage the layout of children. Constraint
 * widgets let the application provide layout information for
 * each child. This information often takes the form of some
 * constraints on the child's position and/or size. This is a
 * more powerful way to arrange children because it allows you
 * to provide different rules for how each child will be laid
 * out.
 *
 * Constraint is an abstract superpattern. It is never
 * instantiated directly in an application, only subclasses are
 * instantiated.
 *)
(# <<SLOT constraintLib: attributes>>;

init::<
    (# widgetClass::<
        (#
            do INNER;
            (if value=0 then constraintWidgetClass->value if)

```



```

        #)
    do INNER
    #);
#) (* constraint *);

```

**Shell:** Composite

```

(* Widgets negotiate their size and positions with their
 * parent widget (i.e. the widget that directly contain
 * them). Widgets at the top of the hierarchy do not have any
 * parent widget. Instead they must deal with the outside
 * world. To provide for this, each top-level widget is
 * encapsulated in a special widget, called a Shell widget.
 *
 * The Shell pattern is a subpattern of Composite, but can have
 * only one child. Shell widgets are special purpose widgets
 * that provide an interface between other widgets and the
 * window manager. A shell widget negotiates the geometry of
 * the widget with the window manager, and sets the properties
 * required by the window manager.
 *
 * Shell is an abstract superpattern. It is never instantiated
 * directly in an application, only subclasses are
 * instantiated.
 *)
(# <<SLOT shellLib: attributes>>;

init::<
    (# widgetClass::<
        (#
            do INNER;
            (if value=0 then shellWidgetClass -> value if);
            ...;
        #)
        do true -> doNotManageChild; INNER;
    #);

(* Popping up THIS(Shell) *)
popUp:
    (* Call this to popup a shell and its enclosed widgets *)
    (# grabKind: @integer
        (* grabKind controls Xt's event dispatching within
         * the application. It may be one of XtGrabNone - no
         * restrictions on event dispatching
         * XtGrabNonExclusive - all pop ups (i.e. in a menu
         * cascade) will get events XtGrabExclusive - only
         * the most recent pop up will get events
         *)
        enter grabKind
        do (theWidget,grabKind) -> xtPopUp
    #);
popUpSpringLoaded:
    (* Like popUp, except that a passive grab of the pointer
     * is made. This is useful for main menus, whereas popUp
     * is preferable for dialogs.
     *)
    (# do theWidget -> xtPopUpSpringLoaded #);
popDown:
    (* Call this to popdown a shell and its enclosed widgets.
     *)
    (# do theWidget -> xtPopDown #);

(* Shell resources *)
allowShellResize:
    (* This attribute controls whether or not the widget
     * contained by the shell is allowed to try to resize
     * itself.

```



```

*)
BooleanResource(# resourceName::<XtNAllowShellResize#);
geometry:
(* If this attribute is specified, it will override the
 * x, y, width and height of THIS(Shell). The format of
 * this string is
 * [<width>x<height> [{+ -}<xoffset>{+-}<yoffset>].
 * An example is "80x100-300+2" which
 * specifies a window 80 pixels wide and 100 pixels high
 * positioned with the right edge positioned 300 pixels
 * from the right edge of the screen and the top edge 2
 * pixels from the top of the screen.
 *)
StringResource(# resourceName::< XtNGeometry #);
overrideRedirect:
(* This attribute determine whether or not the window
 * manager may interpose itself between THIS(Shell) and
 * the root window of the display.
 *)
BooleanResource(# resourceName::<XtNOverrideRedirect#);
saveUnder:
(* If this is true then save unders will be active for
 * THIS(Shell).
 *)
BooleanResource(# resourceName::< XtNSaveUnder #);

installCallbacks::<
(* Install the popup and popdown callbacks *)
(# do ...; INNER #);

popUpCallback:< object
(* This pattern will be called when THIS(Shell) is popped
 * up.
 *);
popDownCallback:< object
(* This pattern will be called when THIS(Shell) is popped
 * down.
 *);

#) (* Shell *);

OverrideShell: Shell
(* An overrideShell instructs the window manager to completely
 * ignore it. OverrideShells completely bypass the window
 * manager and therefore have no added window manager
 * decorations. They are typically used for popup-menus.
 *)
(# <<SLOT overrideShellLib: attributes>>;

init::<
(# widgetClass::<
  (#
    do INNER;
    (if value=0 then
      OverrideShellWidgetClass->value
    if)
  #)
  do INNER
  #);
#) (* overrideShell *);

WMShell: Shell
(* The WMShell is an abstract super class that contains
 * attributes needed by the common window manager protocol.
 *)
(# <<SLOT wmShellLib: attributes>>;

```



```

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then WMShellWidgetClass->value if)
    #)
  do INNER
  #);

(* WM Shell resources *)
baseHeight:
  (* Size hint to the window manager describing height of
  * fixed components
  *)
  IntegerResource(# resourceName::< XtNBaseHeight #);
baseWidth:
  (* Size hint to the window manager describing width of
  * fixed components
  *)
  IntegerResource(# resourceName::< XtNBaseWidth #);
heightInc:
  (* Size hint describing desired height increment *)
  IntegerResource(# resourceName::< XtNHeightInc #);
widthInc:
  (* Size hint describing desired width increment *)
  IntegerResource(# resourceName::< XtNWidthInc #);
iconMask: (* Mask used with icon pixmap. *)
  IntegerResource(# resourceName::< XtNIconMask #);
iconPixmap: (* Picture for icon. *)
  IntegerResource(# resourceName::< XtNIconPixmap #);
iconWindow: (* Window for icon. *)
  IntegerResource(# resourceName::< XtNIconWindow #);
iconX: (* Icon x position *)
  IntegerResource(# resourceName::< XtNIconX #);
iconY: (* Icon y position *)
  IntegerResource(# resourceName::< XtNIconY #);
input:
  (* If this attribute is true the window manager will set
  * the keyboard focus to this application or not,
  * according to its pointer-following or click-to-type
  * model of keyboard input. If it is false, the window
  * manager will not set keyboard focus to this
  * application.
  *)
  BooleanResource(# resourceName::< XtNInput #);
maxAspectX:
  (* Size hint describing maximum horizontal aspect ratio *)
  IntegerResource(# resourceName::< XtNMaxAspectX #);
maxAspectY:
  (* Size hint describing maximum vertical aspect ratio *)
  IntegerResource(# resourceName::< XtNMaxAspectY #);
maxHeight:
  (* Size hint describing maximum acceptable height *)
  IntegerResource(# resourceName::< XtNMaxHeight #);
maxWidth:
  (* Size hint describing maximum acceptable width *)
  IntegerResource(# resourceName::< XtNMaxWidth #);
minAspectX:
  (* Size hint describing minimum horizontal aspect ratio *)
  IntegerResource(# resourceName::< XtNminAspectX #);
minAspectY:
  (* Size hint describing minimum vertical aspect ratio *)
  IntegerResource(# resourceName::< XtNminAspectY #);
minHeight:
  (* Size hint describing minimum acceptable height *)

```



```

IntegerResource(# resourceName::< XtNminHeight #);
minWidth:
(* Size hint describing minimum acceptable width *)
IntegerResource(# resourceName::< XtNminWidth #);
title: (* The text to be used in the title bar *)
StringResource(# resourceName::< XtNTitle #);
transient:
(* True if THIS(WMShell) is transient. Is true for a
 * transientShell and false otherwise
 *)
BooleanResource(# resourceName::< XtNtransient #);
waitForWM:
(* WaitForWM and wmTimeout controls the response to
 * delays by the window manager in response to geometry
 * change requests. By default waitForWm is true and
 * wmTimeout is 5 seconds. If a response for a geometry
 * request does not arrive within 5 seconds, Xt assumes
 * that the window manager is not functioning.
 *)
BooleanResource(# resourceName::< XtNwaitForWm #);
wmTimeout: (* See waitForWM *)
IntegerResource(# resourceName::< XtNwmTimeout #);
winGravity:
(* This attribute controls how to reposition the widget
 * contained by the shell when the window changes. The
 * value can be NorthWestGravity, NorthGravity,...,
 * SouthEastGravity (compas directions), CenterGravity,
 * and UnmapGravity.
 *)
IntegerResource(# resourceName::< XtNwinGravity #);
windowGroup:
(* This attribute links popup windows to the main window
 *)
IntegerResource(# resourceName::< XtNWindowGroup #);
#) (* WMShell *);

```

**VendorShell:** WMShell

```

(* The VendorShell is an abstract super class with
 * implementation dependent resources defined by the different
 * widget sets using it.
 *)
(# <<SLOT VendorShellLib: attributes>>;

```

```

init::<
  (# widgetClass::<
    (#
      do INNER;
        (if value=0 then
          VendorShellWidgetClass->value
        if)
      #)
    do INNER
  #);
#);

```

**TransientShell:** VendorShell

```

(* A transient shell is similar to a toplevel shell except for
 * the way it interacts with the window manager with respect to
 * iconifying. If an applications toplevel window
 * (transientFor) is iconified, the window manager normally
 * iconifies all transient shells created by the application.
 *)
(# <<SLOT transientShellLib: attributes>>;

```

```

init::<
  (# widgetClass::<

```



```

        (#
        do INNER;
        (if value=0 then
            TransientShellWidgetClass->value
        if)
        #)
    do INNER
    #);

transientFor:
    (* The widget, that THIS(TransientShell) is transient
    * for. Default value is topLevel.
    *)
    IntegerResource(# resourceName:<: XtnTransientFor #);
#) (* transientShell *);

ToplevelShell: VendorShell
    (* ToplevelShell widgets are used for normal top-level
    * windows. XtEnv creates the main toplevel widget, "topLevel",
    * of the application. Some applications have multiple
    * permanent top-level windows, and should use ToplevelShell to
    * accomplish this.
    *)
    (# <<SLOT toplevelShellLib: attributes>>;

    init::<
        (# widgetClass:<:
        (#
        do INNER;
        (if value=0 then
            ToplevelShellWidgetClass->value
        if)
        #)
        do INNER
        #);

    iconName: (* The text to be used for the icon *)
        StringResource(# resourceName:<: XtnIconname #);
    iconic:
        (* This attribute may be used by an application to
        * request that the window manager iconify or deiconify
        * THIS(ToplevelShell). If set before realization it is
        * also a method to change initialState
        *)
        BooleanResource(# resourceName:<: XtnIconic #);
    #) (* toplevelShell *);

Display:
    (* This pattern allows an application to have windows open on
    * more than one workstation at the same time. After
    * THIS(Display) have been opened, THIS(Display).shell can be
    * used as a toplevel widget on that display. When children
    * have been added to THIS(Display).shell, it must be realized
    * to show the window on screen.
    *)
    (# displayString: ^text
        (* A text that contains the enter-parameter to open (the
        * name of the display
        *)
        #);
    shell: @ToplevelShell
        (* After THIS(display) have been opened this specifies
        * the toplevel window on the display.
        *)
    open:
        (* Opens the display. Takes as enter parameter the name
        * of the display to open. An example of a displayname is

```



```

    * "www.daimi.au.dk:0.0" which specifies screen
    * number 0 on cpu number 0 on the host
    * www.daimi.au.dk.
    *)
    (# DisplayError:< XtException
      (* Called if opening of the display fails *)
      (# ... #);
    enter displayString[]
    do ...;
    #);
  close:
    (* Close the display connection, and properly dispose all
      * resources is has created.
      *)
    (# ... #);
  private: @...;
#);

```

**Timer:**

```

(* This pattern allow the application to be notified when a
 * period of time have elapsed, while being able to do other
 * things during that time.
 *)
(# on: @boolean
  (* A boolean that specifies if the timer is on for the
    * time being
    *)
  start:
    (* Starts THIS(Timer). Takes as enter parameter the
      * number of milliseconds the timer should run before
      * invoking timeout.
      *)
    (# ms: @integer
      enter ms
      ...
      #);
  disable: (* Disables THIS(Timer) *)
    (# ... #);
  timeout:<
    (* This method is called when the specified number of
      * milliseconds have elapsed if it have not been disabled
      * in the meantime. It is called only once, so to have
      * THIS(Timer) timing out repeatedly, it must be
      * re-started after each timeout.
      *)
    (# ... #);
  private: @...;
#);

```

**WorkProc:**

```

(* The WorkProc-pattern is a means for doing background
 * processing while the application is idle waiting for an
 * event. After a WorkProc have been started, the virtual
 * method "job" is invoked by XtEnv when no events are
 * pending. It is invoked through a BETA component, such that a
 * further binding of job may suspend itself in order not to
 * let the user's response time suffer from time consuming
 * background processing. Next time there are no events
 * pending, job is resumed at the suspended place. More than
 * one Workproc Can be started at a time.
 *)
(# start: (* Starts THIS(WorkProc) *)
  (# ... #);
  disable: (* Disables THIS(WorkProc) *)
    (# ... #);
  job:<

```



```

    (* Called by XtEnv when no events are pending. The BETA
    * suspend- imperative can be used to let XtEnv
    * temporarily come into the foreground again.
    *)
    (# finished: @boolean
    do false -> finished;
      INNER;
      true -> finished;
      suspend;
    exit finished
    #);
  private: @...;
#);

```

**AcceleratorTable:** IntegerObject

```

(* Can be used for the advanced programmer to change the
* translations used by Xt. The value exited can be used to set
* the "accelerators" resource of a widget.
*)
(# parse:
  (* Parses up an accelerator specification and converts it
  * to the interior format of Xt.
  *)
  (# spec: ^text
  enter spec[]
  do spec -> XtParseAcceleratorTable -> value
  exit value
  #);
#);

```

**TranslationTable:** IntegerObject

```

(* Can be used for the advanced programmer to change the
* translations used by Xt. The value exited can be used as
* input to Core.overrideTranslations etc.
*)
(# parse:
  (* Parses up a translationtable specification and
  * converts it to the interior format of Xt.
  *)
  (# spec: ^text
  enter spec[]
  do spec -> XtParseTranslationTable -> value
  exit value
  #);
#);

```

**RegisteredAction:**

```

(* Used to register an X Toolkit action: After
* THIS(RegisteredAction) has been installed, it can be used as
* an Xt action, e.g. be used in translations. When the action
* is invoked, INNER is called. theObj is a reference to the
* widget or gadget, the action was imposed on, event is the
* event that caused the action to be invoked.
*)
(# theObj: ^XtObject;
  eventDesc: < XAnyEvent;
  event: @eventDesc;
  params, numParams: @integer;
  private: @...;

  install:
    (* Install THIS(RegisteredAction). If name is specified,
    * the action will be installed with that name, otherwise
    * the pattern name of THIS(RegisteredAction) is used.
    *)
    (# name: ^text;

```



```

        enter name[]
        do ...;
        #);
do INNER
#);

StringArray:
(* A StringArray is a list of externally allocated text's. *)
(# initialSize:< IntegerObject(# do 25 -> value; INNER #);
init:<
    (* Initializes THIS(StringArray) *)
    (# do ...; INNER #);
clear:< (* Reset THIS(StringArray) *)
    (# do ...; INNER #);
number:
    (* Number of texts currently in THIS(StringArray) *)
    IntegerValue(# ... #);
getText:
    (* Exits a copy of the text of THIS(StringArray) added as
    * number n
    *)
    (# n: @integer;
        t: ^text;
        enter n
        do ...;
        exit t[]
        #);
changeText:
    (* Changes the text of THIS(StringArray) added as number
    * n
    *)
    (# n: @integer;
        newtext: ^text;
        enter (n, newtext[])
        do ...;
        #);
addText:
    (* Adds a new text to
    * THIS(StringArray). THIS(StringArray) is extended if
    * needed.
    *)
    (# t: ^text;
        enter t[]
        do ...;
        #);
<<SLOT StringArrayLib: attributes>>;
private: @...;
address:
    (* Address of externally allocated string array *)
    IntegerValue(# ... #);
exit address
#);

private: @...;

(* Aliases to use in Composite *)
bcore: core(##);

(* sbrandt: Additions needed be XsystemEnv: *)
doSetup:
    (#
        do (if not setupDone then
            ...;
            true -> setupDone
        if)
    #);

```



```

setupDone: @Boolean;
XsystemEnvPresent: @Boolean;
(* TRUE if this is a XsystemEnv program. In this case,
 * callbacks are executed by a separate thread as synchronisation
 * via semaphores between x-callbacks and other coroutines would
 * not be possible otherwise. (It could lead to suspend of
 * coroutines with C stackparts. If TRUE,
 * XsystemEnvHandleCallback should not be NONE.
 *)
XsystemEnvHandleCallbackP
  (# cb: ^Object; enter cb[] do INNER #);
XsystemEnvHandleCallback
  ^XsystemEnvHandleCallbackP;

do doSetup;
  INNER; ...
#) (* xtenv *)

--- compositeLib: attributes ---

(* Redefinition of patterns within Composites: Use the Composite as
 * default father
 *)
core: bcore
  (# init::< (# getFatherWidget::< (# do this(composite)->value #) do INNER #)#)

```

---

Xtenv Interface

[Mjølner Informatics](#)



# Events Interface

```

ORIGIN '~beta/basiclib/external';
LIB_DEF 'Xtevents' '../lib';
( *
  * COPYRIGHT
  *       Copyright Mjolner Informatics, 1992-97
  *       All rights reserved.
  *)

-- LIB: attributes --

XAnyEvent: ExternalRecord
  ( * Unspecified X event *)
  ( # type: @
    ( * Type of THIS(XAnyEvent) *)
    long( # pos::< ( # do 0->value #) # );
    serial: @
    ( * number of last request processed by server *)
    long( # pos::< ( # do 4->value #) # );
    send_event: @
    ( * true if this came from a SendEvent request *)
    long( # pos::< ( # do 8->value #) # );
    display: @
    ( * Display the event was read from *)
    long( # pos::< ( # do 12->value #) # );
    window: @
    ( * window on which event was requested in event mask *)
    long( # pos::< ( # do 16->value #) # );

    init:
    ( * Used to allocate an XAnyEvent from within BETA *)
    ( # do 96 -> malloc -> ptr # );

    <<SLOT XAnyEventLib: attributes>>
  enter ptr
  exit ptr
  # );

( * * Definitions of specific events. *)
XKeyEvent: XAnyEvent
  ( # root: @
    ( * root window that the event occurred on *)
    long( # pos::< ( # do 20->value #) # );
    subwindow: @
    ( * child window *)
    long( # pos::< ( # do 24->value #) # );
    time: @
    ( * milliseconds *)
    long( # pos::< ( # do 28->value #) # );
    x: @
    ( * pointer x coordinate in event window *)
    long( # pos::< ( # do 32->value #) # );
    y: @
    ( * pointer y coordinate in event window *)
    long( # pos::< ( # do 36->value #) # );
    x_root: @
    ( * x coordinate relative to root *)
    long( # pos::< ( # do 40->value #) # );
    y_root: @
    ( * y coordinate relative to root *)
    long( # pos::< ( # do 44->value #) # );
    state: @
    ( * key or button mask *)
```



```

    long(# pos::<(# do 48->value#)#);
keycode: @
    (* detail *)
    long(# pos::<(# do 52->value#)#);
same_screen: @
    (* same screen flag *)
    long(# pos::<(# do 56->value#)#);

shiftModified:
    (# exit (((state,shiftMask) -> tos'%and')=shiftMask) #);
controlModified:
    (# exit (((state,controlMask)-> tos'%and')=controlMask)#);
lockModified:
    (# exit (((state,lockMask) -> tos'%and')=lockMask) #);
mod1Modified:
    (# exit (((state,mod1Mask) -> tos'%and')=mod1Mask) #);
mod2Modified:
    (# exit (((state,mod2Mask) -> tos'%and')=mod2Mask) #);
mod3Modified:
    (# exit (((state,mod3Mask) -> tos'%and')=mod3Mask) #);
mod4Modified:
    (# exit (((state,mod4Mask) -> tos'%and')=mod4Mask) #);
mod5Modified:
    (# exit (((state,mod5Mask) -> tos'%and')=mod5Mask) #);

<<SLot XKeyEventLib: attributes>>
#);

XKeyPressedEvent: XKeyEvent(# #);
XKeyReleasedEvent: XKeyEvent(# #);

XButtonEvent: XAnyEvent
(# root: @
    (* root window that the event occurred on *)
    long(# pos::<(# do 20->value#)#);
subwindow: @
    (* child window *)
    long(# pos::<(# do 24->value#)#);
time: @
    (* milliseconds *)
    long(# pos::<(# do 28->value#)#);
x: @
    (* pointer x coordinate in event window *)
    long(# pos::<(# do 32->value#)#);
y: @
    (* pointer y coordinate in event window *)
    long(# pos::<(# do 36->value#)#);
x_root: @
    (* x coordinate relative to root *)
    long(# pos::<(# do 40->value#)#);
y_root: @
    (* y coordinate relative to root *)
    long(# pos::<(# do 44->value#)#);
state: @
    (* key or button mask *)
    long(# pos::<(# do 48->value#)#);
button: @
    (* detail *)
    long(# pos::<(# do 52->value#)#);
same_screen: @
    (* same screen flag *)
    long(# pos::<(# do 56->value#)#);

shiftModified:
    (# exit (((state,shiftMask) -> tos'%and')=shiftMask) #);
controlModified:

```



```

    (# exit (((state,controlMask)-> tos'%and')=controlMask)#);
lockModified:
    (# exit (((state,lockMask) -> tos'%and')=lockMask) #);
mod1Modified:
    (# exit (((state,mod1Mask) -> tos'%and')=mod1Mask) #);
mod2Modified:
    (# exit (((state,mod2Mask) -> tos'%and')=mod2Mask) #);
mod3Modified:
    (# exit (((state,mod3Mask) -> tos'%and')=mod3Mask) #);
mod4Modified:
    (# exit (((state,mod4Mask) -> tos'%and')=mod4Mask) #);
mod5Modified:
    (# exit (((state,mod5Mask) -> tos'%and')=mod5Mask) #);

    <<SLOT XButtonEventLib: attributes>>
#);

XButtonPressedEvent: XButtonEvent(# #);
XButtonReleasedEvent: XButtonEvent(# #);

XMotionEvent: XAnyEvent
(# root: @
    (* root window that the event occurred on *)
    long(# pos::<(# do 20->value#)#);
subwindow: @
    (* child window *)
    long(# pos::<(# do 24->value#)#);
time: @
    (* milliseconds *)
    long(# pos::<(# do 28->value#)#);
x: @
    (* pointer x coordinate in event window *)
    long(# pos::<(# do 32->value#)#);
y: @
    (* pointer y coordinate in event window *)
    long(# pos::<(# do 36->value#)#);
x_root: @
    (* x coordinate relative to root *)
    long(# pos::<(# do 40->value#)#);
y_root: @
    (* y coordinate relative to root *)
    long(# pos::<(# do 44->value#)#);
state: @
    (* key or button mask *)
    long(# pos::<(# do 48->value#)#);
is_hint: @
    (* detail *)
    byte(# pos::<(# do 52->value#)#);
same_screen: @
    (* same screen flag *)
    long(# pos::<(# do 56->value#)#);

shiftModified:
    (# exit (((state,shiftMask) -> tos'%and')=shiftMask) #);
controlModified:
    (# exit (((state,controlMask)-> tos'%and')=controlMask)#);
lockModified:
    (# exit (((state,lockMask) -> tos'%and')=lockMask) #);
mod1Modified:
    (# exit (((state,mod1Mask) -> tos'%and')=mod1Mask) #);
mod2Modified:
    (# exit (((state,mod2Mask) -> tos'%and')=mod2Mask) #);
mod3Modified:
    (# exit (((state,mod3Mask) -> tos'%and')=mod3Mask) #);
mod4Modified:
    (# exit (((state,mod4Mask) -> tos'%and')=mod4Mask) #);

```



```

mod5Modified:
    (# exit (((state,mod5Mask) -> tos'%and')=mod5Mask) #);

<<SLot XMotionEventLib: attributes>>
#);

XPointerMovedEvent: XMotionEvent(# #);

XCrossingEvent: XAnyEvent
(# root: @
    (* root window that the event occurred on *)
    long(# pos::<(# do 20->value#)#);
subwindow: @
    (* child window *)
    long(# pos::<(# do 24->value#)#);
time: @
    (* milliseconds *)
    long(# pos::<(# do 28->value#)#);
x: @
    (* pointer x coordinate in event window *)
    long(# pos::<(# do 32->value#)#);
y: @
    (* pointer y coordinate in event window *)
    long(# pos::<(# do 36->value#)#);
x_root: @
    (* x coordinate relative to root *)
    long(# pos::<(# do 40->value#)#);
y_root: @
    (* y coordinate relative to root *)
    long(# pos::<(# do 44->value#)#);
Mode: @
    (* NotifyNormal, NotifyGrab, NotifyUngrab *)
    long(# pos::<(# do 48->value#)#);
detail: @
    (* NotifyAncestor, NotifyVirtual, NotifyInferior,
       * NotifyNonLinear,NotifyNonLinearVirtual
       *)
    long(# pos::<(# do 52->value#)#);
same_screen: @
    (* same screen flag *)
    long(# pos::<(# do 56->value#)#);
focus: @
    (* boolean focus *)
    long(# pos::<(# do 60->value#)#);
state: @
    (* key or button mask *)
    long(# pos::<(# do 64->value#)#);

shiftModified:
    (# exit (((state,shiftMask) -> tos'%and')=shiftMask) #);
controlModified:
    (# exit (((state,controlMask)-> tos'%and')=controlMask)#);
lockModified:
    (# exit (((state,lockMask) -> tos'%and')=lockMask) #);
mod1Modified:
    (# exit (((state,mod1Mask) -> tos'%and')=mod1Mask) #);
mod2Modified:
    (# exit (((state,mod2Mask) -> tos'%and')=mod2Mask) #);
mod3Modified:
    (# exit (((state,mod3Mask) -> tos'%and')=mod3Mask) #);
mod4Modified:
    (# exit (((state,mod4Mask) -> tos'%and')=mod4Mask) #);
mod5Modified:
    (# exit (((state,mod5Mask) -> tos'%and')=mod5Mask) #);

<<SLot XCrossingEventLib: attributes>>

```



```

#);

XEnterWindowEvent: XCrossingEvent(# #);
XLeaveWindowEvent: XCrossingEvent(# #);

XFocusChangeEvent: XAnyEvent
(# Mode: @
  (* NotifyNormal, NotifyGrab, NotifyUngrab *)
  long(# pos::<(# do 20->value#)#);
  detail: @
  (* NotifyAncestor, NotifyVirtual, NotifyInferior,
    * NotifyNonLinear,NotifyNonLinearVirtual, NotifyPointer,
    * NotifyPointerRoot, NotifyDetailNone
    *)
  long(# pos::<(# do 24->value#)#);

  <<SLOT XFocusChangeEventLib: attributes>>
#);

XFocusInEvent: XFocusChangeEvent(# #);
XFocusOutEvent: XFocusChangeEvent(# #);

(* Generated on EnterWindow and FocusIn when KeyMapState selected *)
XKeymapEvent: XAnyEvent
(# key_vector: @long(# pos::<(# do 0->value#)#);
  (* Use getbyte/putbyte with index 0-31 *)
  <<SLOT XKeymapEventLib: attributes>>
#);

XExposeEvent: XAnyEvent
(# x: @
  (* Rectangle origin x *)
  long(# pos::<(# do 20->value#)#);
  y: @
  (* Rectangle origin y *)
  long(# pos::<(# do 24->value#)#);
  width:
  (* Rectangle width *)
  @long(# pos::<(# do 28->value#)#);
  height: @
  (* Rectangle height *)
  long(# pos::<(# do 32->value#)#);
  count: @
  (* if non-zero, at least this many more *)
  long(# pos::<(# do 36->value#)#);

  <<SLOT XExposeEventLib: attributes>>
#);

XGraphicsExposeEvent: XAnyEvent
(# drawable: @
  (* Exposed drawable *)
  long(# pos::<(# do 16->value#)#);
  x: @
  (* Rectangle origin x *)
  long(# pos::<(# do 20->value#)#);
  y: @
  (* Rectangle origin y *)
  long(# pos::<(# do 24->value#)#);
  width:
  (* Rectangle width *)
  @long(# pos::<(# do 28->value#)#);
  height: @
  (* Rectangle height *)
  long(# pos::<(# do 32->value#)#);
  count: @

```



```

        (* if non-zero, at least this many more *)
        long(# pos::<(# do 36->value#)#);
major_code: @
        (* core is CopyArea or CopyPlane *)
        long(# pos::<(# do 40->value#)#);
minor_code: @
        (* not defined in the core *)
        long(# pos::<(# do 44->value#)#);

<<SLOT XGraphicsExposeEventLib: attributes>>
#);

XNoExposeEvent: XAnyEvent
(# drawable: @
    (* Drawable *)
    long(# pos::<(# do 16->value#)#);
major_code: @
    (* core is CopyArea or CopyPlane *)
    long(# pos::<(# do 20->value#)#);
minor_code: @
    (* not defined in the core *)
    long(# pos::<(# do 24->value#)#);

<<SLOT XNoExposeEventLib: attributes>>
#);

XVisibilityEvent: XAnyEvent
(# state: @
    (* Visibility State *)
    long(# pos::<(# do 20->value#)#);

<<SLOT XVisibilityEventLib: attributes>>
#);

XCreateWindowEvent: XAnyEvent
(# parent: @
    (* parent of the window *)
    long(# pos::<(# do 16->value#)#);
window: @
    (* window id of window created *)
    long(# pos::<(# do 20->value#)#);
x: @
    (* New x position *)
    long(# pos::<(# do 24->value#)#);
y: @
    (* New y position *)
    long(# pos::<(# do 28->value#)#);
width:
    (* New window width *)
    @long(# pos::<(# do 32->value#)#);
height: @
    (* New window height *)
    long(# pos::<(# do 36->value#)#);
border_width: @
    (* New border width *)
    long(# pos::<(# do 40->value#)#);
override_redirect: @
    (* creation should be overridden by WM *)
    long(# pos::<(# do 44->value#)#);

<<SLOT XCreateWindowEventLib: attributes>>
#);

XDestroyWindowEvent: XAnyEvent
(# event: @
    (* Event window *)

```



```

        long(# pos::<(# do 16->value#)#);
window: @
        (* Destroyed window *)
        long(# pos::<(# do 20->value#)#);

    <<SLOT XDestroyWindowEventLib: attributes>>
#);

XUnmapEvent: XAnyEvent
(# event: @
    (* Event window *)
    long(# pos::<(# do 16->value#)#);
window: @
    (* Unmapped window *)
    long(# pos::<(# do 20->value#)#);
from_configure: @
    (* If gravity processing *)
    long(# pos::<(# do 24->value#)#);

    <<SLOT XUnmapEventLib: attributes>>
#);

XMapEvent: XAnyEvent
(# event: @
    (* Event window *)
    long(# pos::<(# do 16->value#)#);
window: @
    (* Mapped window *)
    long(# pos::<(# do 20->value#)#);
override_redirect: @
    (* Prevent WM intervention *)
    long(# pos::<(# do 24->value#)#);

    <<SLOT XMapEventLib: attributes>>
#);

XMapRequestEvent: XAnyEvent
(# parent: @
    (* Parent window *)
    long(# pos::<(# do 16->value#)#);
window: @
    (* Window to map *)
    long(# pos::<(# do 20->value#)#);

    <<SLOT XMapRequestEventLib: attributes>>
#);

XReparentEvent: XAnyEvent
(# event: @
    (* Event window *)
    long(# pos::<(# do 16->value#)#);
window: @
    (* Reparented window *)
    long(# pos::<(# do 20->value#)#);
parent: @
    (* New parent window *)
    long(# pos::<(# do 24->value#)#);
x: @
    (* New x position *)
    long(# pos::<(# do 28->value#)#);
y: @
    (* New y position *)
    long(# pos::<(# do 32->value#)#);
override_redirect: @
    (* Prevent WM intervention *)
    long(# pos::<(# do 36->value#)#);

```



```
<<SLOT XReparentEventLib: attributes>>
#);
```

**XConfigureEvent:** XAnyEvent

```
(# event: @
  (* Event window *)
  long(# pos::<(# do 16->value#)#);
window: @
  (* Reconfigured window *)
  long(# pos::<(# do 20->value#)#);
x: @
  (* New window x position *)
  long(# pos::<(# do 24->value#)#);
y: @
  (* New window y position *)
  long(# pos::<(# do 28->value#)#);
width: @
  (* New window width *)
  long(# pos::<(# do 32->value#)#);
height: @
  (* New window height *)
  long(# pos::<(# do 36->value#)#);
border_width: @
  (* New border width *)
  long(# pos::<(# do 40->value#)#);
above: @
  (* Sibling ot None *)
  long(# pos::<(# do 44->value#)#);
override_redirect: @
  (* Prevent WM intervention *)
  long(# pos::<(# do 48->value#)#);

<<SLOT XConfigureEventLib: attributes>>
#);
```

**XGravityEvent:** XAnyEvent

```
(#
  event: @
    (* Event window *)
    long(# pos::<(# do 16->value#)#);
  window: @
    (* Moved window *)
    long(# pos::<(# do 20->value#)#);
  x: @
    (* New x position *)
    long(# pos::<(# do 24->value#)#);
  y: @
    (* New y position *)
    long(# pos::<(# do 28->value#)#);

<<SLOT XGravityEventLib: attributes>>
#);
```

**XResizeRequestEvent:** XAnyEvent

```
(# width: @
  (* New width *)
  long(# pos::<(# do 20->value#)#);
height: @
  (* New height *)
  long(# pos::<(# do 24->value#)#);

<<SLOT XResizeRequestEventLib: attributes>>
#);
```

**XConfigureRequestEvent:** XAnyEvent



```

(# parent: @
  (* Parent window *)
  long(# pos::<(# do 16->value#)#);
window: @
  (* Window to configure *)
  long(# pos::<(# do 20->value#)#);
x: @
  (* Requested x position *)
  long(# pos::<(# do 24->value#)#);
y: @
  (* Requested y position *)
  long(# pos::<(# do 28->value#)#);
width: @
  (* Requested width *)
  long(# pos::<(# do 32->value#)#);
height: @
  (* Requested height *)
  long(# pos::<(# do 36->value#)#);
border_width: @
  (* Requested border width *)
  long(# pos::<(# do 40->value#)#);
above: @
  (* Sibling or None *)
  long(# pos::<(# do 44->value#)#);
detail: @
  (* Above, Below, TopIf, BottomIf, Opposite *)
  long(# pos::<(# do 48->value#)#);
value_mask: @
  (* *)
  long(# pos::<(# do 52->value#)#);

<<SLOT XConfigureRequestEventLib: attributes>>
#);

XCirculateEvent: XAnyEvent
(# event: @
  (* Event window *)
  long(# pos::<(# do 16->value#)#);
window: @
  (* Circulated window *)
  long(# pos::<(# do 20->value#)#);
Place: @
  (* PlaceOnTop, PlaceOnBottom *)
  long(# pos::<(# do 24->value#)#);

<<SLOT XCirculateEventLib: attributes>>
#);

XCirculateRequestEvent: XAnyEvent
(# parent: @
  (* Parent window *)
  long(# pos::<(# do 16->value#)#);
window: @
  (* Window to circulate *)
  long(# pos::<(# do 20->value#)#);
Place: @
  (* PlaceOnTop, PlaceOnBottom *)
  long(# pos::<(# do 24->value#)#);

<<SLOT XCirculateRequestEventLib: attributes>>
#);

XPropertyEvent: XAnyEvent
(# atom: @
  (* Atom for property name *)
  long(# pos::<(# do 20->value#)#);

```



```

time: @
    (* Time when the property changed *)
    long(# pos::<(# do 24->value#)#);
state: @
    (* NewValue, Deleted *)
    long(# pos::<(# do 28->value#)#);

<<SLLOT XPropertyEventLib: attributes>>
#);

XSelectionClearEvent: XAnyEvent
(# selection: @
    (* Selection name's atom *)
    long(# pos::<(# do 20->value#)#);
time: @
    (* Selection loss timestamp *)
    long(# pos::<(# do 24->value#)#);

<<SLLOT XSelectionClearEventLib: attributes>>
#);

XSelectionRequestEvent: XAnyEvent
(# owner: @
    (* Owning window *)
    long(# pos::<(# do 16->value#)#);
requestor: @
    (* Requesting window *)
    long(# pos::<(# do 20->value#)#);
selection: @
    (* Selection *)
    long(# pos::<(# do 24->value#)#);
target: @
    (* Data type wanted *)
    long(# pos::<(# do 28->value#)#);
property: @
    (* Property to use *)
    long(# pos::<(# do 32->value#)#);
time: @
    (* Timestamp *)
    long(# pos::<(# do 36->value#)#);

<<SLLOT XSelectionRequestEventLib: attributes>>
#);

XSelectionEvent: XAnyEvent
(# requestor: @
    (* Requesting window *)
    long(# pos::<(# do 16->value#)#);
selection: @
    (* Selection *)
    long(# pos::<(# do 20->value#)#);
target: @
    (* Data type used *)
    long(# pos::<(# do 24->value#)#);
property: @
    (* Property used *)
    long(# pos::<(# do 28->value#)#);
time: @
    (* Timestamp *)
    long(# pos::<(# do 32->value#)#);

<<SLLOT XSelectionEventLib: attributes>>
#);

XColormapEvent: XAnyEvent
(# colormap: @

```



```

        (* Color map identifier *)
        long(# pos::<(# do 20->value#)#);
new: @
        (* Color map changed? *)
        long(# pos::<(# do 24->value#)#);
state: @
        (* ColormapInstalled, ColormapUninstalled *)
        long(# pos::<(# do 28->value#)#);

        <<SLOT XColormapEventLib: attributes>>
#);

XClientMessageEvent: XAnyEvent
(# message_type: @
    (* X atom for message type *)
    long(# pos::<(# do 20->value#)#);
    format: @
    (* 8, 16, or 32 *)
    long(# pos::<(# do 24->value#)#);
    data: @
    (* Pointer to either 20 bytes (format=8), 10 shorts
     * (format=16), or 5 longs (format=32). Use getByte/putByte,
     * getShort/putShort, or getLong/putLong, respectively.
     *)
    long(# pos::<(# do 28->value#)#);

    <<SLOT XClientMessageEventLib: attributes>>
#);

XMappingEvent: XAnyEvent
(# request: @
    (* One Of MappingModifier, MappingKeyboard, MappingPointer *)
    long(# pos::<(# do 20->value#)#);
    first_keycode: @
    (* first keycode *)
    long(# pos::<(# do 24->value#)#);
    count: @
    (* defines range of change w. first_keycode *)
    long(# pos::<(# do 28->value#)#);

    <<SLOT XMappingEventLib: attributes>>
#);

XErrorEvent: XAnyEvent
(# type: @
    (* type *)
    long(# pos::<(# do 0->value#)#);
    display: @
    (* Display The Event Was Read From *)
    long(# pos::<(# do 4->value#)#);
    resourceid: @
    (* resource id *)
    long(# pos::<(# do 8->value#)#);
    serial: @
    (* serial number of failed request *)
    long(# pos::<(# do 12->value#)#);
    error_code: @
    (* error code of failed request *)
    byte(# pos::<(# do 16->value#)#);
    request_code: @
    (* Major Op-Code Of Failed Request *)
    byte(# pos::<(# do 17->value#)#);
    minor_code: @
    (* Minor Op-Code Of Failed Request *)
    byte(# pos::<(# do 18->value#)#);

```



```

    <<SLOT XErrorEventLib: attributes>>
    #);

(* Event masks, used in soliciting events *)
XNoEventMask: (# exit 0 #);
XKeyPressMask: (# exit 1 #);
XKeyReleaseMask: (# exit 2 #);
XButtonPressMask: (# exit 4 #);
XButtonReleaseMask: (# exit 8 #);
XEnterWindowMask: (# exit 16 #);
XLeaveWindowMask: (# exit 32 #);
XPointerMotionMask: (# exit 64 #);
XPointerMotionHintMask: (# exit 128 #);
XButton1MotionMask: (# exit 256 #);
XButton2MotionMask: (# exit 512 #);
XButton3MotionMask: (# exit 1024 #);
XButton4MotionMask: (# exit 2048 #);
XButton5MotionMask: (# exit 4096 #);
XButtonMotionMask: (# exit 8192 #);
XKeyMapStateMask: (# exit 16384 #);
XExposureMask: (# exit 32768 #);
XVisibilityChangeMask: (# exit 65536 #);
XStructureNotifyMask: (# exit 131072 #);
XResizeRedirectMask: (# exit 262144 #);
XSubstructureNotifyMask: (# exit 524288 #);
XSubstructureRedirectMask: (# exit 1048576 #);
XFocusChangeMask: (# exit 2097152 #);
XPropertyChangeMask: (# exit 4194304 #);
XColormapChangeMask: (# exit 8388608 #);
XOwnerGrabButtonMask: (# exit 16777216 #);

(* Event names. Used in "type" field in X events *)
XKeyPress: (# exit 2 #);
XKeyRelease: (# exit 3 #);
XButtonPress: (# exit 4 #);
XButtonRelease: (# exit 5 #);
XMotionNotify: (# exit 6 #);
XEnterNotify: (# exit 7 #);
XLeaveNotify: (# exit 8 #);
XFocusIn: (# exit 9 #);
XFocusOut: (# exit 10 #);
XKeymapNotify: (# exit 11 #);
XExpose: (# exit 12 #);
XGraphicsExpose: (# exit 13 #);
XNoExpose: (# exit 14 #);
XVisibleNotify: (# exit 15 #);
XCreateNotify: (# exit 16 #);
XDestroyNotify: (# exit 17 #);
XUnmapNotify: (# exit 18 #);
XMapNotify: (# exit 19 #);
XMapRequest: (# exit 20 #);
XReparentNotify: (# exit 21 #);
XConfigureNotify: (# exit 22 #);
XConfigureRequest: (# exit 23 #);
XGravityNotify: (# exit 24 #);
XResizeRequest: (# exit 25 #);
XCirculateNotify: (# exit 26 #);
XCirculateRequest: (# exit 27 #);
XPropertyNotify: (# exit 28 #);
XSelectionClear: (# exit 29 #);
XSelectionRequest: (# exit 30 #);
XSelectionNotify: (# exit 31 #);
XColorMapNotify: (# exit 32 #);
XClientMessage: (# exit 33 #);
XMappingNotify: (# exit 34 #);
XLastEvent: (# exit 35 #);

```



```

(* modifier masks *)
mod1Mask: (# exit 8 #);
mod2Mask: (# exit 16 #);
mod3Mask: (# exit 32 #);
mod4Mask: (# exit 64 #);
mod5Mask: (# exit 128 #);

ShiftMask: (# exit 1 #);
LockMask: (# exit 2 #);
ControlMask: (# exit 4 #);

(***** (De)Allocation in C heap *****)

malloc: External
  (# size: @integer;
   ptr: @integer;
   enter size
   exit ptr
   #);
free: External
  (# ptr: @integer;
   enter ptr
   #)

```

---

Events Interface

[Mjølner Informatics](#)



# Xsystemenv Interface

```
ORIGIN '~beta/basiclib/basicssystemenv';
BODY 'private/xsystemenvbody';

(*
 * COPYRIGHT
 *   Copyright Mjolner Informatics, 1992-97
 *   All rights reserved.
 *)

(* XSYSTEMENV
 * =====
 *
 * Use this fragment as the ORIGIN for concurrent programs
 * using the X libraries.
 *
 * The program should look something like:
 *
 * ORIGIN 'xsystemenv';
 * [[
 *   --- program:descriptor ---
 *   systemEnv
 *   (# setWindowEnv::< (# do myWindowEnv[] -> theWindowEnv[] #);
 *     myWindowEnv: @motifEnv (# ... #);
 *     ...
 *   #)
 *   ---]]
 *
 * The 'setWindowEnv' virtual and 'theWindowEnv' reference are declared in
 * basicssystemenv. 'myWindowEnv' does not have to be a motifenv instance
 * as long as it is at least an xtenv instance. (awenv, motifenv, xtenv).
 *
 * The xtenv instance assigned to theWindowEnv is used for scheduling
 * purposes to allow BETA coroutines to cooperate with the X event driven
 * user interface.
 *
 * For concurrency details, see BasicSystemEnv. *)

[[
---]]
```

---



# Basics Interface

```
ORIGIN '../xtenv';
BODY 'private/basicsbody';
INCLUDE 'motiflib';
INCLUDE 'callbackstruct';
INCLUDE '~beta/basiclib/external';

( *
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

MotifCallback: CallbackProc
( * Prefix for Motif callbacks. The callbacks have various data in
 * the callData structure.
 *)
( # callData: < XmAnyCallbackStruct;
  data: @callData;
  do call_data -> data.ptr;
  INNER;
  # );

MotifSelectPref: external ( * prefix for motif callbackentries *)
( # w, clientdata, calldata: @integer;
  status: @integer;
  cb: ^CallbackProc;
  enter (w, clientdata, calldata)
  do CExternalEntry;
  INNER;
  (if XsystemEnvPresent then
    (w, clientdata, calldata)
    -> (cb.widget, cb.client_data, cb.call_data);
    cb[] -> XsystemEnvHandleCallback;
  else
    (w, clientdata, calldata) -> cb;
  if)
  exit status
  # );

MotifString: IntegerObject
( * A MotifString is a compound string made of segments with three
 * elements: a character set, a direction, and a text.
 *)
( # init: < Object;
  setText:
  ( * Initialize THIS(MotifString) from the text t, using the
  * default character set and direction.
  *)
  ( # t: ^Text;
  enter t[]
  do ...
  # );
  setTextSegment:
  ( * Initialize THIS(MotifString) from the text t, using the
  * given character set, and direction.
  *)
  ( # t: ^Text;
    charset: ^text;
    direction: @integer;
  enter (t[], charset[], direction)
```



```

do ...
#);
getText:
(* Get the string segments of THIS(MotifString) composed from
 * the default character set, and default direction.
 *)
(# t: ^text;
do ...
exit t[]
#);
set: (* Set THIS(MotifString) directly *)
(# enter value #);
get: (* Get the external pointer for THIS(MotifString) *)
(# exit value #);
destroy:
(* Free the memory externally allocated for THIS(MotifString)
 *)
(# do ... #);
append:
(* Append the MotifString S to THIS(MotifString) *)
(# S: @MotifString
enter S
do ...
#);
prepend:
(* Prepend the MotifString S to THIS(MotifString) *)
(# S: @MotifString
enter S
do ...
#);
appendText:
(* Append to THIS(MotifString) a MotifString, constructed from
 * the text t, using the default character set, and direction
 *)
(# t: ^text
enter t[]
do ...
#);
appendSegment:
(* Append to THIS(MotifString) a MotifString, constructed from
 * the text t, using the given character set, and direction.
 *)
(# t: ^Text;
charset: ^text;
direction: @integer;
enter (t[], charset[], direction)
do ...
#);
prependText:
(* Prepend to THIS(MotifString) a MotifString, constructed
 * from the text t, using the default character set, and
 * direction
 *)
(# t: ^text
enter t[]
do ...
#);
prependSegment:
(* Prepend to THIS(MotifString) a MotifString, constructed
 * from the text t, using the given character set, and
 * direction.
 *)
(# t: ^Text;
charset: ^text;
direction: @integer;
enter (t[], charset[], direction)

```



```

do ...
#);
copyAppend:
(* Exit the external pointer for a MotifString which is a
 * concatenation of THIS(MotifString) and S.
 *)
(# S, copy: @MotifString;
enter S
do ...
exit copy
#);
copyPrepend:
(* Exit the external pointer for a MotifString which is a
 * concatenation of S and THIS(MotifString).
 *)
(# S, copy: @MotifString;
enter S
do ...
exit copy
#);
copy:
(* Exit a copy of THIS(MotifString) *)
(# S: @MotifString;
do ...
exit S
#);
hasSubString: BooleanValue
(* Exit true iff S is a substring of THIS(MotifString) *)
(# S: @MotifString
enter S
do ...
#);
equal: BooleanValue
(* Exit true iff THIS(MotifString) and S have the same
 * components
 *)
(# S: @MotifString
enter S
do ...
#);
eq: BooleanValue
(* Exit true iff THIS(MotifString) and S are equal on a
 * byte-by-byte basis.
 *)
(# S: @MotifString
enter S
do ...
#);
empty: BooleanValue
(* Exit true iff there are no non-zero length text components
 * in THIS(MotifString).
 *)
(# do ... #);
extent:
(* Exit the width and height, in pixels, of the smallest
 * rectangle, that will enclose THIS(MotifString) in a window
 * with the given fontlist.
 *)
(# width, height: @integer;
fontlist: @integer;
enter fontlist
do ...
exit (width, height)
#);
baseline: IntegerValue
(* Exit the number of pixels between the top of the character

```



```

    * box and the baseline of the first line of text in
    * THIS(MotifString).
    *)
    (# do ... #);
lineHeight: IntegerValue
    (* Exit the line height of THIS(MotifString) *)
    (# do ... #);
size: IntegerValue
    (* Exit the number of bytes in the external representation of
    * THIS(MotifString), including the text, tags, direction
    * indicators, and separators.
    *)
    (# do ... #);
noOfLines: IntegerValue
    (* Exit the number of lines in THIS(MotifString). This is
    * computed as the number of separators plus one.
    *)
    (# do ... #);
ScanSegments:
    (* Scans through each segment in THIS(MotifString). For each
    * segment, string is set to the text of the segment, charset
    * to the character set, direction to the direction, and
    * separator to is set to indicate whether the segment is a
    * separator or not.
    *)
    (# string: @text;
    charset: @text;
    direction: @integer;
    separator: @boolean;
    do ...
    #);
<<SLot MotifStringLib: attributes>>;
#);

```

**MotifStringArray:**

```

(* Array of MotifStrings. *)
(# initialSize:< IntegerObject(# do 25 -> value; INNER #);
init:<
    (* Initializes THIS(MotifStringArray) *)
    (# do ...; INNER #);
reset:<
    (* Reset THIS(MotifStringArray) *)
    (# do ...; INNER #);
clear:<
    (* Reset THIS(MotifStringArray) and free external memory
    * occupied by the MotifStrings.
    *)
    (# do ...; INNER #);
free:<
    (* Free the external memory THIS(MotifStringArray) occupies.
    * Does not free the MotifStrings. THIS(MotifStringArray)
    * should not be accessed after having invoked free on it.
    *)
    (# do ...; INNER #);
number:
    (* Number of MotifStrings currently in THIS(MotifStringArray)
    *)
    IntegerValue(# ... #);
inxget:
    (* Exits the MotifString of THIS(MotifStringArray) added as
    * number n
    *)
    (# n: @integer;
    S: @MotifString;
    enter n
    do ...;

```



```

        exit S
    #);
change:
    (* Changes the MotifString of THIS(MotifStringArray) added as
    * number n
    *)
    (# n: @integer;
     new: @MotifString;
     enter (n, new)
     do ...;
     #);
changeText:
    (* Instantiates a MotifString from t, using the default
    * character set and direction, and changes the MotifString of
    * THIS(MotifStringArray) added as number n.
    *)
    (# n: @integer;
     t: ^text;
     enter (n, t[])
     do ...;
     #);
add:
    (* Adds a MotifString to
    * THIS(MotifStringArray). THIS(MotifStringArray) is extended
    * if needed.
    *)
    (# S: @MotifString;
     enter S
     do ...;
     #);
addText:
    (* Instantiates a MotifString from t, using the default
    * character set and direction, and adds it to
    * THIS(MotifStringArray). THIS(MotifStringArray) is extended
    * if needed.
    *)
    (# t: ^text;
     enter t[]
     do ...;
     #);
    <<SLOT MotifStringArrayLib: attributes>>;
    private: @...;
    exit THIS(MotifStringArray)[]
    #);

--CoreLib: attributes--

```

**MotifFontList:** IntegerObject

```

    (* MotifFontLists are used to specify character sets for widgets
    * displaying MotifStrings.
    * Example:
    *   fontlst: @MotifFontList
    *   (# init:<<
    *       (#
    *         do ('*-courier-*-r-*--12-*', 'charset1') ->addText;
    *         ('*-courier-bold-o-*--14-*', 'charset2') ->addText;
    *         ('*-courier-medium-r-*--18-*', 'charset3')->addText;
    *       #)
    *     #);
    *)
    (# init:< Object;
     addText:
     (* Add one entry to THIS(MotifFontList) *)
     (# font, charset: ^text;
      enter (font[], charset[])
      do ...
    
```



```

    #);
add:
    (* Add one entry to THIS(MotifFontList). Here the font enter
    * parameter must be a pointer to an XFontStruct
    *)
    (# font: @integer; charset: ^text;
    enter (font, charset[])
    do ...
    #);
    <<SLot MotifFontListLib: attributes>>;
#);

-- XtObjectLib: attributes --

MotifStringResource:
(* A resource pattern corresponding to MotifStrings *)
(#
setText:
    (* Set THIS(MotifStringResource) as in MotifString.setText *)
    (# t: ^text
    enter t[]
    do ...
    #);
getText:
    (* Get the value of THIS(MotifStringResource) as in
    * MotifString.getText
    *)
    (# t: ^text
    do ...
    exit t[]
    #);
set:
    (* Set THIS(MotifStringResource) directly as in
    * MotifString.set
    *)
    (# value: @integer;
    enter value
    do ...
    #);
get:
    (* Get the value of THIS(MotifStringResource) directly as in
    * MotifString.get
    *)
    (# value: @Integer
    do ...
    exit value
    #);
resourceName:<
    (* The name of THIS(MotifStringResource) *)
    IntegerObject;
    <<SLot MotifStringResourceLib: attributes>>;
enter setText
exit getText
#);

MotifStringArrayResource
(# set:
    (# theString: ^MotifStringArray;
    enter theString[]
    do ...;
    #);
get:
    (# theString: ^MotifStringArray;
    do ...;
    exit theString[]
    #);

```



```

resourceName:< IntegerObject;
counterName:<
  (* Name of counter resource to update simultaneously when
   * invoking "set", and to obtain the number of items for "get".
   *)
  IntegerObject;
<<SLOT MotifStringArrayResourceLib: attributes>>;
enter set
exit get
#);

```

**ProcResource:**

```

(# (* proc:< external { * "Type" of procedure to call. Furtherbind
 *      to specify enter/exit parts } (# do cExternalEntry;
 *      INNER #);
 *)
resourceName:<
  (* Name of THIS(ProcResource) *)
  IntegerObject;
set:
  (# p: (###proc*) ##object;
   enter p##
   do ...
   #);
get:
  (# p: @integer;
   do ...
   exit p
   #);
<<SLOT ProcResourceLib: attributes>>;
enter set
exit get
#);

```

```

(* Tab Groups: When using the keyboard to traverse through a widget
 * hierarchy, primitive or manager widgets are grouped together into
 * what are known as tab groups. Any manager or primitive widget can
 * be a tab group. Within a tab group, move the focus to the next
 * widget within the tab group with its traversalOn resource set to
 * true by using the arrow keys. The order the widgets within a tab
 * group is traversed is determined by the order the widgets were
 * initialized. To move to the next tab group, press the Tab key, and
 * to move to the previous tab group, press Shift and the Tab key. The
 * order in which the tab groups are traversed is determined by the
 * order in which the application has registered the tab groups.
 *)

```

**AddTabGroup:**

```

(* Tab groups are ordinarily specified by the navigationType
 * resource. AddTabGroup is called to control the order of
 * traversal of tab groups. THIS(XtObject) is appended to the list
 * of tab groups to be traversed, and the navigationType resource is
 * set to XmEXCLUSIVE_TAB_GROUP.
 *)
(# do ... #);

```

**RemoveTabGroup:**

```

(* Removes THIS(XtObject) from the list of tab groups associated
 * with a particular widget hierarchy and sets the navigationType
 * resource to XmNONE.
 *)
(# do ... #);

```

```

(* Aliases used in CompositeLib *)

```

```

mCore: Core(# #);

```

```

--- CompositeLib: attributes ---

```



```
(* Redefinitions of widgets within composites making the composite
 * the default father of the widgets
 *)
Core: mCore
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)
```

---

Basics Interface

[Mjølner Informatics](#)



# Callbackstruct Interface

```
ORIGIN 'motiflib';
LIB_DEF 'motifcallback' '../../lib';

(*
 * COPYRIGHT
 *      Copyright Mjølner Informatics, 1992-97
 *      All rights reserved.
 *)

--LIB: attributes--
(**** Motif External Callback Structures ****)

XmAnyCallbackStruct mExternalRecord
(* External callback struct for Motif callbacks *)
(# reason: @
  (* Indicates why the callback was invoked. *)
  long(# pos::< (# do 0 -> value #) #);
  event: @
  (* Points to the X event that triggered the callback or is 0
   * if this callback was not triggered due to an XEvent.
   *)
  long(# pos::< (# do 4 -> value #) #);
#);

XmArrowButtonCallbackStruct XmAnyCallbackStruct
(* External callback struct for ArrowButton callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
   * event: Points to the X event that triggered the callback.
   * This event is 0 for the activateCallback if the callback was
   * triggered when Primitive's resource traversalOn was True or
   * if the callback was accessed through the ArmAndActivate
   * action routine.
   *)
  click_count: @
  (* This value is valid only when the reason is XmCR_ACTIVATE.
   * It contains the number of clicks in the last multiclick
   * sequence if the multiClick resource is set to
   * XmMULTICLICK_KEEP; otherwise it contains 1. The activate
   * callback is invoked for each click if multiClick is set to
   * XmMULTICLICK_KEEP.
   *)
  long(# pos::< (# do 8 -> value #) #);
#);

XmDrawingAreaCallbackStruct XmAnyCallbackStruct
(* External callback struct for DrawingArea callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
   * event: Points to the X event that triggered the callback.
   * This is 0 for the resizeCallback.
   *)
  window: @
  (* Is set to the widget window *)
  long(# pos::< (# do 8 -> value #) #);
#);

XmDrawnButtonCallbackStruct XmAnyCallbackStruct
(* External callback struct for DrawnButton callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
   * event: Points to the X event that triggered the
   * callback. This is 0 for resizeCallback. It is 0 for the
   * activateCallback if the callback was triggered when
   * Primitive's resource traversalOn was True or if the callback
```



```

    *   was accessed through the ArmAndActivate action routine.
    *)
window: @
    (* Is set to the window ID in which the event occurred. *)
    long(# pos::< (# do 8 -> value #));
click_count: @
    (* Contains the number of clicks in the last multiclick
    * sequence if the multiClick resource is set to
    * XmMULTICLICK_KEEP, otherwise it contains 1. The activate
    * callback is invoked for each click if multiClick is set to
    * XmMULTICLICK_KEEP.
    *)
    long(# pos::< (# do 12 -> value #));
#);

XmPushButtonCallbackStruct XmAnyCallbackStruct
(* External callback struct for PushButton callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
    *   event: Points to the X event that triggered the callback.
    *)
click_count: @
    (* This value is valid only when the reason is XmCR_ACTIVATE.
    * It contains the number of clicks in the last multiclick
    * sequence if the multiClick resource is set to
    * XmMULTICLICK_KEEP, otherwise it contains 1. The activate
    * callback is invoked for each click if multiClick is set to
    * XmMULTICLICK_KEEP.
    *)
    long(# pos::< (# do 8 -> value #));
#);

XmRowColumnCallbackStruct XmAnyCallbackStruct
(* External callback struct for RowColumn callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
    *   event: Points to the X event that triggered the callback.
    *)

    (* The following fields apply only when the callback reason is
    * XmCR_ACTIVATE; for all other callback reasons, these fields
    * are set to 0. The XmCR_ACTIVATE callback reason is generated
    * only when the application has supplied an entry callback,
    * which overrides any activation callbacks registered with the
    * individual RowColumn items.
    *)
widget: @
    (* Is set to the widget ID of the RowColumn item that has been
    *   activated
    *)
    long(# pos::< (# do 8 -> value #));
data: @
    (* Contains the client-data value supplied by the application
    * when the RowColumn item's activation callback was registered
    *)
    long(# pos::< (# do 12 -> value #));
callbackstruct: @
    (* Points to the callback structure generated by the RowColumn
    * item's activation callback
    *)
    long(# pos::< (# do 16 -> value #));
#);

XmScrollBarCallbackStruct XmAnyCallbackStruct
(* External callback struct for ScrollBar callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
    *   event: Points to the X event that triggered the callback.
    *)

```



```

value: @
    (* Contains the new slider location value. *)
    long(# pos::< (# do 8 -> value #));
pixel: @
    (* Is used only for toTopCallback and toBottomCallback. For
    * horizontal ScrollBars, it contains the x coordinate of where
    * the mouse button selection occurred. For vertical
    * ScrollBars, it contains the y coordinate.
    *)
    long(# pos::< (# do 12 -> value #));
#);

```

```

XmToggleButtonCallbackStruct XmAnyCallbackStruct
(* External callback struct for ToggleButton callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
    * event: Points to the X event that triggered the callback.
    *)
set: @
    (* Reflects the ToggleButton's current state when the callback
    * occurred, either True (selected) or False (unselected)
    *)
    long(# pos::< (# do 8 -> value #));
#);

```

```

XmListCallbackStruct: XmAnyCallbackStruct
(* External callback struct for MotifList callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
    * event: Points to the X event that triggered the callback. It
    * can be 0.
    *)
item: @
    (* Is the last item selected at the time of the event that
    * caused the callback. item points to a temporary storage
    * space that is reused after the callback is finished.
    * Therefore, if an application needs to save the item, it
    * should copy the item into its own data space.
    *)
    long(# pos::< (# do 8 -> value #));
item_length: @
    (* Is the length in bytes of item. *)
    long(# pos::< (# do 12 -> value #));
item_position: @
    (* Is the position of item in the MotifList's items
    * MotifStringArrayResource.
    *)
    long(# pos::< (# do 16 -> value #));
selected_items: @
    (* Is a list of items selected at the time of the event that
    * caused the callback. selected_items points to a temporary
    * storage space that is reused after the callback is finished.
    * Therefore, if an application needs to save the selected
    * list, it should copy the list into its own data space.
    *)
    long(# pos::< (# do 20 -> value #));
selected_item_count: @
    (* Is the number of items in the selected_items list. *)
    long(# pos::< (# do 24 -> value #));
selected_item_positions: @
    (* Is an array of integers, one for each selected item,
    * representing the position of each selected item in the
    * MotifList's items MotifStringArrayResource.
    * selected_item_positions points to a temporary storage space
    * that is reused after the callback is finished. Therefore,
    * if an application needs to save this array, it should copy
    * the array into its own data space.
    *)

```



```

    long(# pos::< (# do 28 -> value #));
selection_type: @
    (* Indicates that the most recent extended selection was the
     * initial selection (XmINITIAL), a modification of an existing
     * selection (XmMODIFICATION), or an additional noncontiguous
     * selection (XmADDITION).
     *)
    byte(# pos::< (# do 32 -> value #));
#);

```

```

XmSelectionBoxCallbackStruct XmAnyCallbackStruct
(* External callback struct for SelectionBox callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
   * event: Points to the X event that triggered the callback.
   *)
value: @
    (* Indicates the MotifString value selected by the user from
     * the SelectionBox list or entered into the SelectionBox text
     * field
     *)
    long(# pos::< (# do 8 -> value #));
length: @
    (* Indicates the size in bytes of the MotifString value *)
    long(# pos::< (# do 12 -> value #));
#);

```

```

XmCommandCallbackStruct: XmSelectionBoxCallbackStruct
(* External callback struct for Command callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
   * event: Points to the X event that triggered the callback.
   * value: Specifies the MotifString in the CommandArea length:
   * Specifies the byte size of value
   *)
#);

```

```

XmFileSelectionBoxCallbackStruct XmSelectionBoxCallbackStruct
(* External callback struct for FileSelectionBox callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
   * event: Points to the X event that triggered the callback.
   * value: Specifies the current value of the dirSpec resource.
   * length: Specifies the number of bytes in value.
   *)
mask: @
    (* Specifies the current value of the dirMask resource *)
    long(# pos::< (# do 16 -> value #));
mask_length: @
    (* Specifies the number of bytes in mask *)
    long(# pos::< (# do 20 -> value #));
dir: @
    (* Specifies the current base directory *)
    long(# pos::< (# do 24 -> value #));
dir_length: @
    (* Specifies the number of bytes in dir *)
    long(# pos::< (# do 28 -> value #));
pattern: @
    (* Specifies the current search pattern *)
    long(# pos::< (# do 32 -> value #));
pattern_length: @
    (* Specifies the number of bytes in pattern *)
    long(# pos::< (# do 36 -> value #));
#);

```

```

XmScaleCallbackStruct: XmAnyCallbackStruct
(* External callback struct for Scale callbacks *)
(# (* Inherited: reason: Indicates why the callback was invoked.
   * event: Points to the X event that triggered the callback.

```



```

    *)
    value: @
      (* Is the new slider value *)
      long(# pos::< (# do 8 -> value #));
#);

XmTextVerifyCallbackStruct XmAnyCallbackStruct
(* External callback struct for MotifText verification callbacks *)
(# bs: @boolsize;
  (* Inherited: reason: Indicates why the callback was invoked.
    * event: Points to the X event that triggered the callback.
    *)
  doit: @
    (* Indicates whether the action that invoked the callback is
      * performed. Setting doit to False negates the action.
      *)
    bool(# pos::< (# do 8 -> value #));
  currInsert: @
    (* Indicates the current position of the insert cursor. *)
    long(# pos::< (# do 8+bs -> value #));
  newInsert: @
    (* Indicates the position at which the user attempts to
      * position the insert cursor.
      *)
    long(# pos::< (# do 12+bs -> value #));
  startPos: @
    (* Indicates the starting position of the text to modify. If
      * the callback is not a modify verification callback, this
      * value is the same as currInsert
      *)
    long(# pos::< (# do 16+bs -> value #));
  endPos: @
    (* Indicates the ending position of the text to modify. If no
      * text is replaced or deleted, the value is the same as
      * startPos. If the callback is not a modify verification
      * callback, this value is the same as currInsert.
      *)
    long(# pos::< (# do 20+bs -> value #));
  text: @
    (* Address of a structure of type XmTextBlockRec. This
      * structure holds the textual information to be inserted.
      *)
    long(# pos::< (# do 24+bs -> value #));
#)

```

---

Callbackstruct Interface

[Mjølner Informatics](#)



# Primitive Interface

```
ORIGIN 'basics';
BODY 'private/primitivebody';

(*
 * COPYRIGHT
 *     Copyright Mjølner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

Primitive: Core
(* Primitive is a pattern used as a supporting superpattern for
 * other widget patterns. It handles border drawing and
 * highlighting, traversal activation and deactivation, and
 * various callbacks needed by other widgets.
 *)
(# init::<
    (# WidgetClass:<
        (#
            do INNER;
            (if value=0 then xmPrimitiveWidgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
foreground:
    (* Specifies the foreground drawing color used by
     * THIS(Primitive)
     *)
    IntegerResource(# resourceName:< XmNforeground #);
traversalOn:
    (* Specifies if traversal is activated for THIS(Primitive)
     *)
    BooleanResource(# resourceName:< XmNtraversalOn #);
navigationType:
    (* Controls whether THIS(Primitive) is a navigation group.
     *
     * + XmNONE indicates that THIS(Primitive) is not a
     *   navigation group.
     *
     * + XmTAB_GROUP indicates that THIS(Primitive) is included
     *   automatically in keyboard navigation, unless
     *   AddTabGroup has been called.
     *
     * + XmSTICKY_TAB_GROUP indicates that THIS(Primitive) is
     *   included automatically in keyboard navigation, even if
     *   AddTabGroup has been called.
     *
     * + XmEXCLUSIVE_TAB_GROUP indicates that THIS(Primitive)
     *   is included explicitly in keyboard navigation by the
     *   application. With XmEXCLUSIVE_TAB_GROUP, traversal of
     *   widgets within the group is based on the order of
     *   children. If THIS(Primitive)'s parent is a shell, the
     *   default is XmTAB_GROUP; otherwise, the default is
     *   XmNONE
     *)
    CharResource(# resourceName:< XmNnavigationType #);
highLightOnEnter:
    (* Specifies if the highlighting rectangle is drawn when
     * the cursor moves into THIS(Primitive). If the shell's
```



```

* focus policy is XmEXPLICIT, this resource is ignored, and
* THIS(Primitive) is highlighted when it has the focus. If
* the shell's focus policy is XmPOINTER and if this
* resource is True, the highlighting rectangle is drawn
* when the the cursor moves into THIS(Primitive). If the
* shell's focus policy is XmPOINTER and if this resource is
* False, the highlighting rectangle is not drawn when the
* the cursor moves into THIS(Primitive). The default is
* False.
*)
BooleanResource(# resourceName:< XmNhighLightOnEnter #);
unitType:
(* Provides the basic support for resolution independence.
* It defines the type of units THIS(Primitive) uses with
* sizing and positioning resources. If THIS(Primitive)'s
* parent is a specialization of Manager and if the unitType
* resource is not explicitly set, it defaults to the unit
* type of the parent widget. If THIS(Primitive)'s parent
* is not a specialization of Manager, the resource has a
* default unit type of XmPIXELS unitType can have the
* following values:
*
* + XmPIXELS - all values provided to THIS(Primitive) are
*   treated as normal pixel values.
*
* + Xm100TH_MILLIMETERS - all values provided to
*   THIS(Primitive) are treated as 1/100 millimeter.
*
* + Xm1000TH_INCHES - all values provided to
*   THIS(Primitive) are treated as 1/1000 inch.
*
* + Xm100TH_POINTS - all values provided to
*   THIS(Primitive) are treated as 1/100 point. A point is a
*   unit used in text processing applications and is defined
*   as 1/72 inch.
*
* + Xm100TH_FONT_UNITS - all values provided to the widget
*   are treated as 1/100 of a font unit. See the Motif
*   documentation for details on using font units.
*)
CharResource(# resourceName:< XmNunitType #);
highlightThickness:
(* Specifies the thickness of the highlighting rectangle *)
ShortResource(# resourceName:< XmNhighlightThickness #);
highlightColor:
(* Specifies the color of the highlighting rectangle. This
* color is used if the highlight pixmap resource is
* XmUNSPECIFIED_PIXMAP
*)
IntegerResource(# resourceName:< XmNhighlightColor #);
highlightPixmap:
(* Specifies the pixmap used to draw the highlighting
* rectangle
*)
IntegerResource(# resourceName:< XmNhighlightPixmap #);
shadowThickness:
(* Specifies the size of the drawn border shadow *)
ShortResource(# resourceName:< XmNshadowThickness #);
topShadowColor:
(* Specifies the color to use to draw the top and left
* sides of the border shadow. This color is used if the
* topShadowPixmap resource is unspecified
*)
IntegerResource(# resourceName:< XmNtopShadowColor #);
topShadowPixmap:
(* Specifies the pixmap to use to draw the top and left

```



```

    * sides of the border shadow
    *)
IntegerResource(# resourceName::< XmNtopShadowPixmap #);
bottomShadowColor:
(* Specifies the color to use to draw the bottom and right
 * sides of the border shadow. This color is used if the
 * topShadowPixmap resource is unspecified
 *)
IntegerResource(# resourceName::< XmNbottomShadowColor #);
bottomShadowPixmap:
(* Specifies the pixmap to use to draw the bottom and right
 * sides of the border shadow
 *)
IntegerResource(# resourceName::< XmNbottomShadowPixmap #);
userData:
(* Allows the application to attach any necessary specific
 * data to THIS(Primitive). It is an internally unused
 * resource.
 *)
IntegerResource(# resourceName::< XmNuserData #);

(* Callback definitions. Only the helpCallback pattern is
 * actually used in Primitive, the rest is used in various
 * specializations. But since many specializations define
 * callbacks with identical names, for convenience, most of
 * the corresponding patterns are declared here.
 *)
helpCallback:< CallbackProc
(* Called when the help key is pressed. The reason sent by
 * the callback is XmCR_HELP
 *)

activateCallback:< CallbackProc;
armCallback:< CallbackProc;
disarmCallback:< CallbackProc;
cascadingCallback:< CallbackProc;
exposeCallback:< CallbackProc;
resizeCallback:< CallbackProc;
valueChangedCallback:< CallbackProc;
incrementCallback:< CallbackProc;
decrementCallback:< CallbackProc;
pageincrementCallback:< CallbackProc;
pagedecrementCallback:< CallbackProc;
toTopCallback:< CallbackProc;
toBottomCallback:< CallbackProc;
dragCallback:< CallbackProc;
focusCallback:< CallbackProc;
losingFocusCallback:< CallbackProc;
modifyVerifyCallback:< CallbackProc;
motionVerifyCallback:< CallbackProc;
gainPrimaryCallback:< CallbackProc;
losePrimaryCallback:< CallbackProc;

installCallbacks:< (* Private *)
    (# do ...; INNER #);

<<SLOT PrimitiveLib: attributes>>
#) (* Primitive *);

(* Aliases used in CompositeLib *)
mPrimitive: Primitive(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widgets within composites making the composite
 * the default father of the widgets

```



```
*)  
Primitive: mPrimitive  
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);  
    do INNER  
    #)  
  #)
```

---

Primitive Interface

[Mjølner Informatics](#)



# Arrowbutton Interface

```
ORIGIN 'primitive';
```

```
(*  
 * COPYRIGHT  
 * Copyright Mjolner Informatics, 1992-97  
 * All rights reserved.  
 *)
```

```
BODY 'private/arrowbuttonbody';  
-- XtEnvLib: attributes --
```

**ArrowButton:** Primitive

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

```
(# init::<  
    (# WidgetClass::<  
        (#  
        do INNER;  
        (if value=0 then xmArrowButtonWidgetClass->value if)  
        #)  
    do INNER #);
```

```
(* Resources *)
```

**arrowDirection:**

```
(* Sets the arrow direction. Can be one of XmARROW_UP,  
 * XmARROW_DOWN, XmARROW_LEFT, and XmARROW_RIGHT.  
 *)
```

```
CharResource(# resourceName::< XmNarrowDirection #);
```

**multiClick:**

```
(* If a button click is followed by another button click  
 * within the time span specified by the display's multi-click  
 * time, and this resource is set to XmMULTICLICK_DISCARD, do * not process the second c  
 * XmMULTICLICK_KEEP, process the event and increment  
 * click_count in the callback structure. When the button is  
 * not in a menu, the default value is XmMULTICLICK_KEEP.  
 *)
```

```
CharResource(# resourceName::< XmNmultiClick #);
```

```
(* Callbacks *)
```

**ArrowButtonCallback:** MotifCallback

```
(* Prefix for ArrowButton callbacks *)
```

```
(# callData::< XmArrowButtonCallbackStruct do INNER #);
```

**activateCallback:** < ArrowButtonCallback

```
(* Called when THIS(ArrowButton) is activated. Activating  
 * THIS(ArrowButton) also disarms it. The reason sent by this  
 * callback is XmCR_ACTIVATE.  
 *)
```

**armCallback:** < ArrowButtonCallback

```
(* Called when THIS(ArrowButton) is armed. The reason sent by  
 * this callback is XmCR_ARM.  
 *)
```

**disarmCallback:** < ArrowButtonCallback

```
(* Called when THIS(ArrowButton) is disarmed. The reason for  
 * this callback is XmCR_DISARM.  
 *)
```

```
(* Inherited callbacks *)
```

**helpCallback:** < ArrowButtonCallback;



```

installCallbacks: < (* Private *)
  (# do ...; INNER #);

<<SLOT ArrowButtonLib: attributes>>;

#) (* ArrowButton *);

(* Alias used in CompositeLib *)
mArrowButton: ArrowButton(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widget within composites making the composite the
 * default father of the widget
 *)
ArrowButton: mArrowButton
  (# init: < (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Arrowbutton Interface

[Mjølner Informatics](#)



# Separator Interface

```
ORIGIN 'primitive';

(*
 * COPYRIGHT
 *      Copyright Mjølner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

Separator: Primitive
(* Separator is a primitive widget that separates items in a
 * display. Several different line drawing styles are provided,
 * as well as horizontal or vertical orientation.
 *)
(# init:<
    (# WidgetClass:<
        (#
            do INNER;
            (if value=0 then xmSeparatorWidgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
separatorType:
    (* Specifies the type of line drawing to be done in
     * THIS(Separator):
     *
     * + XmSINGLE_LINE - single line.
     *
     * + XmDOUBLE_LINE - double line.
     *
     * + XmSINGLE_DASHED_LINE - single-dashed line.
     *
     * + XmDOUBLE_DASHED_LINE - double-dashed line.
     *
     * + XmNO_LINE - no line.
     *
     * + XmSHADOW_ETCHED_IN - double line giving the effect of
     * a line etched into the window. The thickness of the
     * double line is equal to the value of shadowThickness.
     * For horizontal orientation, the top line is drawn in
     * topShadowColor and the bottom line is drawn in
     * bottomShadowColor. For vertical orientation, the left
     * line is drawn in topShadowColor and the right line is
     * drawn in bottomShadowColor.
     *
     * + XmSHADOW_ETCHED_OUT - double line giving the effect of
     * an etched line coming out from the window. The thickness
     * of the double line is equal to the value of
     * shadowThickness. For horizontal orientation, the top
     * line is drawn in bottomShadowColor and the bottom line is
     * drawn in topShadowColor. For vertical orientation, the
     * left line is drawn in bottomShadowColor and the right
     * line is drawn in topShadowColor.
     *)
    CharResource(# resourceName:< XmNseparatorType #);
margin:
    (* For horizontal orientation, specifies the space on the
     * left and right sides between the border of
     * THIS(Separator) and the line drawn. For vertical
```



```

    * orientation, specifies the space on the top and bottom
    * between the border of the THIS(Separator) and the line
    * drawn.
    *)
    ShortResource(# resourceName::< XmNmargin #);
orientation:
    (* Displays THIS(Separator) vertically or horizontally.
    * This resource can have values of XmVERTICAL and
    * XmHORIZONTAL.
    *)
    CharResource(# resourceName::< XmNOrientation #);

    <<SLOT SeparatorLib: attributes>>
    #) (* Separator *);

(* Alias used in CompositeLib *)
mSeparator: Separator(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widget within composites making the composite
 * the default father of the widget
 *)
Separator: mSeparator
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #)

```

---

## Separator Interface

[Mjølner Informatics](#)



# Scrollbar Interface

```
ORIGIN 'primitive';
BODY 'private/scrollbarbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

Scrollbar: Primitive
(* The ScrollBar widget allows the user to view data that is too
 * large to be displayed all at once. ScrollBars are usually
 * located inside a ScrolledWindow and adjacent to the widget that
 * contains the data to be viewed. When the user interacts with the
 * ScrollBar, the data within the other widget scrolls. A ScrollBar
 * consists of two arrows placed at each end of a rectangle. The
 * rectangle is called the scroll region. A smaller rectangle,
 * called the slider, is placed within the scroll region. The data
 * is scrolled by clicking either arrow, selecting on the scroll
 * region, or dragging the slider. When an arrow is selected, the
 * slider within the scroll region is moved in the direction of the
 * arrow by an amount supplied by the application. If the mouse
 * button is held down, the slider continues to move at a constant
 * rate. The ratio of the slider size to the scroll region size
 * typically corresponds to the relationship between the size of the
 * visible data and the total size of the data. For example, if 10
 * percent of the data is visible, the slider typically occupies 10
 * percent of the scroll region. This provides the user with a
 * visual clue to the size of the invisible data.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmScrollbarWidgetClass->value if)
        #)
        do INNER
    #);

    (* ScrollBar resources *)
value:
    (* Specifies the slider's position, between minimum and
     * (maximum - sliderSize).
     *)
    IntegerResource(# resourceName::< XmNvalue #);
minimum:
    (* Specifies the slider's minimum value. *)
    IntegerResource(# resourceName::< XmNminimum #);
maximum:
    (* Specifies the slider's maximum value. *)
    IntegerResource(# resourceName::< XmNmaximum #);
sliderSize:
    (* Specifies the length of the slider between the values of 1
     * and (maximum - minimum). When (maximum - minimum) is less
     * than 100, the default value is the lesser of 10 and (maximum
     * - minimum); otherwise, the default value is (maximum -
     * minimum) divided by 10.
     *)
    IntegerResource(# resourceName::< XmNsliderSize #);
showArrows:
```



```

    (* Specifies whether the arrows are displayed. *)
    BooleanResource(# resourceName::< XmNshowArrows #);

orientation:
    (* Specifies whether THIS(ScrollBar) is displayed vertically
    * or horizontally. This resource can have values of
    * XmVERTICAL and XmHORIZONTAL
    *)
    CharResource(# resourceName::< XmNorientation #);

processingDirection:
    (* Specifies whether the value for maximum should be on the
    * right or left side of minimum for horizontal ScrollBars or
    * above or below minimum for vertical ScrollBars. This
    * resource can have values of XmMAX_ON_TOP, XmMAX_ON_BOTTOM,
    * XmMAX_ON_LEFT, and XmMAX_ON_RIGHT. If THIS(ScrollBar) is
    * oriented vertically, the default value is XmMAX_ON_BOTTOM.
    * If THIS(ScrollBar) is oriented horizontally, the default
    * value may depend on the value of the stringDirection
    * resource.
    *)
    CharResource(# resourceName::< XmNprocessingDirection #);

increment:
    (* Specifies the amount by which the value increases or
    * decreases when the user takes an action that moves the
    * slider by one increment. The actual change in value is the
    * lesser of increment and (previous value - minimum) when the
    * slider moves to the end of THIS(ScrollBar) with the minimum
    * value, and the lesser of increment and (maximum - sliderSize
    * - previous value) when the slider moves to the end of
    * THIS(ScrollBar) with the maximum value.
    *)
    IntegerResource(# resourceName::< XmNincrement #);

pageIncrement:
    (* Specifies the amount by which the value increases or
    * decreases when the user takes an action that moves the
    * slider by one page increment. The actual change in value is
    * the lesser of pageIncrement and (previous value - minimum)
    * when the slider moves to the end of THIS(ScrollBar) with the
    * minimum value, and the lesser of pageIncrement and (maximum-
    * sliderSize - previous value) when the slider moves to the
    * end of THIS(ScrollBar) with the maximum value.
    *)
    IntegerResource(# resourceName::< XmNpageIncrement #);

initialDelay:
    (* Specifies the amount of time in milliseconds to wait before
    * starting continuous slider movement while a button is
    * pressed in an arrow or the scroll region.
    *)
    IntegerResource(# resourceName::< XmNinitialDelay #);

repeatDelay:
    (* Specifies the amount of time in milliseconds to wait
    * between subsequent slider movements after the initialDelay
    * has been processed.
    *)
    IntegerResource(# resourceName::< XmNrepeatDelay #);

troughColor:
    (* Specifies the color of the slider trough. *)
    IntegerResource(# resourceName::< XmNtroughColor #);

(* Utility patterns *)
values:
    (* A pattern that accesses THIS(ScrollBar)'s increment values
    *)
    (# value: @integer
    (* The slider position between the minimum and maximum
    * resources.
    *)
    );

```



```

slider_size: @integer
    (* The size of the slider as a value between zero and the
    * absolute value of maximum minus minimum. The size of
    * the slider varies, depending on how much of the slider
    * scroll area it represents.
    *);
increment: @integer
    (* The amount of increment and decrement *);
page_increment: @integer
    (* The amount of page increment and decrement *);
get:
    (#
    do ...
    exit (value, slider_size, increment, page_increment)
    #);
set:
    (# notify: @boolean
    (* If true, the valueChangedCallback is called *);
    enter (value, slider_size, increment,
    page_increment, notify)
    do ...
    #);
enter set
exit get
#);

(* Callbacks *)
ScrollBarCallback: MotifCallback
    (* Prefix for callbacks in THIS(ScrollBar) *)
    (# callData::< XmScrollBarCallbackStruct do INNER #);

valueChangedCallback: < ScrollBarCallback
    (* Called when the slider is released after being dragged. It
    * is also called in place of incrementCallback,
    * decrementCallback, pageIncrementCallback,
    * pageDecrementCallback, toTopCallback, or toBottomCallback
    * when one of these callbacks would normally be called but the
    * value of the corresponding resource is NULL. The reason
    * passed to the callback is XmCR_VALUE_CHANGED.
    *);

incrementCallback: < ScrollBarCallback
    (* Called when the user takes an action that moves
    * THIS(ScrollBar) by one increment and the value increases.
    * The reason passed to the callback is XmCR_INCREMENT
    *);

decrementCallback: < ScrollBarCallback
    (* Called when the user takes an action that moves
    * THIS(ScrollBar) by one increment and the value decreases.
    * The reason passed to the callback is XmCR_DECREMENT.
    *);

pageIncrementCallback: < ScrollBarCallback
    (* Called when the user takes an action that moves
    * THIS(ScrollBar) by one page increment and the value
    * increases. The reason passed to the callback is
    * XmCR_PAGE_INCREMENT.
    *);

pageDecrementCallback: < ScrollBarCallback
    (* Called when the user takes an action that moves
    * THIS(ScrollBar) by one page increment and the value
    * decreases. The reason passed to the callback is
    * XmCR_PAGE_DECREMENT.
    *);

toTopCallback: < ScrollBarCallback
    (* Called when the user takes an action that moves the slider
    * to the end of THIS(ScrollBar) with the minimum value. The
    * reason passed to the callback is XmCR_TO_TOP

```



```

    *);
toBottomCallback:< ScrollBarCallback
  (* Called when the user takes an action that moves the slider
   * to the end of THIS(ScrollBar) with the maximum value. The
   * reason passed to the callback is XmCR_TO_BOTTOM.
   *);
dragCallback:< ScrollBarCallback
  (* Called on each incremental change of position when the
   * slider is being dragged. The reason sent by the callback is
   * XmCR_DRAG.
   *);
  (* Inherited callbacks *)
helpCallback:< ScrollBarCallback;

installCallbacks:< (* Private *)
  (# do ...; INNER #);

  <<SLOT ScrollbarLib: attributes>>
#) (* Scrollbar *);

(* Alias used in CompositeLib *)
mScrollbar: Scrollbar(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widget within composites making the composite the
 * default father of the widget
 *)
Scrollbar: mScrollbar
  (# init:< (# GetFatherWidget:< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)
#)

```

---

Scrollbar Interface

[Mjølner Informatics](#)



# Label Interface

```
ORIGIN 'primitive';

( *
 * COPYRIGHT
 *     Copyright Mjølner Informatics, 1992-97
 *     All rights reserved.
 * )

-- XtEnvLib: attributes --

Label: Primitive
( * THIS(Label) can contain either a MotifString or a pixmap.
 * When THIS(Label) is insensitive, its text is stippled, or the
 * user-supplied insensitive pixmap is displayed
 * )
( # init: <
    ( # WidgetClass: <
        ( #
            do INNER;
                (if value=0 then xlabelWidgetClass->value if)
        #)
    do INNER #);

( * Resources * )
accelerator:
( * Sets the accelerator on a button widget in a menu, which
 * activates a visible or invisible button from the
 * keyboard. This resource is a string that describes a set
 * of modifiers and the key that may be used to select the
 * button. The format of this string is identical to that
 * used by the translations manager, with the exception that
 * only a single event may be specified and only KeyPress
 * events are allowed. Accelerators for buttons are
 * supported only for PushButton and ToggleButton in
 * Pulldown and Popup MenuPanels.
 * )
StringResource( # resourceName: < XmNaccelerator # );
acceleratorText:
( * Specifies the text displayed for the accelerator. The
 * text is displayed to the side of the label string or
 * pixmap. Accelerator text for buttons is displayed only
 * for PushButtons and ToggleButtons in Pulldown and Popup
 * Menus.
 * )
MotifStringResource( # resourceName: < XmNacceleratorText # );
alignment:
( * Specifies the label alignment for text or pixmap.
 *
 * + XmALIGNMENT_BEGINNING (left alignment) - causes the
 * left sides of the lines of text to be vertically
 * aligned with the left edge of the widget window. For a
 * pixmap, its left side is vertically aligned with the
 * left edge of the widget window. XmALIGNMENT_CENTER
 * (center alignment) - causes the centers of the lines of
 * text to be vertically aligned in the center of the
 * widget window. For a pixmap, its center is vertically
 * aligned with the center of the widget window.
 *
 * + XmALIGNMENT_END (right alignment) - causes the right
 * sides of the lines of text to be vertically aligned
 * with the right edge of the widget window. For a
 * pixmap, its right side is vertically aligned with the
```



```

*      right edge of the widget window.  The above
*      descriptions for text are correct when stringDirection
*      is XmSTRING_DIRECTION_L_TO_R.  When that resource is
*      XmSTRING_DIRECTION_R_TO_L, the descriptions for
*      XmALIGNMENT_BEGINNING and XmALIGNMENT_END are switched.
*)
CharResource(# resourceName::< XmNalignment #);
labelType:
(* Specifies the label type: XmSTRING - text displays
*   labelString.  XmPIXMAP - icon data in pixmap displays
*   labelPixmap or labelInsensitivePixmap.
*)
CharResource(# resourceName::< XmNlabelType #);
marginWidth:
(* Specifies the amount of spacing between the left side of
*   THIS(Label) (specified by marginLeft) and the right edge
*   of the left shadow, and the amount of spacing between the
*   right side of THIS(Label) (specified by marginRight) and
*   the left edge of the right shadow.
*)
ShortResource(# resourceName::< XmNmarginWidth #);
marginHeight:
(* Specifies the amount of spacing between the top of
*   THIS(Label) (specified by marginTop) and the bottom edge
*   of the top shadow, and the amount of spacing between the
*   bottom of THIS(Label) (specified by marginBottom) and the
*   top edge of the bottom shadow.
*)
ShortResource(# resourceName::< XmNmarginHeight #);
marginLeft:
(* Specifies the amount of spacing between the left edge of
*   the label text and the right side of the left margin
*   (specified by marginWidth).  This may be modified in
*   subpatterns.  For example, ToggleButton may increase this
*   field to make room for the toggle indicator and for
*   spacing between the indicator and label.  Whether this
*   actually applies to the left or right side of the label
*   may depend on the value of stringDirection.
*)
ShortResource(# resourceName::< XmNmarginLeft #);
marginRight:
(* Specifies the amount of spacing between the right edge
*   of the label text and the left side of the right margin
*   (specified by marginWidth).  This may be modified in
*   subpatterns.  For example, CascadeButton may increase
*   this field to make room for the cascade pixmap.  Whether
*   this actually applies to the left or right side of the
*   label may depend on the value of stringDirection.
*)
ShortResource(# resourceName::< XmNmarginRight #);
marginBottom:
(* Specifies the amount of spacing between the bottom of
*   the label text and the top of the bottom margin
*   (specified by marginHeight).  This may be modified in
*   subpatterns.  For example, CascadeButton may increase
*   this field to make room for the cascade pixmap
*)
ShortResource(# resourceName::< XmNmarginBottom #);
marginTop:
(* Specifies the amount of spacing between the top of the
*   label text and the bottom of the top margin (specified by
*   marginHeight).  This may be modified in subpatterns.  For
*   example, CascadeButton may increase this field to make
*   room for the cascade pixmap.
*)
ShortResource(# resourceName::< XmNmarginTop #);

```



**fontList:**

```
( * Specifies the font of the text used in THIS(Label).  If
 * this resource is unspecified at initialization, it is
 * initialized by looking up the parent hierarchy of
 * THIS(Label) for an ancestor that is a specialization of
 * BulletinBoard, VendorShell, or MenuShell.  If such an
 * ancestor is found, the font list is initialized to the
 * appropriate default font list of the ancestor widget
 * (defaultFontList for VendorShell and XmMenuShell,
 * labelFontList or buttonFontList for BulletinBoard).  Use
 * the MotifFontList pattern, if you want to set this
 * resource from the program.
 *)
```

```
IntegerResource(# resourceName:< XmNfontList #);
```

**labelPixmap:**

```
( * Specifies the pixmap when labelType is XmPIXMAP *)
IntegerResource(# resourceName:< XmNlabelPixmap #);
```

**labelInsensitivePixmap**

```
( * Specifies a pixmap used as the button face if labelType
 * is XmPIXMAP and the button is insensitive.
 *)
```

```
IntegerResource(# resourceName:< XmNlabelInsensitivePixmap #);
```

**labelString:**

```
( * Specifies the compound string when the labelType is
 * XmSTRING.  If this resource is not set, it is initialized
 * by converting the name of the widget to a MotifString.
 *)
```

```
MotifStringResource(# resourceName:< XmNlabelString #);
```

**mnemonic:**

```
( * Provides the user with an alternate means of selecting a
 * button.  A button in a MenuBar, a Popup MenuPane, or a
 * Pulldown MenuPane can have a mnemonic.  This resource
 * contains a keysym as listed in the X11 keysym table.  The
 * first character in the label string that exactly matches
 * the mnemonic in the character set specified in
 * mnemonicCharSet is underlined when the button is
 * displayed.  When a mnemonic has been specified, the user
 * activates the button by pressing the mnemonic key while
 * the button is visible.  If the button is a CascadeButton
 * in a MenuBar and the MenuBar does not have the focus, the
 * user must use the Alt modifier while pressing the
 * mnemonic.  The user can activate the button by pressing
 * either the shifted or the unshifted mnemonic key.
 *)
```

```
IntegerResource(# resourceName:< XmNmnemonic #);
```

**mnemonicCharSet:**

```
( * Specifies the character set of the mnemonic for
 * THIS(Label).  The default is determined dynamically
 * depending on the current language environment.
 *)
```

```
IntegerResource(# resourceName:< XmNmnemonicCharSet #);
```

**recomputeSize:**

```
( * Indicates whether THIS(Label) attempts to be big enough
 * to contain the label.  If True, setting a new labelString
 * or pixmap, accelerator text, margins, font, or label type
 * causes THIS(Label) to shrink or expand to exactly fit the
 * new labelString or pixmap.  If False, THIS(Label) never
 * attempts to change size on its own.
 *)
```

```
BooleanResource(# resourceName:< XmNrecomputeSize #);
```

**stringDirection:**

```
( * Specifies the direction in which the string is to be
 * drawn.  The following are the values:
 * XmSTRING_DIRECTION_L_TO_R - left to right
 * XmSTRING_DIRECTION_R_TO_L - right to left The default for
 * this resource is determined at creation time.  If no
```



```

    * value is specified for this resource and THIS(Label)'s
    * parent is a manager, the value is inherited from the
    * parent; otherwise, it defaults to
    * XmSTRING_DIRECTION_L_TO_R.
    *)
IntegerResource(# resourceName:< XmNstringDirection #);

<<SLOT LabelLib: attributes>>
#) (* label *);

(* Alias used in CompositeLib *)
mLabel: Label(# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
 * the default father of the widget
 *)
Label: mLabel
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Label Interface

[Mjølner Informatics](#)



# Cascadebutton Interface

```
ORIGIN 'label';
BODY 'private/cascadebutbody';

(*
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

CascadeButton: Label
(* CascadeButton links two MenuPanes or a MenuBar to a MenuPane.
 * It is used in menu systems and must have a RowColumn parent with
 * its rowColumnType resource set to XmMENU_BAR, XmMENU_POPUP or
 * XmMENU_PULLDOWN. It is the only widget that can have a Pulldown
 * MenuPane attached to it as a submenu. The submenu is displayed
 * when this widget is activated within a MenuBar, a PopupMenu, or a
 * PulldownMenu. Its visuals can include a label or pixmap and a
 * cascading indicator when it is in a Popup or Pulldown MenuPane;
 * or, it can include only a label or a pixmap when it is in a
 * MenuBar.
 *)
(# init: <
    (# WidgetClass: <
        (#
            do INNER;
            (if value=0 then xmCascadeButtonWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
subMenuId:
    (* Specifies the widget ID for the Pulldown MenuPane to be
     * associated with THIS(CascadeButton). The specified MenuPane
     * is displayed when THIS(CascadeButton) becomes armed. The
     * MenuPane must have been created with the appropriate
     * parentage depending on the type of menu used.
     *)
    IntegerResource(# resourceName:< XmNsubMenuId #);
cascadePixmap:
    (* Specifies the cascade pixmap displayed on one end of
     * THIS(CascadeButton) when a CascadeButton is used within a
     * Popup or Pulldown MenuPane and a submenu is attached. The
     * Label resources marginBottom, marginLeft, marginRight, and
     * marginTop may be modified to ensure that room is left for
     * the cascade pixmap. The default cascade pixmap is an arrow
     * pointing to the side of the menu where the submenu will
     * appear.
     *)
    IntegerResource(# resourceName:< XmNcascadePixmap #);
mappingDelay:
    (* Specifies the amount of time, in milliseconds, between when
     * THIS(CascadeButton) becomes armed and when it maps its
     * submenu. This delay is used only when the widget is within
     * a Popup or Pulldown MenuPane.
     *)
    IntegerResource(# resourceName:< XmNmappingDelay #);

    (* Utility patterns *)
highlight:
```



```

(* Draws or erases the shadow highlight around
 * THIS(CascadeButton): If draw if true, the shadow is drawn,
 * otherwise it is erased.
 *)
(# draw: @boolean;
 enter draw do ...
 #);

(* Callbacks *)
CascadeButtonCallback
(* Prefix for Label callbacks *)
MotifCallback(# do INNER #);

activateCallback:< CascadeButtonCallback
(* Called when the user activates THIS(CascadeButton), and
 * there is no submenu attached to pop up. The activation
 * occurs by releasing a mouse button or by typing the mnemonic
 * associated with the widget. The specific mouse button
 * depends on information in the RowColumn parent. The reason
 * sent by the callback is XmCR_ACTIVATE.
 *);
cascadingCallback:< CascadeButtonCallback
(* Called just prior to the mapping of the submenu associated
 * with THIS(CascadeButton). The reason sent by the callback
 * is XmCR_CASCADING.
 *);
(* Inherited callbacks *)
helpCallback:< CascadeButtonCallback;

installCallbacks:< (* Private *)
(# do ...; INNER #);

<<SLOT CascadeButtonLib: attributes>>
#) (* CascadeButton *);

(* Alias used in CompositeLib *)
mCascadeButton: CascadeButton (# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite the
 * default father of the widget
 *)
CascadeButton: mCascadeButton
(# init:<(# GetFatherWidget:< (# do THIS(Composite)->value #);
 do INNER
 #)
 #)
#)

```

---

Cascadebutton Interface

[Mjølner Informatics](#)



# Drawnbutton Interface

```
ORIGIN 'label';
BODY 'private/drawnbuttonbody';

(*
 * COPYRIGHT Copyright Mjolner Informatics, 1992-97
 * All rights reserved.
 *)

-- XtEnvLib: attributes --

DrawnButton: Label
(* A DrawnButton consists of an empty widget window surrounded
 * by a shadow border. It provides the application developer
 * with a graphics area that can have PushButton input semantics.
 *)
(# init:<
  (# WidgetClass:<
    (#
      do INNER;
      (if value=0 then xmDrawnButtonWidgetClass->value if)
    #)
  do INNER
  #);

  (* Resources *)
pushButtonEnabled:
  (* Enables or disables the three-dimensional shadow drawing
   * as in PushButton.
   *)
  BooleanResource(# resourceName:< XmNpushButtonEnabled #);
shadowType:
  (* Describes the drawing style for THIS(DrawnButton). This
   * resource can have the following values:
   *
   * + XmSHADOW_IN - draws THIS(DrawnButton) so that the
   *   shadow appears inset. This means that the bottom shadow
   *   visuals and top shadow visuals are reversed.
   *
   * + XmSHADOW_OUT - draws THIS(DrawnButton) so that the
   *   shadow appears outset.
   *
   * + XmSHADOW_ETCHED_IN - draws THIS(DrawnButton) using a
   *   double line. This gives the effect of a line etched into
   *   the window. The thickness of the double line is equal to
   *   the value of shadowThickness.
   *
   * + XmSHADOW_ETCHED_OUT - draws THIS(DrawnButton) using a
   *   double line. This gives the effect of a line coming out
   *   of the window. The thickness of the double line is equal
   *   to the value of shadowThickness.
   *)
  CharResource(# resourceName:< XmNshadowType #);
multiClick:
  (* If a button click is followed by another button click
   * within the time span specified by the display's
   * multi-click time, and this resource is set to
   * XmMULTICLICK_DISCARD, do not process the second click.
   * If this resource is set to XmMULTICLICK_KEEP, process the
   * event and increment click_count in the callback
   * structure. When THIS(DrawnButton) is not in a menu, the
   * default value is XmMULTICLICK_KEEP.
   *)
```



```

CharResource(# resourceName:< XmNmultiClick #);

(* Callbacks *)
DrawnButtonCallback: MotifCallback
(* Prefix for callbacks to THIS(DrawnButton) *)
(# callData:< XmDrawnButtonCallbackStruct do INNER #);

activateCallback:< DrawnButtonCallback
(* Called when THIS(DrawnButton) becomes selected. The
 * reason sent by this callback is XmCR_ACTIVATE.
 *);
armCallback:< DrawnButtonCallback
(* Called when THIS(DrawnButton) is armed. The reason sent
 * by this callback is XmCR_ARM.
 *);
disarmCallback:< DrawnButtonCallback
(* Called when THIS(DrawnButton) is disarmed. The reason
 * for this callback is XmCR_DISARM.
 *);
exposeCallback:< DrawnButtonCallback
(* Called when THIS(DrawnButton) receives an exposure
 * event. The reason sent by the callback is XmCR_EXPOSE.
 *);
resizeCallback:< DrawnButtonCallback
(* Called when THIS(DrawnButton) receives a resize event.
 * The reason sent by the callback is XmCR_RESIZE. The
 * event pointer returned for this callback is nil.
 *);
(* Inherited callbacks *)
helpCallback:< DrawnButtonCallback;

installCallbacks:< (* Private *)
(# do ...; INNER #);

<<SLOT DrawnButtonLib: attributes>>
#) (* DrawnButton *);

(* Alias used in CompositeLib *)
mDrawnButton: DrawnButton(# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
 * the default father of the widget
 *)
DrawnButton: mDrawnButton
(# init:<(# GetFatherWidget:< (# do THIS(Composite)->value #);
do INNER
#)
#)
#)

```

---



# Pushbutton Interface

```
ORIGIN 'label';
BODY 'private/pushbuttonbody';

(*
 * COPYRIGHT
 *     Copyright Mjølner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

PushButton: Label
(* PushButton issues commands within an application. It consists
 * of a text label or pixmap surrounded by a border shadow. When a
 * PushButton is selected, the shadow changes to give the appearance
 * that it has been pressed in. When a PushButton is unselected,
 * the shadow changes to give the appearance that it is out.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmPushButtonWidgetClass->value if)
        #)
        do INNER;
    #);

    (* Resources *)
fillOnArm:
    (* Forces THIS(PushButton) to fill the background with the
     * color specified by armColor when THIS(PushButton) is armed
     * and when this resource is set to True. If False, only the
     * top and bottom shadow colors are switched. When
     * THIS(PushButton) is in a menu, this resource is ignored and
     * assumed to be False.
     *)
    BooleanResource(# resourceName::< XmNfillOnArm #);
armColor:
    (* Specifies the color with which to fill the armed button.
     * fillOnArm must be set to True for this resource to have an
     * effect. The default for a color display is a color between
     * the background and the bottom shadow color. For a
     * monochrome display, the default is set to the foreground
     * color, and any text in the label appears in the background
     * color when the button is armed.
     *)
    IntegerResource(# resourceName::< XmNarmColor #);
armPixmap:
    (* Specifies the pixmap to be used as the button face if
     * labelType is XmPIXMAP and THIS(PushButton) is armed. This
     * resource is disabled when THIS(PushButton) is in a menu.
     *)
    IntegerResource(# resourceName::< XmNarmPixmap #);
defaultButtonShadowThickness
    (* This resource specifies the width of the default button
     * indicator shadow. If this resource is zero, the width of
     * the shadow comes from the value of the showAsDefault
     * resource. If this resource is greater than zero, the
     * showAsDefault resource is only used to specify whether
     * THIS(PushButton) is the default.
     *)
    ShortResource(# resourceName::< XmNarmPixmap #);
```



```

multiClick:
(* If a button click is followed by another button click
 * within the time span specified by the display's multi-click
 * time, and this resource is set to XmMULTICLICK_DISCARD, do * not process the second click
 * XmMULTICLICK_KEEP, process the event and increment
 * click_count in the callback structure. When
 * THIS(PushButton) is not in a menu, the default value is
 * XmMULTICLICK_KEEP.
 *)
CharResource(# resourceName:< XmNmultiClick #);
ShowAsDefault:
(* If defaultButtonShadowThickness is greater than zero, a
 * value greater than zero in this resource specifies to mark
 * THIS(PushButton) as the default button. If
 * defaultButtonShadowThickness is zero, a value greater than
 * zero in this resource specifies to mark THIS(PushButton) as
 * the default button with the shadow thickness specified by
 * this resource. The space between the shadow and the default
 * shadow is equal to the sum of both shadows. The default
 * value is zero. When this value is not zero, the Label
 * resources marginLeft, marginRight, marginTop, and
 * marginBottom may be modified to accommodate the second
 * shadow. This resource is disabled when THIS(PushButton) is
 * in a menu.
 *)
ShortResource(# resourceName:< XmNshowAsDefault #);

(* Callbacks *)
PushButtonCallback: MotifCallback
(* Prefix for callbacks to THIS(PushButton) *)
(# callData:< XmPushButtonCallbackStruct do INNER #);

activateCallback:< PushButtonCallback
(* Called when THIS(PushButton) is activated.
 * THIS(PushButton) is activated when the user presses and
 * releases the active mouse button while the pointer is inside
 * THIS(PushButton). Activating THIS(PushButton) also disarms
 * it. For this callback the reason is XmCR_ACTIVATE.
 *);
armCallback:< PushButtonCallback
(* Called when THIS(PushButton) is armed. THIS(PushButton) is
 * armed when the user presses the active mouse button while
 * the pointer is inside THIS(PushButton). For this callback
 * the reason is XmCR_ARM.
 *);
disarmCallback:< PushButtonCallback
(* Called when THIS(PushButton) is disarmed. THIS(PushButton)
 * is disarmed when the user presses and releases the active
 * mouse button while the pointer is inside THIS(PushButton).
 * For this callback, the reason is XmCR_DISARM.
 *);
(* Inherited callbacks *)
helpCallback:< PushButtonCallback;

installCallbacks:< (* Private *)
(# do ...; INNER #);

<<SLOT PushButtonLib: attributes>>
#) (* PushButton *);

(* Alias used in CompositeLib *)
mPushButton: PushButton(# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite the

```



```
* default father of the widget
*)
PushButton: mPushButton
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)
```

---

Pushbutton Interface

[Mjølner Informatics](#)



# Togglebutton Interface

```
ORIGIN 'label';
BODY 'private/togglebuttonbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

ToggleButton: Label
(* ToggleButton sets nontransitory state data within an
 * application.  Usually this widget consists of an indicator
 * (square or diamond) with either text or a pixmap on one side of
 * it.  However, it can also consist of just text or a pixmap
 * without the indicator.  The toggle graphics display a 1-of-many
 * or N-of-many selection state.  When a toggle indicator is
 * displayed, a square indicator shows an N-of-many selection state
 * and a diamond indicator shows a 1-of-many selection state.
 * ToggleButton implies a selected or unselected state.  In the case
 * of a label and an indicator, an empty indicator (square or
 * diamond shaped) indicates that ToggleButton is unselected, and a
 * filled indicator shows that it is selected.  In the case of a
 * pixmap toggle, different pixmaps are used to display the
 * selected/unselected states.
 *)
(# init:<
    (# WidgetClass:<
        (#
            do INNER;
            (if value=0 then xmToggleButtonWidgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
indicatorSize:
    (* Sets the size of the indicator.  A value of
     * XmINVALID_DIMENSION causes the indicator to be set to the
     * size of the font of the label string.
     *)
    ShortResource(# resourceName:< XmNindicatorSize #);
indicatorType:
    (* Specifies if the indicator is a 1-of or N-of indicator.
     * For the 1-of indicator, the value is XmONE_OF_MANY.  For the
     * N-of indicator, the value is XmN_OF_MANY.  The N-of-many
     * indicator is square. The 1-of-many indicator is diamond
     * shaped.  This resource specifies only the visuals and does
     * not enforce the behavior.  When THIS(ToggleButton) is in a
     * RadioBox, the default is XmONE_OF_MANY; otherwise, the
     * default is XmN_OF_MANY.
     *)
    CharResource(# resourceName:< XmNindicatorType #);
visibleWhenOff:
    (* Indicates that the toggle indicator is visible in the
     * unselected state when the Boolean value is True.  When
     * THIS(ToggleButton) is in a menu, the default value is False.
     * When THIS(ToggleButton) is in a RadioBox, the default value
     * is True.
     *)
    BooleanResource(# resourceName:< XmNvisibleWhenOff #);
```



```

spacing:
(* Specifies the amount of spacing between the toggle
 * indicator and the toggle label (text or pixmap).
 *)
ShortResource(# resourceName:< XmNspacing #);

selectPixmap:
(* Specifies the pixmap to be used as the button face if
 * labelType is XmPIXMAP and THIS(ToggleButton) is selected.
 * When THIS(ToggleButton) is unselected, the pixmap specified
 * in Label's labelPixmap is used.
 *)
IntegerResource(# resourceName:< XmNselectPixmap #);

selectInsensitivePixmap
(* Specifies a pixmap used as the button face when
 * THIS(ToggleButton) is selected and the button is insensitive
 * if the Label resource labelType is set to XmPIXMAP. If the
 * ToggleButton is unselected and the button is insensitive,
 * the pixmap in labelInsensitivePixmap is used as the button
 * face.
 *)
IntegerResource(# resourceName:<XmNselectInsensitivePixmap #);

set:
(* Displays the button in its selected state if set to True.
 * This shows some conditions as active when a set of buttons
 * first appears.
 *)
BooleanResource(# resourceName:< XmNset #);

indicatorOn:
(* Specifies that a toggle indicator is drawn to one side of
 * the toggle text or pixmap when set to True. When set to
 * False, no space is allocated for the indicator, and it is
 * not displayed. If indicatorOn is True, the indicator
 * shadows are switched when the button is selected or
 * unselected, but, any shadows around the entire widget are
 * not switched. However, if indicatorOn is False, any shadows
 * around the entire widget are switched when the toggle is
 * selected or unselected.
 *)
BooleanResource(# resourceName:< XmNindicatorOn #);

fillOnSelect:
(* Fills the indicator with the color specified in selectColor
 * and switches the top and bottom shadow colors when set to
 * True. Otherwise, it switches only the top and bottom shadow
 * colors.
 *)
BooleanResource(# resourceName:< XmNfillOnSelect #);

selectColor:
(* Allows the application to specify what color fills the
 * center of the square or diamond-shaped indicator when it is
 * set. If this color is the same as either the top or the
 * bottom shadow color of the indicator, a one-pixel-wide
 * margin is left between the shadows and the fill; otherwise,
 * it is filled completely. This resource's default for a
 * color display is a color between the background and the
 * bottom shadow color. For a monochrome display, the default
 * is set to the foreground color. The meaning of this
 * resource is undefined when indicatorOn is False.
 *)
IntegerResource(# resourceName:< XmNselectColor #);

(* Utility patterns *)

state:
(* Used to manipulate the state of THIS(ToggleButton) directly
 *)
(# value: @boolean;
 get:

```



```

    (* Returns True if THIS(ToggleButton) is selected and
    * False if THIS(ToggleButton) is unselected.
    *)
    (#
    do ...;
    exit value
    #);
set:
    (* Enters a Boolean value that indicates whether
    * THIS(ToggleButton) state should be selected or
    * unselected. If True, the button state is selected; if
    * False, the button state is unselected. The notify
    * parameter indicates whether the valueChangedCallback is
    * called; it can be either True or False. When this
    * argument is True and THIS(ToggleButton) is a child of a
    * RowColumn widget whose radioBehavior is True, selecting
    * THIS(ToggleButton) causes other ToggleButton and
    * ToggleButtonGadget children of the RowColumn to be
    * unselected.
    *)
    (# notify: @boolean
    enter (value, notify)
    do ...;
    #)
enter set
exit get
#);

(* Callbacks *)
ToggleButtonCallback: MotifCallback
    (* Prefix for ToggleButton callbacks *)
    (# callData::< XmToggleButtonCallbackStruct do INNER #);

armCallback:< ToggleButtonCallback
    (* Called when THIS(ToggleButton) is armed. To arm
    * THIS(ToggleButton), press the active mouse button while the
    * pointer is inside THIS(ToggleButton). The reason sent by
    * this callback is XmCR_ARM.
    *);

disarmCallback:< ToggleButtonCallback
    (* Called when THIS(ToggleButton) is disarmed. To disarm
    * THIS(ToggleButton), press and release the active mouse
    * button while the pointer is inside the THIS(ToggleButton).
    * THIS(ToggleButton) is also disarmed when the user moves out
    * of THIS(ToggleButton) and releases the mouse button when the
    * pointer is outside THIS(ToggleButton). For this callback,
    * the reason is XmCR_DISARM.
    *);

valueChangedCallback:< ToggleButtonCallback
    (* Called when THIS(ToggleButton) value is changed. To change
    * the value, press and release the active mouse button while
    * the pointer is inside THIS(ToggleButton). This action also
    * causes THIS(ToggleButton) to be disarmed. For this
    * callback, the reason is XmCR_VALUE_CHANGED.
    *);

(* Inherited callbacks *)
helpCallback:< ToggleButtonCallback;

installCallbacks:< (* Private *)
    (# do ...; INNER #);

<<SLOT ToggleButtonLib: attributes>>
#) (* ToggleButton *);

(* Alias used in CompositeLib *)
mToggleButton: ToggleButton(# #);

```



```

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite the
 * default father of the widget
 *)
ToggleButton: mToggleButton
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

ToggleButton Interface

[Mjølner Informatics](#)



# Lists Interface

```
ORIGIN 'primitive';
BODY 'private/listsbody';
INCLUDE 'scrolledwindow';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

MotifList: Primitive
(* MotifList allows a user to select one or more items from a
 * group of choices.  Items are selected from the list in a
 * variety of ways, using both the pointer and the keyboard.
 * MotifList operates on a StringArray that are defined by the
 * application.  Each string becomes an item in the MotifList,
 * with the first string becoming the item in position 1, the
 * second string becoming the item in position 2, and so on.
 *)
(# init:<
    (# WidgetClass:<
        (#
            do INNER;
            (if value=0 then xmListWidgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
listSpacing:
    (* Specifies the spacing between list items.  This spacing
     * increases by the value of the highlightThickness resource
     * in Primitive.
     *)
    IntegerResource(# resourceName:< XmNlistSpacing #);
listMarginWidth:
    (* Specifies the width of the margin between the list
     * border and the items.
     *)
    ShortResource(# resourceName:< XmNlistMarginWidth #);
listMarginHeight:
    (* Specifies the height of the margin between the list
     * border and the items.
     *)
    ShortResource(# resourceName:< XmNlistMarginHeight #);
fontList:
    (* Specifies the font list associated with the list items.
     * This is used in conjunction with the visibleItemCount
     * resource to determine the height of the List widget.  If
     * this value is unspecified at initialization, it is
     * initialized by looking up the parent hierarchy of the
     * widget for an ancestor that is a specialization of
     * BulletinBoard, VendorShell, or MenuShell.  If such an
     * ancestor is found, the font list is initialized to the
     * appropriate default font list of the ancestor widget
     * (defaultFontList for VendorShell and MenuShell,
     * textFontList for BulletinBoard).
     *)
    IntegerResource(# resourceName:< XmNfontList #);
stringDirection:
```



```

(* Specifies the initial direction to draw the strings.
 * The values are XmSTRING_DIRECTION_L_TO_R and
 * XmSTRING_DIRECTION_R_TO_L. The value of this resource is
 * determined at creation time. If THIS(MotifList)'s parent
 * is a manager, this value is inherited from
 * THIS(MotifList)'s parent, otherwise it is set to
 * XmSTRING_DIRECTION_L_TO_R.
 *)
IntegerResource(# resourceName::< XmNstringDirection #);
items: MotifStringArrayResource
(* Points to an array of MotifStrings, that are to be
 * displayed as the list items.
 *)
(# resourceName::< XmNitems;
 counterName::< XmNitemCount;
 #);
itemCount:
(* Specifies the total number of items. This number must
 * match the items resource. It is automatically updated by
 * THIS(MotifList) whenever an item is added to or deleted
 * from THIS(MotifList).
 *)
IntegerResource(# resourceName::< XmNitemCount #);
selectedItems: MotifStringArrayResource
(* Array of MotifStrings that represents the list items
 * that are currently selected, either by the user or by the
 * application.
 *)
(# resourceName::< XmNselectedItems;
 counterName::< XmNselectedItemCount;
 #);
selectedItemCount:
(* Specifies the number of strings in the selected items
 * list.
 *)
IntegerResource(# resourceName::< XmNselectedItemCount #);
visibleItemCount:
(* Specifies the number of items that can fit in the
 * visible space of the list work area. THIS(MotifList)
 * uses this value to determine its height.
 *)
IntegerResource(# resourceName::< XmNvisibleItemCount #);
selectionPolicy:
(* Defines the interpretation of the selection action.
 * This can be one of the following:
 *
 * + XmSINGLE_SELECT - An item is selected when mouse
 * button 1 is clicked on it. The previously selected item
 * is deselected. Only one item can be selected at a time.
 *
 * + XmMULTIPLE_SELECT - An unselected item is selected
 * when mouse button 1 is clicked on it. Previously selected
 * items remains selected. A selected item is deselected
 * when mouse button 1 is clicked on it.
 *
 * + XmEXTENDED_SELECT - When mouse button 1 is pressed and
 * held down while dragged up or down, all items between the
 * initial item and the pointer are selected. Releasing
 * mouse button 1 stops the selection proces, and those
 * items selected remains selected.
 *
 * + XmBROWSE_SELECT - An item is selected when mouse
 * button 1 is pressed on it. Dragging the pointer up or
 * down from that point causes each succeeding item to be
 * selected while the preceding item is deselected. When the
 * mouse button is released, the item on which the pointer

```



```

    * rests is selected. This is the default.
    *)
    CharResource(# resourceName::< XmNselectionPolicy #);
automaticSelection:
    (* Invokes singleSelectionCallback when the user moves into
    * a new item if the value is True and the selection mode is
    * either XmBROWSE_SELECT or XmEXTENDED_SELECT. If False,
    * no selection callbacks are invoked until the user
    * releases the mouse button.
    *)
    BooleanResource(# resourceName::< XmNautomaticSelection #);
doubleClickInterval:
    (* If a button click is followed by another button click
    * within the time span specified by this resource (in
    * milliseconds), the button clicks are considered a
    * double-click action, rather than two single-click
    * actions. The default value is the display's multi-click
    * time.
    *)
    IntegerResource(# resourceName::< XmNdoubleClickInterval #);

(* Utility patterns *)
addItem:
    (* Adds an item to THIS(MotifList) at the given position.
    * If unselected is False, when the item is inserted, it is
    * compared with the current selectedItems list. If the new
    * item matches an item on the selected list, it appears
    * selected. If unselected is True, the item does not appear
    * selected, even if it matches an item in the current
    * selectedItems list.
    *)
    (# item: @MotifString;
     position: @integer;
     unselected: @boolean;
     enter (item, position, unselected)
     do ...
     #);
addItem:
    (* Adds the specified items to THIS(MotifList) at the given
    * position. The first item_count items of the items array
    * are added. When the items are inserted into
    * THIS(MotifList), they are compared with the current
    * selectedItems list. If the any of the new items matches
    * an item on the selected list, it appears selected.
    *)
    (# items: ^MotifStringArray;
     position: @integer;
     enter (items[], position)
     do ...
     #);
getItemPos:
    (* Obtain item number 'pos' *)
    (# pos: @integer;
     item: @MotifString;
     enter pos
     do ...
     exit item
     #);
getItemPos:
    (* Obtain item number 'pos1' to 'pos2' *)
    (# pos1, pos2: @integer;
     theitems: ^MotifStringArray;
     enter (pos1, pos2)
     do ...
     exit theitems[]
     #);

```



```

deleteItem:
(* Deletes a specified item from THIS(MotifList). A
 * warning message appears if the item does not exist.
 *)
(# item: @MotifString;
 enter item
 do ...
 #);

deleteItems:
(* Deletes the specified items from THIS(MotifList). A
 * warning message appears if the items do not exist.
 *)
(# items: ^MotifStringArray;
 enter items[]
 do ...
 #);

deletePos:
(* Deletes an item at a specified position. A warning
 * message appears if the position does not exist.
 *)
(# position: @integer
 enter position
 do ...
 #);

deleteItemsPos:
(* Deletes the specified number of items from the list
 * starting at the specified position.
 *)
(# item_count, position: @integer
 enter (item_count, position)
 do ...
 #);

deleteAllItems:
(* Deletes all items from THIS(MotifList) *)
(# position: @integer
 enter position
 do ...
 #);

replaceItem:
(* Replaces the specified item of THIS(MotifList) with the
 * corresponding new item.
 *)
(# old_item, new_item: @MotifString;
 enter (old_item, new_item)
 do ...
 #);

replaceItemPos:
(* Replaces an item of THIS(MotifList) with new_item, at
 * the specified position in THIS(MotifList).
 *)
(# new_item: @MotifString;
 position: @integer;
 enter (new_item, position)
 do ...
 #);

replaceItems:
(* Replaces each specified item of THIS(MotifList) with a
 * corresponding new item.
 *)
(# old_items, new_items: ^MotifStringArray;
 enter (old_items[], new_items[])
 do ...
 #);

replaceItemsPos:
(* Replaces items of THIS(MotifList) with new_items,
 * starting at the specified position in THIS(MotifList).
 *)

```



```

    *)
    (# new_items: ^MotifStringArray;
       position: @integer;
       enter (new_items[], position)
       do ...
    #);

selectItem:
    (* Highlights and adds the specified item to the current
    * selected list. The notify parameter specifies a Boolean
    * value that when True invokes the selection callback for
    * the current mode. From an application interface view,
    * calling this function with notify True is
    * indistinguishable from a user-initiated selection action.
    *)
    (# item: @MotifString;
       notify: @boolean;
       enter (item, notify)
       do ...
    #);

selectPos:
    (* Highlights an item at the specified position and adds it
    * to the list of selected items. The notify parameter
    * specifies a Boolean value that when True invokes the
    * selection callback for the current mode. From an
    * application interface view, calling this function with
    * notify True is indistinguishable from a user-initiated
    * selection action.
    *)
    (# position: @integer;
       notify: @boolean;
       enter (position, notify)
       do ...
    #);

deselectItem:
    (* Unhighlights and removes the specified item from the
    * selected list.
    *)
    (# item: @MotifString;
       enter item
       do ...
    #);

deselectPos:
    (* Unhighlights the item at the specified position and
    * deletes it from the list of selected items.
    *)
    (# position: @integer;
       enter position
       do ...
    #);

deselectAllItems:
    (* Unhighlights and removes all items from the selected
    * list.
    *)
    (#
       do ...
    #);

setPos:
    (* Makes the item at the given position the first visible
    * position in THIS(MotifList). Setting pos to 1 indicates
    * that the first item in the list is the first visible
    * item; setting pos to 2 indicates that the second item is
    * the first visible item; and so on. Setting pos to 0
    * indicates that the last item in the list is the first
    * visible item.
    *)
    (# position: @integer;

```



```

    enter position
    do ...
    #);

setBottomPos:
    (* Makes the item at the specified position the last
    * visible item in THIS(MotifList). Setting pos to 1
    * indicates that the first item in the list is the last
    * visible item; setting pos to 2 indicates that the second
    * item is the last visible item; and so on. Setting pos to
    * 0 indicates that the last item in the list is the last
    * visible item.
    *)
    (# position: @integer;
    enter position
    do ...
    #);

setItem:
    (* Makes an existing item the first visible item in
    * THIS(MotifList). The item can be any valid item in the
    * list.
    *)
    (# item: @MotifString;
    enter item
    do ...
    #);

setBottomItem:
    (* Makes an existing item the last visible item in
    * THIS(MotifList). The item can be any valid item in the
    * list.
    *)
    (# item: @MotifString;
    enter item
    do ...
    #);

setAddMode:
    (* Allows applications control over Add Mode in the
    * extended selection model. If add_mode is True, Add Mode
    * is activated. If add_mode is False, Add Mode is
    * deactivated.
    *)
    (# add_mode: @integer;
    enter add_mode
    do ...
    #);

itemExists: BooleanValue
    (* Checks if a specified item is present in the list. *)
    (# item: @MotifString;
    enter item
    do ...
    #);

itemPos: IntegerValue
    (* Exits the position of the first instance of the
    * specified item in a List.
    *)
    (# item: @MotifString;
    enter item
    do ...
    #);

getMatchPos: IntegerValue
    (* Exits an array of positions where a specified item is
    * found in THIS(MotifList).
    *)
    (# item: @MotifString;
    position_list: [1]@integer;
    enter item
    do ...

```



```

    exit position_list
  #);
getSelectedPos: IntegerValue
  (* Exits an array of the positions of the selected items in
   * THIS(MotifList).
   *)
  (# position_list: [1]@integer;
  do ...
  exit position_list
  #);
setHorizPos:
  (* Scrolls to the specified position in the list.  If
   * THIS(MotifList).listSizePolicy is set to XmCONSTANT or
   * XmRESIZE_IF_POSSIBLE and the horizontal ScrollBar is
   * currently visible, the value resource of the horizontal
   * ScrollBar is set to the specified position and the
   * visible portion of the list is updated with the new
   * value.  This is equivalent to moving the horizontal
   * ScrollBar to the specified position.
   *)
  (# position: @integer;
  enter position
  do ...
  #);

(* Callbacks *)
MotifListCallback: MotifCallback
  (* Prefix for MotifList specific Callbacks *)
  (# callData:< XmListCallBackStruct do INNER #);

singleSelectionCallback< MotifListCallback
  (* Called when an item is selected in single selection
   * mode.  The reason is XmCR_SINGLE_SELECT.
   *);
multipleSelectionCallback< MotifListCallback
  (* Called when an item is selected in multiple selection
   * mode.  The reason is XmCR_MULTIPLE_SELECT.
   *);
extendedSelectionCallback< MotifListCallback
  (* Called when items are selected using the extended
   * selection mode. The reason is XmCR_EXTENDED_SELECT.
   *);
browseSelectionCallback< MotifListCallback
  (* Called when an item is selected in the browse selection
   * mode.  The reason is XmCR_BROWSE_SELECT.
   *);
defaultActionCallback< MotifListCallback
  (* Called when an item is double clicked.  The reason is
   * XmCR_DEFAULT_ACTION.
   *);
(* Inherited callbacks *)
helpCallback:< MotifListCallback;

installCallbacks:< (* Private *)
  (# do ...; INNER #);

<<SLOT MotifListLib: attributes>>
#) (* MotifList *);

ScrolledList: MotifList
  (* A ScrolledList is a MotifList that is contained within a
   * ScrolledWindow.
   *)
  (# init:< (# WidgetClass:< (# do ... #);
  do INNER
  #);

```



```

(* Resources belonging to the INNER MotifList *)
horizontalScrollbar:
  (* The horizontal Scrollbar. *)
  IntegerResource(# resourceName:< XmNhorizontalScrollbar #);
listSizePolicy:
  (* Controls the reaction of THIS(MotifList) when an item
   * grows horizontally beyond the current size of
   * THIS(MotifList) work area. If the value is XmCONSTANT,
   * THIS(MotifList) viewing area does not grow, and a
   * horizontal ScrollBar is added for a ScrolledList. If
   * this resource is set to XmVARIABLE, the List grows to
   * match the size of the longest item, and no horizontal
   * ScrollBar appears. When the value of this resource is
   * XmRESIZE_IF_POSSIBLE, THIS(MotifList) attempts to grow or
   * shrink to match the width of the widest item. If it
   * cannot grow to match the widest size, a horizontal
   * ScrollBar is added for a ScrolledList if the longest item
   * is wider than THIS(MotifList) viewing area. The size
   * policy must be set at the time THIS(MotifList) widget is
   * created, i.e. from a resource file, or in
   * fallbackResources. It cannot be changed after creation.
   *)
  CharResource(# resourceName:< XmNlistSizePolicy #);
scrollBarDisplayPolicy:
  (* Controls the display of vertical ScrollBars in a
   * ScrolledList. When the value of this resource is
   * XmAS_NEEDED, a vertical ScrollBar is displayed only when
   * the number of items in THIS(MotifList) exceeds the number
   * of Visible items. When the value is XmSTATIC, a vertical
   * ScrollBar is always displayed.
   *)
  CharResource(# resourceName:< XmNscrollBarDisplayPolicy #);
scrollBarPlacement:
  (* Specifies the positioning of the ScrollBars in relation
   * to the visible items. The following are the values:
   *
   * + XmTOP_LEFT - The horizontal ScrollBar is placed above
   *   the visible items, and the vertical ScrollBar to the left
   *   of the visible items.
   *
   * + XmBOTTOM_LEFT - The horizontal ScrollBar is placed
   *   below the visible items, and the vertical ScrollBar to
   *   the left of the visible items.
   *
   * + XmTOP_RIGHT - The horizontal ScrollBar is placed above
   *   the visible items, and the vertical ScrollBar to the
   *   right of the visible items.
   *
   * + XmBOTTOM_RIGHT - The horizontal ScrollBar is placed
   *   below the visible items, and the vertical ScrollBar to
   *   the right of the visible items.
   *)
  IntegerResource(# resourceName:< XmNscrollBarPlacement #);
scrolledWindowMarginWidth:
  (* Specifies the margin width on the right and left sides
   * of the ScrolledWindow.
   *)
  IntegerResource
    (# resourceName:< XmNscrolledWindowMarginWidth #);
scrolledWindowMarginHeight:
  (* Specifies the margin height on the top and bottom of the
   * ScrolledWindow.
   *)
  IntegerResource
    (# resourceName:< XmNscrolledWindowMarginHeight #);

```



```

spacing:
  (* Specifies the distance that separates the ScrollBars
   * from the visible items.
   *)
  IntegerResource(# resourceName::< XmNspacing #);
verticalScrollBar:
  (* The horizontal Scrollbar. *)
  IntegerResource(# resourceName::< XmNverticalScrollBar #);

(* Utility patterns *)
getScrolledWindow:
  (* Used to obtain the ScrolledWindow widget automatically
   * created as surrounding the list widget of
   * THIS(ScrolledList). This can be used to manipulate
   * resources belonging to the ScrolledWindow, e.g. geometric
   * resources.
   *)
  (# scrolled: ^ScrolledWindow;
   do &ScrolledWindow[] -> scrolled[];
   theWidget -> XtParent -> scrolled.thewidget;
   exit scrolled[]
  #);

<<SLOT ScrolledListLib: attributes>>
#) (* ScrolledList *);

(* Aliases used in CompositeLib *)
mList: MotifList(# #);
mScrolledList: ScrolledList(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widgets within composites making the composite
 * the default father of the widgets
 *)
MotifList: mList
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
   do INNER
   #)
  #);
ScrolledList: mScrolledList
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
   do INNER
   #)
  #)
#)

```

---

Lists Interface

[Mjølner Informatics](#)



# Texts Interface

```
ORIGIN 'primitive';
BODY 'private/textsbody';
INCLUDE 'scrolledwindow';
```

```
( *
 * COPYRIGHT
 *      Copyright Mjølner Informatics, 1992-97
 *      All rights reserved.
 * )
```

```
-- XtEnvLib: attributes --
```

**MotifText:** Primitive

```
( * MotifText provides a single-line and multiline text editor
 * for customizing both user and programmatic interfaces. It can
 * be used for single-line string entry, forms entry with
 * verification procedures, and full-window editing. It provides
 * an application with a consistent editing system for textual
 * data. The screen's textual data adjusts to the application
 * writer's needs. MotifText provides separate callbacks to
 * verify movement of the insert cursor, modification of the
 * text, and changes in input focus. Each of these callbacks
 * provides the verification function with the widget instance,
 * the event that caused the callback, and a data structure
 * specific to the verification type. From this information the
 * function can verify if the application considers this to be a
 * legitimate state change and can signal THIS(MotifText) whether
 * to continue with the action.
 * )

(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmTextWidgetClass->value if)
        #)
    do INNER #);

    (* Resources *)
source:
    (* Specifies the source with which THIS(MotifText) displays
    * text. If no source is specified, the widget creates a
    * default string source. This resource can be used to
    * share text sources between MotifText widgets.
    * )
    IntegerResource(# resourceName::< XmNsource #);
value:
    (* The string value of THIS(MotifText) *)
    StringResource(# resourceName::< XmNvalue #);
maxLength:
    (* Specifies the maximum length of the text string that can
    * be entered into THIS(MotifText) from the keyboard.
    * Strings that are entered using the value resource ignore
    * this resource.
    * )
    IntegerResource(# resourceName::< XmNmaxLength #);
marginHeight:
    (* Specifies the distance between the top edge of the
    * widget window and the text, and between the bottom edge
    * of the widget window and the text. This resource is
    * forced to True when the THIS(MotifText) widget is placed
    * in a ScrolledWindow with scrollingPolicy set to
    * XmAUTOMATIC.
```



```

*)
ShortResource(# resourceName:< XmNmarginHeight #);
marginWidth:
(* Specifies the distance between the left edge of the
 * widget window and the text, and between the right edge of
 * the widget window and the text. This resource is forced
 * to True when the THIS(MotifText) widget is placed in a
 * ScrolledWindow with scrollingPolicy set to XmAUTOMATIC.
 *)
ShortResource(# resourceName:< XmNmarginWidth #);
topCharacter:
(* Displays the position of text at the top of the window.
 * Position is determined by the number of characters from
 * the beginning of the text. The first character position
 * is 0
 *)
IntegerResource(# resourceName:< XmNtopCharacter #);
cursorPosition:
(* Indicates the position in the text where the current
 * insert cursor is to be located. Position is determined
 * by the number of characters from the beginning of the
 * text. The first character position is 0.
 *)
IntegerResource(# resourceName:< XmNcursorPosition #);
editMode:
(* Specifies the set of keyboard bindings used in
 * THIS(MotifText). The default keyboard bindings
 * (XmSINGLE_LINE_EDIT) provides the set of key bindings to
 * be used in editing single-line text. The multiline
 * bindings (XmMULTI_LINE_EDIT) provides the set of key
 * bindings to be used in editing multiline text.
 *)
IntegerResource(# resourceName:< XmNeditMode #);
autoShowCursorPosition:
(* Ensures that the visible text contains the insert cursor
 * when set to True. If the insert cursor changes, the
 * contents of THIS(MotifText) may scroll in order to bring
 * the insertion point into the window.
 *)
BooleanResource(# resourceName:< XmNautoShowCursorPosition #);
editable:
(* Indicates that the user can edit the text string when
 * set to True. Prohibits the user from editing the text
 * when set to False.
 *)
BooleanResource(# resourceName:< XmNeditable #);
verifyBell:
(* Specifies whether the bell should sound when the
 * verification returns without continuing the action. The
 * default is True, indicating that the bell should sound.
 *)
BooleanResource(# resourceName:< XmNverifyBell #);
pendingDelete:
(* Indicates that pending delete mode is on when set to
 * True. Pending deletion is defined as deletion of the
 * selected text when an insertion is made.
 *)
BooleanResource(# resourceName:< XmNpendingDelete #);
selectionArray:
(* Defines the actions for multiple mouse clicks. The
 * value of the resource is an array of XmTextScanType
 * elements. XmTextScanType is an enumeration indicating
 * possible actions. Each mouse click performed within half
 * a second of the previous mouse click increments the index
 * into this array and performs the defined action for that
 * index. The possible actions in the order they occur in

```



```

* the default array are:
*
* + XmSELECT_POSITION - resets the insert cursor position
*
* + XmSELECT_WORD - selects a word
*
* + XmSELECT_LINE - selects a line of text
*
* + XmSELECT_ALL - selects all of the text
*)
IntegerResource(# resourceName:< XmNselectionArray #);
selectionArrayCount:
(* Indicates the number of elements in the
 * XmNselectionArray resource.
 *)
IntegerResource(# resourceName:< XmNselectionArrayCount #);
selectThreshold:
(* Specifies the number of pixels of motion that is
 * required to select the next character when selection is
 * performed using the click-drag mode of selection.
 *)
IntegerResource(# resourceName:< XmNselectThreshold #);
fontList:
(* Specifies the font list to be used for THIS(MotifText).
 * If this value is unspecified at initialization, it is
 * initialized by looking up the parent hierarchy of the
 * widget for an ancestor that is a specialization of the
 * BulletinBoard, VendorShell, or MenuShell. If such an
 * ancestor is found, the font list is initialized to the
 * appropriatedefault font list of the ancestor widget
 * (defaultFontList for VendorShell and MenuShell,
 * textFontList for BulletinBoard).
 *)
IntegerResource(# resourceName:< XmNfontList #);
wordWrap:
(* Indicates that lines are to be broken at word breaks
 * (that is, the text does not go off the right edge of the
 * window) when set to True. Words are defined as a
 * sequence of characters separated by white space. White
 * space is defined as a space, tab, or newline. This
 * attribute is ignored if editMode is XmSINGLE_LINE_EDIT.
 *)
BooleanResource(# resourceName:< XmNwordWrap #);
blinkRate:
(* Specifies the blink rate of the text cursor in
 * milliseconds. The time indicated in the blink rate
 * relates to the time the cursor is visible and the time
 * the cursor is invisible (that is, the time it takes to
 * blink the insertion cursor on and off is twice the blink
 * rate). The cursor does not blink when the blink rate is
 * set to zero.
 *)
IntegerResource(# resourceName:< XmNblinkRate #);
columns:
(* Specifies the initial width of the text window measured
 * in character spaces. The default value depends on the
 * value of the width resource.
 *)
ShortResource(# resourceName:< XmNcolumns #);
rows:
(* Specifies the initial height of the text window measured
 * in character heights. This attribute is ignored if
 * editMode is XmSINGLE_LINE_EDIT. The default value
 * depends on the value of the height resource.
 *)
ShortResource(# resourceName:< XmNrows #);

```



```

resizeWidth:
    (* Indicates that THIS(MotifText) attempts to resize its
     * width to accommodate all the text contained in the widget
     * when set to True. This attribute is ignored if wordWrap
     * is True.
     *)
    BooleanResource(# resourceName:< XmNresizeWidth #);
resizeHeight:
    (* Indicates that THIS(MotifText) attempts to resize its
     * height to accommodate all the text contained in the
     * widget when set to True. If set to True, the text is
     * always displayed starting from the first position in the
     * source, even if instructed otherwise. This attribute is
     * ignored when the application uses a ScrolledText widget
     * and when scrollVertical is True.
     *)
    BooleanResource(# resourceName:< XmNresizeHeight #);
cursorPositionVisible:
    (* Indicates that the insert cursor position is marked by a
     * blinking text cursor when set to True.
     *)
    BooleanResource(# resourceName:< XmNcursorPositionVisible #);
insertionPointVisible: cursorPositionVisible(##);

(* Utility patterns *)
lastPosition: IntegerValue
    (* Exits the position of the last character in the text
     * buffer of THIS(MotifText)
     *)
    (# do ... #);
replace:
    (* Replaces part of the text string in THIS(MotifText).
     * The character positions begin at zero and are numbered
     * sequentially from the beginning of the text.
     *)
    (# from, to: @integer;
     string: ^text;
     enter (from, to, string[])
     do ...
     #);
insert:
    (* Inserts a character string into the text string in
     * THIS(MotifText). The character positions begin at zero
     * and are numbered sequentially from the beginning of the
     * text.
     *)
    (# position: @integer;
     string: ^text;
     enter (position, string[])
     do ...
     #);
remove:
    (* Deletes the primary selected text, and calls
     * modifyVerifyCallback and valueChangedCallback if there is
     * a selection.
     *)
    (# do ... #);
copy:
    (* Copies the primary selected text to the clipboard. The
     * time parameter specifies the time at which the selection
     * value is to be modified. This should be the time of the
     * event which triggered this request.
     *)
    (# time: @integer;
     enter time
     do ... #);

```



```

cut:
(* Copies the primary selected text to the clipboard and
 * then deletes the primary selected text. This routine
 * also calls modifyVerifyCallback and valueChangedCallback
 * if there is a selection. The time parameter specifies
 * the time at which the selection value is to be modified.
 * This should be the time of the event which triggered this
 * request.
 *)
(# time: @integer;
 enter time
 do ... #);

paste:
(* Inserts the clipboard selection at the destination
 * cursor. If pendingDelete is True and the destination
 * cursor is inside the current selection, the clipboard
 * selection replaces the selected text. This routine also
 * calls modifyVerifyCallback and valueChangedCallback if
 * there is a clipboard selection.
 *)
(# do ... #);

selection: @
(#
  position:
  (* Exits the left and right position of the primary
   * selection in the text buffer of THIS(MotifText).
   *)
  (# left, right: @integer;
   primary: @boolean
   (* true iff THIS(MotifText) owns the primary
    * selection
    *)
   do ...
   exit (left, right, primary)
  #);

  get:
  (* Retrieves the value of the primary selection of
   * THIS(MotifText).
   *)
  (# selection: ^text;
   do ...
   exit selection[]
  #);

  set:
  (* Sets the primary selection of the text in
   * THIS(MotifText). Also sets the insertion cursor
   * position to the last position of the selection and
   * calls motionVerifyCallback
   *)
  (# first, last: @integer
   (* First and last text position to be selected *)
   time: @integer
   (* Specifies the time at which the selection
    * value is desired. This should be the time of
    * the event which triggered this request.
    *)
   enter (first, last, time)
   do ...
  #);

  clear:
  (* Clears the primary selection of THIS(MotifText) *)
  (# time: @integer
   (* Specifies the time at which the selection
    * value is desired. This should be the time of
    * the event which triggered this request.
    *)

```



```

        enter time
        do ...
        #);
enter set
exit get
#);
xyToPos: IntegerValue
(* Exits the character position nearest to the specified x
 * and y position, relative to the upper left corner of
 * THIS(MotifText)
 *)
(# x, y: @integer;
enter (x, y)
do ...
#);
posToXY:
(* Exits the x and y position, relative to the upper left
 * corner of THIS(MotifText), of a given character position
 * in the text buffer. If the character position is not
 * displayed in THIS(MotifText), x and y will both be zero
 * after conversion. The first character position is 0.
 *)
(# pos, x, y: @integer;
enter pos
do ...
exit (x, y)
#);
showPosition:
(* Forces text at the specified position to be displayed.
 * If the autoShowCursorPosition resource is True, the
 * application should also set the insert cursor to this
 * position. The first character position is 0.
 *)
(# position: @integer;
enter position
do ...
#);
setHighlight:
(* Highlights text between the two specified character
 * positions. The mode parameter determines the type of
 * highlighting. Highlighting text merely changes the
 * visual appearance of the text; it does not set the
 * selection.
 *)
(# left, right, mode: @integer;
enter (left, right, mode)
do ...
#);
scroll:
(* Scrolls the text in THIS(MotifText). The lines argument
 * specifies the number of lines of text to scroll. A
 * positive value causes text to scroll upward; a negative
 * value causes text to scroll downward.
 *)
(# lines: @integer;
enter lines
do ...;
#);
getBaseLine: IntegerValue
(* Exits the y position of the first baseline in
 * THIS(MotifText), relative to the y position of the top of
 * THIS(MotifText).
 *)
(# do ... #);
insertFile:
(* Inserts the contents of the specified file into

```



```

    * THIS(MotifText) at the specified position.
    *)
    (# position: @integer;
     filename: ^text;
     error:< Exception
       (* Called if the file cannot be read *)
       (# reason: ^text;
        enter reason[]
        do ...;
         INNER;
        #);
     enter (position, filename[])
     do ...
     #);
saveToFile:
    (* Saves the contents of THIS(MotifText) into the specified
    * file
    *)
    (# filename: ^text;
     error:< Exception
       (* Called if the file cannot be written *)
       (# reason: ^text;
        enter reason[]
        do ...;
         INNER;
        #);
     enter filename[]
     do ...
     #);

    (* Callbacks *)
MotifTextCallback: MotifCallback
    (* Prefix for MotifText callbacks *)
    (# do INNER #);
TextVerifyCallback: MotifCallback
    (* Prefix for verification callbacks *)
    (# callData:< XmTextVerifyCallbackStruct do INNER #);

activateCallback:< MotifTextCallback
    (* Called when the user invokes an event that calls the
    * Activate() function. The reason sent by the callback is
    * XmCR_ACTIVATE.
    *);
gainPrimaryCallback:< MotifTextCallback
    (* Called when an event causes the THIS(MotifText) to gain
    * ownership of the primary selection. The reason sent by
    * the callback is XmCR_GAIN_PRIMARY.
    *);
losePrimaryCallback:< MotifTextCallback
    (* Called when an event causes the THIS(MotifText) to lose
    * ownership of the primary selection. The reason sent by
    * the callback is XmCR_LOSE_PRIMARY.
    *);
losingFocusCallback:< TextVerifyCallback
    (* Called before THIS(MotifText) loses input focus. The
    * reason sent by the callback is XmCR_LOSING_FOCUS.
    *);
modifyVerifyCallback:< TextVerifyCallback
    (* Called before text is deleted from or inserted into
    * THIS(MotifText). The reason sent by the callback is
    * XmCR_MODIFYING_TEXT_VALUE.
    *);
motionVerifyCallback:< TextVerifyCallback
    (* Called before the insert cursor is moved to a new
    * position. The reason sent by the callback is
    * XmCR_MOVING_INSERT_CURSOR.

```



```

    *);
valueChangedCallback:< MotifTextCallback
    (* Called after text is deleted from or inserted into
    * THIS(MotifText). The reason sent by the callback is
    * XmCR_VALUE_CHANGED.
    *);
    (* Inherited callbacks *)
helpCallback:< MotifTextCallback;

installCallbacks:< (* Private *)
    (# do ...; INNER #);

    <<SLOT MotifTextLib: attributes>>
#) (* MotifText *);

ScrolledText: MotifText
    (* A ScrolledText is a MotifText that is contained within a
    * ScrolledWindow. All ScrolledWindow subarea widgets are
    * automatically created. The ScrolledText defaults to multi-line
    * text edit.
    *)
    (# init:< (# WidgetClass:< (# do ... #);
        do INNER
        #);

    (* ScrolledText resources. These can only be set at creation
    * time, e.g., from fallbackResources, or a resource file.
    *)
    scrollVertical:
        (* Adds a ScrollBar that allows the user to scroll
        * vertically through text when set to True. This resource
        * is forced to False when THIS(ScrolledText) is placed in a
        * ScrolledWindow with scrollingPolicy set to XmAUTOMATIC.
        *)
        BooleanResource(# resourceName:< XmNscrollVertical #);
    scrollHorizontal:
        (* Adds a ScrollBar that allows the user to scroll
        * horizontally through text when set to True. This
        * attribute is ignored if the MotifText resource editMode
        * is XmSINGLE_LINE_EDIT. This resource is forced to False
        * when THIS(MotifText) is placed in a ScrolledWindow with
        * scrollingPolicy set to XmAUTOMATIC.
        *)
        BooleanResource(# resourceName:< XmNscrollHorizontal #);
    scrollLeftside:
        (* Indicates that the vertical ScrollBar should be placed
        * on the left side of the scrolled text window when set to
        * True. This attribute is ignored if scrollVertical is
        * False or the MotifText resource editMode is
        * XmSINGLE_LINE_EDIT. The default value may depend on the
        * value of the stringDirection resource.
        *)
        BooleanResource(# resourceName:< XmNscrollLeftside #);
    scrollTopSide:
        (* Indicates that the horizontal ScrollBar should be placed
        * on the top side of the scrolled text window when set to
        * True.
        *)
        BooleanResource(# resourceName:< XmNscrollTopSide #);

    (* Utility patterns *)
    getScrolledWindow:
        (* Used to obtain the ScrolledWindow widget automatically
        * created as surrounding the text widget of
        * THIS(ScrolledText). This can be used to manipulate
        * resources belonging to the ScrolledWindow, e.g. geometric

```



```

    * resources.
    *)
    (# scrolled: ^ScrolledWindow;
    do &ScrolledWindow[] -> scrolled[];
        theWidget -> XtParent -> scrolled.thewidget;
    exit scrolled[]
    #);

<<SLOT ScrolledTextLib: attributes>>
#) (* ScrolledText *);

```

**TextField: Primitive**

```

(* TextField provides a single line text editor for customizing
 * both user and programmatic interfaces. It is used for
 * single-line string entry, and forms entry with verification
 * procedures. It provides an application with a consistent
 * editing system for textual data. TextField provides separate
 * callbacks to verify movement of the insert cursor,
 * modification of the text, and changes in input focus. Each of
 * these callbacks provides the verification function with the
 * widget instance, the event that caused the callback, and a
 * data structure specific to the verification type. From this
 * information the function can verify if the application
 * considers this to be a legitimate state change and can signal
 * the widget whether to continue with the action. TextField
 * allows the user to select regions of text. Selection is based
 * on the Interclient Communication Conventions (ICCC) selection
 * model. TextField supports both primary and secondary
 * selection.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
                (if value=0 then xmTextFieldWidgetClass->value if)
            #)
        do INNER
    #);

    (* Resources *)
blinkRate:
    (* Specifies the blink rate of the text cursor in
     * milliseconds. The time indicated in the blink rate
     * relates to the length of time the cursor is visible and
     * the time the cursor is invisible (i.e., the time it will
     * take to blink the insertion cursor on and off will be 2
     * times the blink rate). The cursor will not blink when
     * the blink rate is set to zero.
     *)
    IntegerResource(# resourceName::< XmNblinkRate #);
columns:
    (* Specifies the initial width of THIS(TextField) measured
     * in character spaces. The default value depends on the
     * value of the width resource.
     *)
    ShortResource(# resourceName::< XmNcolumns #);
cursorPosition:
    (* Indicates the position in the text where the current
     * insert cursor is (to be) located. The position is
     * determined by the number of characters from the beginning
     * of the text.
     *)
    IntegerResource(# resourceName::< XmNcursorPosition #);
cursorPositionVisible:
    (* Indicates that the insert cursor position is marked by a
     * blinking text cursor when the Boolean is True.

```



```

*)
BooleanResource(# resourceName:< XmNcursorPositionVisible #);
editable:
(* Indicates that the user can edit the text string when
 * set to True. A false value will prohibit the user from
 * editing the text.
 *)
BooleanResource(# resourceName:< XmNeditable #);
fontList:
(* Specifies the font list to be used for THIS(TextField).
 * If this value not specified at initialization, it is
 * initialized by looking up the parent hierarchy of the
 * widget for an ancestor that is a specialization of
 * BulletinBoard, VendorShell, or MenuShell. If such an
 * ancestor is found, the font list is initialized to the
 * appropriate default font list of the ancestor widget
 * (defaultFontList for VendorShell and MenuShell,
 * textFontList for BulletinBoard).
 *)
IntegerResource(# resourceName:< XmNfontList #);
marginHeight:
(* Specifies the distance between the top edge of the
 * widget window and the text, and the bottom edge of the
 * widget window and the text.
 *)
ShortResource(# resourceName:< XmNmarginHeight #);
marginWidth:
(* Specifies the distance between the left edge of the
 * widget window and the text, and the right edge of the
 * widget window and the text.
 *)
ShortResource(# resourceName:< XmNmarginWidth #);
maxLength:
(* Specifies the maximum length of the text string that can
 * be entered into text from the keyboard. Strings that are
 * entered using the value resource are truncated by this
 * resource.
 *)
IntegerResource(# resourceName:< XmNmaxLength #);
pendingDelete:
(* Indicates that pending delete mode is on when set to
 * True. Pending deletion is defined as deletion of the
 * selected text when an insertion is made.
 *)
BooleanResource(# resourceName:< XmNpendingDelete #);
resizeWidth:
(* Indicates that THIS(TextField) will attempt to resize
 * its width to accommodate all the text contained in the
 * widget when set to True.
 *)
BooleanResource(# resourceName:< XmNresizeWidth #);
selectionArray:
(* Defines the actions for multiple-mouse clicks. Each
 * mouse click performed within a half of a second of the
 * previous mouse click will increment the index into this
 * array and perform the defined action for that index. The
 * possible actions are:
 *
 * + XmSELECT_POSITIONS - resets the insert cursor
 *   position.
 *
 * + XmSELECT_WORD - selects a word.
 *
 * + XmSELECT_LINE - selects a line of text.
 *)
IntegerResource(# resourceName:< XmNselectionArray #);

```



```

selectionArrayCount:
    (* Specifies the number of actions that are defined in the
     * selectionArray resource. This resource must be reset
     * when the number of actions in the selection array
     * changes.
     *)
    IntegerResource(# resourceName::< XmNselectionArrayCount #);
selectThreshold:
    (* Specifies the number of pixels of motion that is
     * required to select the next character when selection is
     * performed using the click-drag mode of selection.
     *)
    IntegerResource(# resourceName::< XmNselectThreshold #);
value:
    (* The displayed string value. *)
    StringResource(# resourceName::< XmNvalue #);
verifyBell:
    (* Specifies whether a bell will sound when an action is
     * reversed during a verification callback.
     *)
    BooleanResource(# resourceName::< XmNverifyBell #);

(* Utility patterns *)
lastPosition: IntegerValue
    (* Exits the position of the last character in the text
     * buffer of THIS(TextField)
     *)
    (# do ... #);
replace:
    (* Replaces part of the text string in THIS(TextField).
     * The character positions begin at zero and are numbered
     * sequentially from the beginning of the text.
     *)
    (# from, to: @integer;
     string: ^text;
     enter (from, to, string[])
     do ...
     #);
insert:
    (* Inserts a character string into the text string in
     * THIS(TextField). The character positions begin at zero
     * and are numbered sequentially from the beginning of the
     * text.
     *)
    (# position: @integer;
     string: ^text;
     enter (position, string[])
     do ...
     #);
remove:
    (* Deletes the primary selected text, and calls
     * modifyVerifyCallback and valueChangedCallback if there is
     * a selection.
     *)
    (# do ... #);
copy:
    (* Copies the primary selected text to the clipboard. The
     * time parameter specifies the time at which the selection
     * value is to be modified. This should be the time of the
     * event which triggered this request.
     *)
    (# time: @integer;
     enter time
     do ... #);
cut:
    (* Copies the primary selected text to the clipboard and

```



```

* then deletes the primary selected text. This routine
* also calls modifyVerifyCallback and valueChangedCallback
* if there is a selection. The time parameter specifies
* the time at which the selection value is to be modified.
* This should be the time of the event which triggered this
* request.
*)
(# time: @integer;
enter time
do ... #);

paste:
(* Inserts the clipboard selection at the destination
* cursor. If pendingDelete is True and the destination
* cursor is inside the current selection, the clipboard
* selection replaces the selected text. This routine also
* calls modifyVerifyCallback and valueChangedCallback if
* there is a clipboard selection.
*)
(# do ... #);

selection: @
(#
position:
(* Exits the left and right position of the primary
* selection in the text buffer of THIS(TextField).
*)
(# left, right: @integer;
primary: @boolean
(* true iff THIS(TextField) owns the primary
* selection
*);
do ...
exit (left, right, primary)
#);

get:
(* Retrieves the value of the primary selection of
* THIS(TextField).
*)
(# selection: ^text;
do ...
exit selection[]
#);

set:
(* Sets the primary selection of the text in
* THIS(TextField). Also sets the insertion cursor
* position to the last position of the selection and
* calls motionVerifyCallback
*)
(# first, last: @integer
(* First and last text position to be selected *);
time: @integer
(* Specifies the time at which the selection
* value is desired. This should be the time of
* the event which triggered this request.
*)
enter (first, last, time)
do ...
#);

clear:
(* Clears the primary selection of THIS(TextField) *)
(# time: @integer
(* Specifies the time at which the selection
* value is desired. This should be the time of
* the event which triggered this request.
*);
enter time
do ...

```



```

        #);
enter set
exit get
#);
xyToPos: IntegerValue
(* Exits the character position nearest to the specified x
 * and y position, relative to the upper left corner of
 * THIS(TextField)
 *)
(# x, y: @integer;
enter (x, y)
do ...
#);
posToXY:
(* Exits the x and y position, relative to the upper left
 * corner of THIS(TextField), of a given character position
 * in the text buffer.
 *)
(# pos, x, y: @integer;
enter pos
do ...
exit (x, y)
#);
showPosition:
(* Forces text at the specified position to be displayed.
 * If the autoShowCursorPosition resource is True, the
 * application should also set the insert cursor to this
 * position.
 *)
(# position: @integer;
enter position
do ...
#);
setHighlight:
(* Highlights text between the two specified character
 * positions. The mode parameter determines the type of
 * highlighting. Highlighting text merely changes the
 * visual appearance of the text; it does not set the
 * selection.
 *)
(# left, right, mode: @integer;
enter (left, right, mode)
do ...
#);
getBaseLine: IntegerValue
(* Exits the y position of the first baseline in
 * THIS(TextField), relative to the y position of the top of
 * THIS(TextField).
 *)
(# do ... #);

(* Callbacks *)
TextFieldCallback: MotifCallback
(* Prefix for TextField callbacks *)
(# do INNER #);
TextFieldVerifyCallback: MotifCallback
(* Prefix for verification callbacks *)
(# callData::< XmTextVerifyCallbackStruct do INNER #);

activateCallback: < TextFieldCallback
(* Called when the user invokes an event that calls the
 * Activate() function. The reason sent by the callback is
 * XmCR_ACTIVATE.
 *)
focusCallback: < TextFieldCallback
(* Called before THIS(TextField) has accepted input focus.

```



```

    * The reason sent by the callback is XmCR_FOCUS.
    *);

gainPrimaryCallback:< TextFieldCallback
    (* Called when an event causes the THIS(TextField) to gain
    * ownership of the primary selection. The reason sent by
    * the callback is XmCR_GAIN_PRIMARY.
    *);

losePrimaryCallback:< TextFieldCallback
    (* Called when an event causes the THIS(TextField) to lose
    * ownership of the primary selection. The reason sent by
    * the callback is XmCR_LOSE_PRIMARY.
    *);

losingFocusCallback:< TextFieldVerifyCallback
    (* Called before THIS(TextField) loses input focus. The
    * reason sent by the callback is XmCR_LOSING_FOCUS.
    *);

modifyVerifyCallback:< TextFieldVerifyCallback
    (* Called before text is deleted from or inserted into
    * THIS(TextField). The reason sent by the callback is
    * XmCR_MODIFYING_TEXT_VALUE.
    *);

motionVerifyCallback:< TextFieldVerifyCallback
    (* Called before the insert cursor is moved to a new
    * position. The reason sent by the callback is
    * XmCR_MOVING_INSERT_CURSOR.
    *);

valueChangedCallback:< TextFieldCallback
    (* Called after text is deleted from or inserted into
    * THIS(TextField). The reason sent by the callback is
    * XmCR_VALUE_CHANGED.
    *);

    (* Inherited callbacks *)
helpCallback:< TextFieldCallback;

installCallbacks:< (* Private *)
    (# do ...; INNER #);

    <<SLOT TextFieldLib: attributes>>;
#);

(* Aliases used in CompositeLib *)
mText: MotifText(# #);
mScrolledText: ScrolledText(# #);
mTextField: TextField(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widgets within composites making the composite
 * the default father of the widgets
 *)
MotifText: mText
    (# init:< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #);

ScrolledText: mScrolledText
    (# init:< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #);

TextField: mTextField
    (# init:< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #)

```



Texts Interface

[Mjølner Informatics](#)



# Manager Interface

```
ORIGIN 'basics';
BODY 'private/managerbody';
```

```
(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)
```

```
-- XtEnvLib: attributes --
```

**Manager:** Constraint

```
(* Manager is a pattern used as a supporting superpattern for
 * other widget patterns. It supports the visual resources,
 * graphics contexts, and traversal resources necessary for the
 * graphics and traversal mechanisms.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmManagerWidgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
foreground:
    (* Specifies the foreground drawing color used by
     * THIS(Manager)
     *)
    IntegerResource(# resourceName::< XmNforeground #);
highlightColor:
    (* Specifies the color of the highlighting rectangle. This
     * color is used if the highlight pixmap resource is
     * XmUNSPECIFIED_PIXMAP.
     *)
    IntegerResource(# resourceName::< XmNhighlightColor #);
highlightPixmap:
    (* Specifies the pixmap used to draw the highlighting
     * rectangle.
     *)
    IntegerResource(# resourceName::< XmNhighlightPixmap #);
unitType:
    (* Provides the basic support for resolution independence.
     * It defines the type of units THIS(Manager) uses with
     * sizing and positioning resources. If THIS(Manager)'s
     * parent is a specialization of Manager and if the unitType
     * resource is not explicitly set, it defaults to the unit
     * type of the parent widget. If THIS(Manager)'s parent is
     * not a specialization of Manager, the resource has a
     * default unit type of XmPIXELS unitType can have the
     * following values:
     *
     * + XmPIXELS - all values provided to THIS(Manager) are
     *   treated as normal pixel values.
     *
     * + Xm100TH_MILLIMETERS - all values provided to
     *   THIS(Manager) are treated as 1/100 millimeter.
     *
     * + Xm1000TH_INCHES - all values provided to THIS(Manager)
```



```

*   are treated as 1/1000 inch.
*
*   + Xm100TH_POINTS - all values provided to THIS(Manager)
*   are treated as 1/100 point. A point is a unit used in
*   text processing applications and is defined as 1/72 inch.
*
*   + Xm100TH_FONT_UNITS - all values provided to the widget
*   are treated as 1/100 of a font unit. See the Motif
*   documentation for details on using font units.
*)
CharResource(# resourceName:< XmNunitType #);
shadowThickness:
(* Specifies the thickness of the drawn border shadow.
* BulletinBoard and its descendants set this value
* dynamically. If the widget is a top level window, this
* value is set to 1. If it is not a top level window, this
* value is set to 0.
*)
ShortResource(# resourceName:< XmNshadowThickness #);
topShadowColor:
(* Specifies the color to use to draw the top and left
* sides of the border shadow. This color is used if the
* topShadowPixmap resource is 0.
*)
IntegerResource(# resourceName:< XmNtopShadowColor #);
topShadowPixmap:
(* Specifies the pixmap to use to draw the top and left
* sides of the border shadow.
*)
IntegerResource(# resourceName:< XmNtopShadowPixmap #);
bottomShadowColor:
(* Specifies the color to use to draw the bottom and right
* sides of the border shadow. This color is used if the
* bottomShadowPixmap resource is 0.
*)
IntegerResource(# resourceName:< XmNbottomShadowColor #);
bottomShadowPixmap:
(* Specifies the pixmap to use to draw the bottom and right
* sides of the border shadow.
*)
IntegerResource(# resourceName:< XmNbottomShadowPixmap #);
userData:
(* Allows the application to attach any necessary specific
* data to THIS(Manager). This is an internally unused
* resource.
*)
IntegerResource(# resourceName:< XmNuserData #);
navigationType:
(* Controls whether the Widget is a navigation group.
*
*   + XmNONE indicates that the Widget is not a navigation
*   group.
*
*   + XmTAB_GROUP indicates that the Widget is included
*   automatically in keyboard navigation, unless
*   XmAddTabGroup has been called.
*
*   + XmSTICKY_TAB_GROUP indicates that the Widget is
*   included automatically in keyboard navigation, even if
*   XmAddTabGroup has been called.
*
*   + XmEXCLUSIVE_TAB_GROUP indicates that the Widget is
*   included explicitly in keyboard navigation by the
*   application. With XmEXCLUSIVE_TAB_GROUP, traversal of
*   widgets within the group is based on the order of
*   children.

```



```

    *)
    IntegerResource(# resourceName:<< XmNnavigationType #);
stringDirection:
    (* Specifies the initial direction to draw strings. The
    * values are XmSTRING_DIRECTION_L_TO_R and
    * XmSTRING_DIRECTION_R_TO_L. The value of this resource is
    * determined at creation time. If THIS(Manager)'s parent
    * is a Manager, this value is inherited from
    * THIS(Manager)'s parent, otherwise it is set to
    * XmSTRING_DIRECTION_L_TO_R.
    *)
    IntegerResource(# resourceName:<< XmNstringDirection #);
traversalOn:
    (* Specifies traversal activation for THIS(Manager). *)
    BooleanResource(# resourceName:<< XmNtraversalOn #);

    (* Callbacks. Only the helpCallback pattern is actually used
    * in Manager, the rest is used in various
    * specializations. But since many specializations define
    * callbacks with identical names, for convenience, most of
    * the corresponding patterns are declared here.
    *)
helpCallback:< CallbackProc
    (* Called when the help key sequence is pressed. The
    * reason sent by this callback is XmCR_HELP. No
    * translation is bound to this resource. It is up to the
    * application to install a translation for help.
    *);

okCallback:< CallbackProc;
cancelCallback:< CallbackProc;
applyCallback:< CallbackProc;
exposeCallback:< CallbackProc;
inputCallback:< CallbackProc;
resizeCallback:< CallbackProc;
entryCallback:< CallbackProc;
focusCallback:< CallbackProc;
mapCallback:< CallbackProc;
unmapCallback:< CallbackProc;
valueChangedCallback:< CallbackProc;
dragCallback:< CallbackProc;

installCallbacks:<< (* Private *)
    (# do ...; INNER; #);

    <<SLOT ManagerLib: attributes>>
    #) (* Manager *);

    (* Alias used in CompositeLib *)
mManager: Manager(# #);

    --- CompositeLib: attributes ---

    (* Redefinition of widget within composites making the composite
    * the default father of the widget
    *)
Manager: mManager
    (# init:<< (# GetFatherWidget:<< (# do THIS(Composite)->value #);
        do INNER
        #)
    #)
    #)

```

---







# Bulletinboard Interface

```
ORIGIN 'manager';
BODY 'private/bulletinboardbody';

(*
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

BulletinBoard: Manager
(* A BulletinBoard is a composite widget that provides simple
 * geometry management for children widgets. It does not force
 * positioning on its children, but can be set to reject geometry
 * requests that result in overlapping children. BulletinBoard is
 * the base widget for most dialog widgets and is also used as a
 * general container widget. Modal and modeless dialogs are
 * implemented as collections of widgets that include a
 * DialogShell, a BulletinBoard (or subpattern) child of the
 * shell, and various dialog components (buttons, labels, etc.)
 * that are children of BulletinBoard. BulletinBoard defines
 * callbacks useful for dialogs (focus, map, unmap), which are
 * available for application use. If its parent is a DialogShell,
 * BulletinBoard passes title and input mode (based on dialog
 * style) information to the parent, which is responsible for
 * appropriate communication with the window manager.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmBulletinBoardWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
    shadowType:
    (* Describes the shadow drawing style for
     * THIS(BulletinBoard). This resource can have the following
     * values:
     *
     * + XmSHADOW_IN - draws THIS(BulletinBoard) shadow so that
     *   it appears inset. This means that the bottom shadow
     *   visuals and top shadow visuals are reversed.
     *
     * + XmSHADOW_OUT - draws THIS(BulletinBoard) shadow so that
     *   it appears outset
     *
     * + XmSHADOW_ETCHED_IN - draws THIS(BulletinBoard) shadow
     *   using a double line giving the effect of a line etched
     *   into the window, similar to the Separator widget
     *
     * + XmSHADOW_ETCHED_OUT - draws THIS(BulletinBoard) shadow
     *   using a double line giving the effect of a line coming
     *   out of the window, similar to the Separator widget
     *
     * THIS(BulletinBoard) draws shadows just within its borders
     * of if shadowThickness is greater than zero. If the parent
     * of THIS(BulletinBoard) is a DialogShell,
     * THIS(BulletinBoard) dynamically changes the default for
     * shadowThickness from 0 to 1.
```



```

    *)
    CharResource(# resourceName::< XmNshadowType #);
marginWidth:
    (* Specifies the minimum spacing in pixels between the left
    * or right edge of THIS(BulletinBoard) and any child widget.
    *)
    ShortResource(# resourceName::< XmNmarginWidth #);
marginHeight:
    (* Specifies the minimum spacing in pixels between the top
    * or bottom edge of THIS(BulletinBoard) and any child
    * widget.
    *)
    ShortResource(# resourceName::< XmNmarginHeight #);
defaultButton:
    (* Specifies the default button. Some subpatterns of
    * BulletinBoard, which define a default button, set this
    * resource. BulletinBoard defines translations and installs
    * accelerators that activate that button when <Return> is
    * pressed and the keyboard focus is not in another button.
    *)
    IntegerResource(# resourceName::< XmNdefaultButton #);
cancelButton:
    (* Specifies the Cancel button. Subpatterns of
    * BulletinBoard, which define a Cancel button, set this
    * resource. The BulletinBoard pattern does not directly
    * provide any behavior for that button.
    *)
    IntegerResource(# resourceName::< XmNcancelButton #);
buttonFontList:
    (* Specifies the font list used for THIS(BulletinBoard)'s
    * button children (PushButtons, PushButtonGadgets,
    * ToggleButtons, and ToggleButtonGadgets). If this value is
    * 0 at initialization, it is initialized by looking up the
    * parent hierarchy of the widget for an ancestor that is a
    * specialization of BulletinBoard, VendorShell, or
    * MenuShell. If such an ancestor is found, the font list is
    * initialized to the appropriate default font list of the
    * ancestor widget (defaultFontList for VendorShell and
    * MenuShell, buttonFontList for BulletinBoard).
    *)
    IntegerResource(# resourceName::< XmNbuttonFontList #);
labelFontList:
    (* Specifies the font list used for THIS(BulletinBoard)'s
    * Label children (Labels and LabelGadgets). If this value
    * is 0 at initialization, it is initialized by looking up
    * the parent hierarchy of the widget for an ancestor that is
    * a subpattern of BulletinBoard, VendorShell, or MenuShell.
    * If such an ancestor is found, the font list is initialized
    * to the appropriate default font list of the ancestor
    * widget (defaultFontList for VendorShell and MenuShell,
    * labelFontList for BulletinBoard).
    *)
    IntegerResource(# resourceName::< XmNlabelFontList #);
textFontList:
    (* Specifies the font list used for THIS(BulletinBoard)'s
    * Text children. If this value is 0 at initialization, it
    * is initialized by looking up the parent hierarchy of the
    * widget for an ancestor that is a specialization of the
    * BulletinBoard, VendorShell, or MenuShell. If such an
    * ancestor is found, the font list is initialized to the
    * appropriate default font list of the ancestor widget
    * (defaultFontList for VendorShell and MenuShell,
    * textFontList for BulletinBoard).
    *)
    IntegerResource(# resourceName::< XmNtextFontList #);
textTranslations:

```



```

(* Adds translations to any MotifText that is added as a
 * child of THIS(BulletinBoard).
 *)
IntegerResource(# resourceName:< XmNtextTranslations #);
allowOverlap:
(* Controls the policy for overlapping children widgets. If
 * True, THIS(BulletinBoard) allows geometry requests that
 * result in overlapping children.
 *)
BooleanResource(# resourceName:< XmNallowOverlap #);
autoUnmanage:
(* Controls whether or not THIS(BulletinBoard) is
 * automatically unmanaged after a button is activated. If
 * this resource is True on initialization and if
 * THIS(BulletinBoard)'s parent is a DialogShell,
 * THIS(BulletinBoard) adds a callback to button children
 * (PushButtons, PushButtonGadgets, and DrawnButtons) that
 * unmanages THIS(BulletinBoard) when a button is activated.
 * If this resource is False on initialization or if
 * THIS(BulletinBoard)'s parent is not a DialogShell,
 * THIS(BulletinBoard) is not automatically unmanaged. For
 * BulletinBoard subpatterns with Apply or Help buttons,
 * activating those buttons does not automatically unmanage
 * the BulletinBoard.
 *)
BooleanResource(# resourceName:< XmNautoUnmanage #);
defaultPosition:
(* Controls whether or not THIS(BulletinBoard) is
 * automatically positioned by its parent. If True, and the
 * parent of THIS(BulletinBoard) is a DialogShell,
 * THIS(BulletinBoard) is centered within or around the
 * parent of the DialogShell when THIS(BulletinBoard) is
 * mapped and managed. If False, THIS(BulletinBoard) is not
 * automatically positioned.
 *)
BooleanResource(# resourceName:< XmNdefaultPosition #);
resizePolicy:
(* Controls the policy for resizing THIS(BulletinBoard).
 * Possible values include the following:
 *
 * + XmRESIZE_NONE - fixed size
 *
 * + XmRESIZE_ANY - shrink or grow as needed
 *
 * + XmRESIZE_GROW - grow only
 *)
CharResource(# resourceName:< XmNresizePolicy #);
noResize:
(* Controls whether or not resize controls are included in
 * the window manager frame around the dialog. If set to
 * True, the window manager does not include resize controls
 * in the window manager frame containing the DialogShell or
 * TopLevelShell parent of THIS(BulletinBoard). If set to
 * False, the window manager frame does include resize
 * controls.
 *)
BooleanResource(# resourceName:< XmNnoResize #);
dialogStyle:
(* Indicates the dialog style associated with
 * THIS(BulletinBoard). If the parent of THIS(BulletinBoard)
 * is a DialogShell, the parent is configured according to
 * this resource and DialogShell sets the inputMode of
 * VendorShell accordingly. Possible values for this resource
 * include the following:
 *
 * + XmDIALOG_SYSTEM_MODAL - used for dialogs that must be

```



```

*      responded to before any other interaction in any
*      application + XmDIALOG_PRIMARY_APPLICATION_MODAL - used
*      for dialogs that must be responded to before some other
*      interactions in ancestors of the widget +
*      XmDIALOG_APPLICATION_MODAL - used for dialogs that must
*      be responded to before some other interactions in
*      ancestors of the widget. This value is the same as
*      XmDIALOG_PRIMARY_APPLICATION_MODAL, and remains for
*      compatibility.
*
* + XmDIALOG_FULL_APPLICATION_MODAL - used for dialogs that
* must be responded to before some other interactions in
* the same application
*
* + XmDIALOG_MODELESS - used for dialogs that do not
* interrupt interaction of any application. This is the
* default when the parent of THIS(BulletinBoard) is a
* DialogShell.
*
* + XmDIALOG_WORK_AREA - used for BulletinBoard widgets
* whose parents are not DialogShells. This is the default
* when the parent of a BulletinBoard is not a DialogShell.
*)
CharResource(# resourceName:< XmNdialogStyle #);
dialogTitle:
(* Specifies the dialog title. If this resource is not 0,
* and the parent of THIS(BulletinBoard) is a subpattern of
* WMShell, THIS(BulletinBoard) sets the title and
* nameEncoding of its parent. If the only character set in
* dialogTitle is ISO8859-1, title is set to the string of
* the title, and nameEncoding is set to STRING. If
* dialogTitle contains character sets other than ISO8859-1,
* the title is set to the string of the title converted to a
* compound text string, and nameEncoding is set to
* COMPOUND_TEXT.
*)
MotifStringResource(# resourceName:< XmNdialogTitle #);

(* Callbacks *)
focusCallback:< MotifCallback
(* Called when THIS(BulletinBoard) or one of its descendants
* accepts the input focus. The callback reason is
* XmCR_FOCUS.
*);
mapCallback:< MotifCallback
(* Called only when the parent of THIS(BulletinBoard) is a
* DialogShell; in which case, this callback is invoked when
* THIS(BulletinBoard) is mapped. The callback reason is
* XmCR_MAP.
*);
unmapCallback:< MotifCallback
(* Called only when the parent of THIS(BulletinBoard) is a
* DialogShell; in which case, this callback is invoked when
* THIS(BulletinBoard) is unmapped. The callback reason is
* XmCR_UNMAP.
*);
(* Inherited callbacks *)
helpCallback:< MotifCallback;

installCallbacks:< (* Private *)
(# do ...; INNER; #);

<<SLOT BulletinBoardLib: attributes>>
#) (* BulletinBoard *);

```

**BulletinBoardDialog:** BulletinBoard



```

(* A BulletinBoardDialog consists of a DialogShell and an
 * unmanaged BulletinBoard child of the DialogShell. A
 * BulletinBoardDialog is used for interactions not supported by
 * the standard dialog set. It does not include any labels,
 * buttons, or other dialog components. Such components should be
 * added to the BulletinBoardDialog by the application.
 *)
(# init::< (# WidgetClass::< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

    <<SLOT BulletinBoardDialogLib: attributes>>
#) (* BulletinBoardDialog *);

(* Aliases used in CompositeLib *)
mBulletinBoard: BulletinBoard(# #);
mBulletinBoardDialog: BulletinBoardDialog(# #);

--- CompositeLib: attributes ---

(* Redefinition of widgets within composites making the composite
 * the default father of the widgets
 *)
BulletinBoard: mBulletinBoard
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
BulletinBoardDialog: mBulletinBoardDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->Value #);
    do INNER
    #)
  #)
#)

```

---

Bulletinboard Interface

[Mjølner Informatics](#)



# Form Interface

```
ORIGIN 'bulletinboard';
BODY 'private/formbody';
```

```
(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)
```

```
-- XtEnvLib: attributes --
```

**Form:** BulletinBoard

```
(* A Form is a BulletinBoard with no input semantics of its own.
 * Constraints are placed on children of the Form to define
 * attachments for each of the child's four sides. These
 * attachments can be to the Form, to another child widget or
 * gadget, to a relative position within the Form, or to the
 * initial position of the child. The attachments determine the
 * layout behavior of the Form when resizing occurs. The
 * following are some important considerations in using a Form:
 *
 * + Every child must have an attachment on either the left or
 *   the right. If not specified, the result depends upon the
 *   value of the rubberPositioning resource. If the
 *   rubberPositioning resource is False, the child is given an
 *   leftAttachment of XmATTACH_FORM and an leftOffset equal to its
 *   current x value. If the rubberPositioning resource is True,
 *   the child is given an leftAttachment of XmATTACH_POSITION and
 *   an leftPosition proportional to the current x value divided by
 *   the width of the Form. In either case, if the child has not
 *   been previously given an x value, its x value is taken to be
 *   0, which places the child at the left side of the Form.
 *
 * + If you want to create a child without any attachments, and
 *   then later (e.g., after creating and managing it, but before
 *   realizing it) sets its rightAttachment, you must set its
 *   leftAttachment resource to XmATTACH_NONE at the same time.
 *
 * + The the resizable resource resource controls only whether a
 *   geometry request by the child will be granted. It has no
 *   effect on whether the child's size can be changed because of
 *   changes in geometry of the Form or of other children.
 *
 * + Every child has a preferred width, based on geometry
 *   requests it makes (whether they are granted or not).
 *
 * + If a child has attachments on both the left and the right
 *   sides, its size is completely controlled by the Form. It can
 *   be shrunk below its preferred width or enlarged above it, if
 *   necessary, due to other constraints. In addition, the child's
 *   geometry requests to change its own width may be refused.
 *
 * + If a child has attachments on only its left or right side,
 *   it will always be at its preferred width (if resizable,
 *   otherwise at its current width). This may cause it to be
 *   clipped by the Form or by other children.
 *
 * + If a child's left (or right) attachment is set to
 *   XmATTACH_SELF, its corresponding left (or right) offset is
 *   forced to 0. The attachment is then changed to
 *   XmATTACH_POSITION, with a position that corresponds to x value
 *   of the child's left (or right) edge. To fix the position of a
```



```

*   side at a specific x value use XmATTACH_FORM or
*   XmATTACH_OPPOSITE_FORM with the x value as the left (or right)
*   offset.
*
* + Unmapping a child has no effect on the Form except that the
*   child is not mapped.
*
* + Unmanaging a child unmaps it.  If no other child is
*   attached to it, or if all children attached to it and all
*   children recursively attached to them are also all unmanaged,
*   all of those children are treated as if they did not exist in
*   determining the size of the Form.
*
* + When changing the the x resource resource of a child, you
*   must first set its left attachment to either XmATTACH_SELF or
*   XmATTACH_NONE.  Otherwise, the request is not granted.  If the
*   resizable resource is False, the request is granted only if
*   the child's size can remain the same.
*
* + A left (or right) attachment of XmATTACH_WIDGET, where the
*   leftWidget resource (or the rightWidget resource) is 0, acts
*   like an attachment of XmATTACH_FORM.
*
* + If an attachment is made to a widget that is not a child of
*   the Form, but an ancestor of the widget is a child of the
*   Form, the attachment is made to the ancestor.  All these
*   considerations are true of top and bottom attachments as well,
*   with top acting like left, bottom acting like right, y acting
*   like x, and height acting like width.
*)
(# init::<
  (# WidgetClass::<
    (#
      do INNER;
      (if value=0 then xmFormWidgetClass->value if)
    #)
  do INNER
  #);

  (* Resources *)
horizontalSpacing:
  (* Specifies the offset for right and left attachments. *)
  ShortResource(# resourceName::< XmNhorizontalSpacing #);
verticalSpacing:
  (* Specifies the offset for top and bottom attachments. *)
  ShortResource(# resourceName::< XmNverticalSpacing #);
fractionBase:
  (* Specifies the denominator used in calculating the
   * relative position of a child widget using
   * XmATTACH_POSITION constraints.  If the value of a child's
   * leftAttachment (or rightAttachment) resource is
   * XmATTACH_POSITION, the position of the left (or right)
   * side of the child is relative to the left (or right) side
   * of THIS(Form) and is a fraction of the width of
   * THIS(Form).  This fraction is the value of the child's
   * leftPosition (or rightPosition) resource divided by the
   * value of THIS(Form).fractionBase.  If the value of a
   * child's topAttachment (or bottomAttachment) resource is
   * XmATTACH_POSITION, the position of the top (or bottom)
   * side of the child is relative to the top (or bottom) side
   * of THIS(Form) and is a fraction of the height of
   * THIS(Form).  This fraction is the value of the child's
   * topPosition (or bottomPosition) resource divided by the
   * value of THIS(Form).fractionBase.
   *)
  IntegerResource(# resourceName::< XmNfractionBase #);

```



**rubberPositioning:**

```
( * Indicates the default left and top attachments for a
 * child of THIS(Form). (Note: Whether this resource
 * actually applies to the left or right side of the child
 * and its attachment may depend on the value of the
 * stringDirection resource.) The default left attachment
 * is applied whenever resource changes leaves the child
 * without either a left or right attachment. The default
 * top attachment is applied whenever resource changes
 * leaves the child without either a top or bottom
 * attachment. If this Boolean resource is set to False,
 * the leftAttachment and topAttachment resources default to
 * XmATTACH_FORM, the leftOffset resource defaults to the
 * current x value of the left side of the child, and the
 * topOffset resource defaults to the current y value of the
 * child. The effect is to position the child according to
 * its absolute distance from the left or top side of
 * THIS(Form). If this resource is set to True, the
 * leftAttachment and topAttachment resources default to
 * XmATTACH_POSITION, the leftPosition resource defaults to
 * a value proportional to the current x value of the left
 * side of the child divided by the width of THIS(Form), and
 * the topPosition resource defaults to a value proportional
 * to the current y value of the child divided by the height
 * of THIS(Form). The effect is to position the child
 * relative to the left or top side of THIS(Form) and in
 * proportion to the width or height of THIS(Form).
 *)
BooleanResource(# resourceName:< XmNrubberPositioning #);

<<SLOT FormLib: attributes>>
#) (* Form *);
```

**FormDialog: Form**

```
( * A FormDialog consists of a DialogShell and an unmanaged Form
 * child of the DialogShell. A FormDialog is used for
 * interactions not supported by the standard dialog set. It
 * does not include any labels, buttons, or other dialog
 * components. Such components should be added to the FormDialog
 * by the application.
 *)
(# init::< (# WidgetClass:< (# do ... #);
  do true -> doNotManageChild;
  INNER
  #);

<<SLOT FormDialogLib: attributes>>
#) (* FormDialog *);

(* Aliases used in CompositeLib *)
mForm: Form(# #);
mFormDialog: FormDialog(# #);

--- CompositeLib: attributes ---

(* Redefinition of widgets within composites making the composite
 * the default father of the widgets
 *)
Form: mForm
  (# init::< (# GetFatherWidget:< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
FormDialog: mFormDialog
  (# init::< (# GetFatherWidget:< (# do THIS(Composite)->Value #);
    do INNER
```



```

        #)
    #);

--- XtObjectlib: attributes ---

(* Form Constraint Resources. Setting these resources reflects
 * constraints on children of Forms.
 *)
bottomAttachment:
    (* Specifies attachment of the bottom side of this child. It
     * can have the following values:
     *
     * + XmATTACH_NONE - do not attach the bottom side of this
     *   child.
     *
     * + XmATTACH_FORM - Attach the bottom side of this child to the
     *   bottom side of the Form parent.
     *
     * + XmATTACH_OPPOSITE_FORM - Attach the bottom side of this
     *   child to the top side of the Form parent. The bottomOffset
     *   resource can be used to determine the visibility of this
     *   child.
     *
     * + XmATTACH_WIDGET - Attach the bottom side of this child to
     *   the top side of the widget or gadget specified in the
     *   bottomWidget resource. If the bottomWidget resource is 0,
     *   XmATTACH_WIDGET is replaced by XmATTACH_FORM, and this child
     *   is attached to the bottom side of the Form parent.
     *
     * + XmATTACH_OPPOSITE_WIDGET - Attach the bottom side of this
     *   child to the bottom side of the widget or gadget specified in
     *   the bottomWidget resource.
     *
     * + XmATTACH_POSITION - Attach the bottom side of this child to
     *   a position that is relative to the bottom side of the Form
     *   parent and in proportion to the height of the Form parent.
     *   This position is determined by the bottomPosition and
     *   fractionBase resources.
     *
     * + XmATTACH_SELF - Attach the bottom side of this child to a
     *   position that is proportional to the current y value of the
     *   bottom of this child divided by the height of the Form parent.
     *   This position is determined by the bottomPosition and
     *   fractionBase resources. The bottomPosition resource is set to
     *   a value proportional to the current y value of the bottom of
     *   the child divided by the height of the Form parent.
     *)
    IntegerResource(# resourceName:< XmNbottomAttachment #);
bottomWidget:
    (* Specifies the widget or gadget to which the bottom side of
     * this child is attached. This resource is used if the
     * bottomAttachment resource is set to either XmATTACH_WIDGET or
     * XmATTACH_OPPOSITE_WIDGET.
     *)
    IntegerResource(# resourceName:< XmNbottomWidget #);
bottomPosition:
    (* This resource is used to determine the position of the bottom
     * side of this child when this child's bottomAttachment resource
     * is set to XmATTACH_POSITION. In this case the position of the
     * bottom side of this child is relative to the bottom side of
     * the Form parent and is a fraction of the height of the Form
     * parent. This fraction is the value of this child's
     * bottomPosition resource divided by the value of the Form's
     * fractionBase. For example, if this child's bottomPosition is
     * 50, the Form's fractionBase is 100, and the Form's height is
     * 200, the position of the bottom side of this child is 100.

```



```

*)
IntegerResource(# resourceName:< XmNbottomPosition #);
bottomOffset:
(* Specifies the constant offset between the bottom side of this
 * child and the object to which it is attached. The effect of a
 * nonzero value for this resource is undefined if the
 * bottomAttachment resource is set to XmATTACH_POSITION. The
 * relationship established remains, regardless of any resizing
 * operations that occur.
 *)
IntegerResource(# resourceName:< XmNbottomOffset #);
topWidget:
(* Specifies the widget or gadget to which the top side of this
 * child is attached. This resource is used if the topAttachment
 * resource is set to either XmATTACH_WIDGET or
 * XmATTACH_OPPOSITE_WIDGET.
 *)
IntegerResource(# resourceName:< XmNtopWidget #);
topPosition:
(* This resource is used to determine the position of the top
 * side of this child when this child's topAttachment resource is
 * set to XmATTACH_POSITION. In this case the position of the
 * top side of this child is relative to the top side of the Form
 * parent and is a fraction of the height of the Form parent.
 * This fraction is the value of this child's topPosition
 * resource divided by the value of the Form parent's
 * fractionBase resource. For example, if this child's
 * topPosition resource is 50, the Form parent's fractionBase
 * resource is 100, and the Form parent's height is 200, the
 * position of the top side of this child is 100.
 *)
IntegerResource(# resourceName:< XmNtopPosition #);
topOffset:
(* Specifies the constant offset between the top side of this
 * child and the object to which it is attached. The effect of a
 * nonzero value for this resource is undefined if topAttachment
 * is set to XmATTACH_POSITION. The relationship established
 * remains, regardless of any resizing operations that occur.
 *)
IntegerResource(# resourceName:< XmNtopOffset #);
topAttachment:
(* Specifies attachment of the top side of this child. It can
 * have following values:
 *
 * + XmATTACH_NONE - do not attach the top side of this child.
 *   If the bottomAttachment resource is also XmATTACH_NONE, this
 *   value is ignored and this child is given a default top
 *   attachment.
 *
 * + XmATTACH_FORM - Attach the top side of this child to the
 *   top side of the Form parent.
 *
 * + XmATTACH_OPPOSITE_FORM - Attach the top side of this child
 *   to the bottom side of the Form parent. the topOffset resource
 *   can be used to determine the visibility of this child.
 *
 * + XmATTACH_WIDGET - Attach the top side of this child to the
 *   bottom side of the widget or gadget specified in the topWidget
 *   resource. If the topWidget resource is NULL, XmATTACH_WIDGET
 *   is replaced by XmATTACH_FORM, and this child is attached to
 *   the top side of the Form parent.
 *
 * + XmATTACH_OPPOSITE_WIDGET - Attach the top side of this
 *   child to the top side of the widget or gadget specified in the
 *   topWidget resource.
 *)

```



```

* + XmATTACH_POSITION - Attach the top side of this child to a
* position that is relative to the top side of the Form parent
* and in proportion to the height of the Form parent. This
* position is determined by the topPosition and fractionBase
* resources.
*
* + XmATTACH_SELF - Attach the top side of this child to a
* position that is proportional to the current y value of this
* child divided by the height of the Form parent. This position
* is determined by the topPosition and fractionBase resources.
* the topPosition resource is set to a value proportional to the
* current y value of this child divided by the height of the
* Form parent.
*)
IntegerResource(# resourceName:< XmNtopAttachment #);
leftAttachment:
(* Specifies attachment of the left side of this child. (Note:
* Whether this resource actually applies to the left or right
* side of this child and its attachment may depend on the value
* of the stringDirection resource.) It can have the following
* values:
*
* + XmATTACH_NONE - do not attach the left side of this child.
* If the rightAttachment resource is also XmATTACH_NONE, this
* value is ignored and this child is given a default left
* attachment.
*
* + XmATTACH_FORM - Attach the left side of this child to the
* left side of the Form parent.
*
* + XmATTACH_OPPOSITE_FORM - Attach the left side of this child
* to the right side of the Form parent. The leftOffset resource
* can be used to determine the visibility of this child.
*
* + XmATTACH_WIDGET - Attach the left side of this child to the
* right side of the widget or gadget specified in the leftWidget
* resource. If the leftWidget resource is 0, XmATTACH_WIDGET is
* replaced by XmATTACH_FORM, and this child is attached to the
* left side of the Form parent.
*
* + XmATTACH_OPPOSITE_WIDGET - Attach the left side of this
* child to the left side of the widget or gadget specified in
* the leftWidget resource.
*
* + XmATTACH_POSITION - Attach the left side of this child to a
* position that is relative to the left side of the Form parent
* and in proportion to the width of the Form parent. This
* position is determined by the leftPosition and fractionBase
* resources.
*
* + XmATTACH_SELF - Attach the left side of this child to a
* position that is proportional to the current x value of the
* left side of this child divided by the width of the Form
* parent. This position is determined by the leftPosition and
* fractionBase resources. The leftPosition resource is set to a
* value proportional to the current x value of the left side of
* this child divided by the width of the Form parent.
*)
IntegerResource(# resourceName:< XmNleftAttachment #);
leftWidget:
(* Specifies the widget or gadget to which the left side of this
* child is attached. (Note: Whether this resource actually
* applies to the left or right side of this child and its
* attachment may depend on the value of the stringDirection
* resource.) This resource is used if leftAttachment is set to
* either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.

```



```

*)
IntegerResource(# resourceName:< XmNleftWidget #);
leftPosition:
(* This resource is used to determine the position of the left
 * side of this child when this child's leftAttachment resource
 * is set to XmATTACH_POSITION. (Note: Whether this resource
 * actually applies to the left or right side of this child and
 * its attachment may depend on the value of the stringDirection
 * resource.) In this case the position of the left side of this
 * child is relative to the left side of the Form parent and is a
 * fraction of the width of the Form parent. This fraction is
 * the value of this child's leftPosition resource resource
 * divided by the value of the Form parent's fractionBase
 * resource. For example, if this child's leftPosition resource
 * is 50, the Form parent's fractionBase resource is 100, and the
 * Form's width is 200, the position of the left side of this
 * child is 100.
*)
IntegerResource(# resourceName:< XmNleftPosition #);
leftOffset:
(* Specifies the constant offset between the left side of this
 * child and the object to which it is attached. (Note: Whether
 * this resource actually applies to the left or right side of
 * this child and its attachment may depend on the value of the
 * stringDirection resource.) The effect of a nonzero value for
 * this resource is undefined if the leftAttachment resource is
 * set to XmATTACH_POSITION. The relationship established
 * remains, regardless of any resizing operations that occur.
*)
IntegerResource(# resourceName:< XmNleftOffset #);
rightAttachment:
(* Specifies attachment of the right side of this child. (Note:
 * Whether this resource actually applies to the left or right
 * side of this child and its attachment may depend on the value
 * of the stringDirection resource.) It can have the following
 * values:
 *
 * + XmATTACH_NONE - do not attach the right side of this child.
 *
 * + XmATTACH_FORM - Attach the right side of this child to the
 *   right side of the Form parent.
 *
 * + XmATTACH_OPPOSITE_FORM - Attach the right side of this
 *   child to the left side of the Form parent. the rightOffset
 *   resource can be used to determine the visibility of this
 *   child.
 *
 * + XmATTACH_WIDGET - Attach the right side of this child to
 *   the left side of the widget or gadget specified in the
 *   rightWidget resource. If the rightWidget resource is 0,
 *   XmATTACH_WIDGET is replaced by XmATTACH_FORM, and this child
 *   is attached to the right side of the Form parent.
 *
 * + XmATTACH_OPPOSITE_WIDGET - Attach the right side of this
 *   child to the right side of the widget or gadget specified in
 *   the rightWidget resource.
 *
 * + XmATTACH_POSITION - Attach the right side of this child to
 *   a position that is relative to the right side of the Form
 *   parent and in proportion to the width of the Form parent.
 *   This position is determined by the rightPosition and
 *   fractionBase resources.
 *
 * + XmATTACH_SELF - Attach the right side of this child to a
 *   position that is proportional to the current x value of the
 *   right side of this child divided by the width of the Form

```



```

*   parent. This position is determined by the rightPosition and
*   fractionBase resources. The rightPosition resource is set to
*   a value proportional to the current x value of the right side
*   of this child divided by the width of the Form parent.
*)
IntegerResource(# resourceName:< XmNrightAttachment #);
rightWidget:
(* Specifies the widget or gadget to which the right side of
* this child is attached. (Note: Whether this resource actually
* applies to the left or right side of this child and its
* attachment may depend on the value of the stringDirection
* resource.) This resource is used if rightAttachment is set to
* either XmATTACH_WIDGET or XmATTACH_OPPOSITE_WIDGET.
*)
IntegerResource(# resourceName:< XmNrightWidget #);
rightPosition:
(* This resource is used to determine the position of the right
* side of this child when this child's rightAttachment is set to
* XmATTACH_POSITION. (Note: Whether this resource actually
* applies to the left or right side of this child and its
* attachment may depend on the value of the stringDirection
* resource.) In this case the position of the right side of
* this child is relative to the right side of the Form parent
* and is a fraction of the width of the Form parent. This
* fraction is the value of this child's rightPosition resource
* divided by the value of the Form parent's fractionBase. For
* example, if this child's rightPosition is 50, the Form
* parents's fractionBase is 100, and the Form parent's width is
* 200, the position of the right side of this child is 100.
*)
IntegerResource(# resourceName:< XmNrightPosition #);
rightOffset:
(* Specifies the constant offset between the right side of this
* child and the object to which it is attached. (Note: Whether
* this resource actually applies to the left or right side of
* this child and its attachment may depend on the value of the
* stringDirection resource.) The effect of a nonzero value for
* this resource is undefined if rightAttachment is set to
* XmATTACH_POSITION. The relationship established remains,
* regardless of any resizing operations that occur.
*)
IntegerResource(# resourceName:< XmNrightOffset #);
resizable:
(* This Boolean resource specifies whether or not this child's
* request for a new size is (conditionally) granted by the Form
* parent. If this resource is set to True the request is
* granted if possible. If this resource is set to False the
* request is always refused. If this child has both left and
* right attachments, its width is completely controlled by the
* Form parent, regardless of the value of this child's resizable
* resource. If this child has a left or right attachment but
* not both, this child's width is used in setting its width if
* the value of this child's resizable resource is True. These
* conditions are also true for top and bottom attachments, with
* height acting like width.
*)
BooleanResource(# resourceName:< XmNresizable #);

(* PanedWindow Constraint Resources. Setting these resources
* reflects constraints on children of PanedWindows
*)
allowResize:
(* Allows an application to specify whether the PanedWindow
* parent should allow a pane to request to be resized. This
* flag has an effect only after the PanedWindow and its children
* have been realized. If this flag is set to True, the

```



```

    * PanedWindow parent tries to honor requests to alter the height
    * of the pane. If False, it always denies pane requests to
    * resize.
    *)
BooleanResource(# resourceName:< XmNallowResize #);
minimum:
(* Allows an application to specify the minimum size to which a
 * pane may be resized. This value must be greater than 0.
 *)
IntegerResource(# resourceName:< XmNminimum #);
maximum:
(* Allows an application to specify the maximum size to which a
 * pane may be resized. This value must be greater than the
 * specified minimum.
 *)
IntegerResource(# resourceName:< XmNmaximum #);
skipAdjust:
(* When set to True, this Boolean resource allows an application
 * to specify that the PanedWindow parent should not
 * automatically resize this pane.
 *)
BooleanResource(# resourceName:< XmNskipAdjust #)

```

---

Form Interface

[Mjølner Informatics](#)



# Drawingarea Interface

```
ORIGIN 'manager';
BODY 'private/drawingareabody';

(*
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

DrawingArea: Manager
(* A DrawingArea is an empty widget that is easily adaptable to
 * a variety of purposes. It does no drawing and defines no
 * behavior except for invoking callbacks. Callbacks notify the
 * application when graphics need to be drawn (exposure events or
 * widget resize) and when the widget receives input from the
 * keyboard or mouse. Applications are responsible for defining
 * appearance and behavior as needed in response to DrawingArea
 * callbacks.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmDrawingAreaWidgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
marginWidth:
    (* Specifies the minimum spacing in pixels between the left
     * or right edge of THIS(DrawingArea) and any child widget.
     *)
    ShortResource(# resourceName::< XmNmarginWidth #);
marginHeight:
    (* Specifies the minimum spacing in pixels between the top
     * or bottom edge of THIS(DrawingArea) and any child widget.
     *)
    ShortResource(# resourceName::< XmNmarginHeight #);
resizePolicy:
    (* Controls the policy for resizing THIS(DrawingArea).
     * Possible values include XmRESIZE_NONE (fixed size),
     * XmRESIZE_ANY (shrink or grow as needed), and
     * XmRESIZE_GROW (grow only).
     *)
    CharResource(# resourceName::< XmNresizePolicy #);

    (* Callbacks *)
DrawingAreaCallback: MotifCallback
    (* Prefix for DrawingArea callbacks *)
    (# callData::< XmDrawingAreaCallbackStruct do INNER #);

exposeCallback:: DrawingAreaCallback
    (* Called when THIS(DrawingArea) receives an exposure
     * event. The callback reason is XmCR_EXPOSE. The callback
     * structure also includes the exposure event.
     *);
inputCallback:: DrawingAreaCallback
    (* Called when THIS(DrawingArea) receives a keyboard or
     * mouse event (key or button, up or down). The callback
```



```

    * reason is XmCR_INPUT.  The callback structure also
    * includes the input event.
    *);
resizeCallback:< DrawingAreaCallback
    (* Ccalled when THIS(DrawingArea) is resized.  The callback
    * reason is XmCR_RESIZE.
    *);
    (* Inherited callbacks *)
helpCallback:< DrawingAreaCallback;

installCallbacks:< (* Private *)
    (# do ...; INNER #);

    <<SLOT DrawingAreaLib: attributes>>
    #) (* DrawingArea *);

(* Alias used in CompositeLib *)
mDrawingArea: DrawingArea (# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
 * the default father of the widget
 *)
DrawingArea: mDrawingArea
    (# init:< (# GetFatherWidget:< (# do THIS(Composite)->value #);
        do INNER
        #)
    #)

```

---

Drawingarea Interface

[Mjølner Informatics](#)



# Frame Interface

```
ORIGIN 'manager';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

Frame: Manager
(* Frame is a very simple manager used to enclose a single child
 * in a border drawn by Frame. It uses the Manager class
 * resources for border drawing and performs geometry management
 * so that its size always matches its child's size plus the
 * margins defined for it. Frame is most often used to enclose
 * other managers when the application developer desires the
 * manager to have the same border appearance as the primitive
 * widgets. Frame can also be used to enclose primitive widgets
 * that do not support the same type of border drawing. This
 * gives visual consistency when you develop applications using
 * diverse widget sets. If the Frame's parent is a Shell widget,
 * shadowType is set to XmSHADOW_OUT and Manager's resource
 * shadowThickness is set to one by default.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmFrameWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
marginWidth:
    (* Specifies the padding space on the left and right sides
     * between THIS(Frame)'s child and THIS(Frame)'s shadow
     * drawing.
     *)
    ShortResource(# resourceName::< XmNmarginWidth #);
marginHeight:
    (* Specifies the padding space on the top and bottom sides
     * between THIS(Frame)'s child and THIS(Frame)'s shadow
     * drawing.
     *)
    ShortResource(# resourceName::< XmNmarginHeight #);
shadowType:
    (* Describes the drawing style for THIS(Frame). This
     * resource can have the following values:
     *
     * + XmSHADOW_IN - draws THIS(Frame) so that it appears
     * inset. This means that the bottom shadow visuals and
     * top shadow visuals are reversed.
     *
     * + XmSHADOW_OUT - draws THIS(Frame) so that it appears
     * outset. This is the default if THIS(Frame)'s parent
     * is a Shell widget.
     *
     * + XmSHADOW_ETCHED_IN - draws THIS(Frame) using a double
     * line giving the effect of a line etched into the
     * window. The thickness of the double line is equal to
```



```

*      the value of shadowThickness.  This is the default
*      except when THIS(Frame)'s parent is a Shell widget.
*
*      + XmSHADOW_ETCHED_OUT - draws THIS(Frame) using a double
*      line giving the effect of a line coming out of the
*      window.  The thickness of the double line is equal to
*      the value of shadowThickness.
*)
CharResource(# resourceName::< XmNshadowType #);

<<SLOT FrameLib: attributes>>
#) (* Frame *);

(* Alias used in CompositeLib *)
mFrame: Frame (# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
* the default father of the widget
*)
Frame: mFrame
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Frame Interface

[Mjølner Informatics](#)



# Rowcolumn Interface

```
ORIGIN 'manager';
BODY 'private/rowcolumnbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

RowColumn: Manager
(* The RowColumn widget is a general purpose RowColumn manager
 * capable of containing any widget type as a child. In general,
 * it requires no special knowledge about how its children
 * function and provides nothing beyond support for several
 * different layout styles. However, it can be configured as a
 * menu, in which case, it expects only certain children, and it
 * configures to a particular layout. The menus supported are:
 * MenuBar, Pulldown or Popup MenuPanels, and OptionMenu. The
 * type of layout performed is controlled by how the application
 * has set the various layout resources. It can be configured to
 * lay out its children in either rows or columns. In addition,
 * the application can specify how the children are laid out, as
 * follows:
 *
 * + the children are packed tightly together into either rows
 *   or columns
 *
 * + each child is placed in an identically sized box (producing
 *   a symmetrical look)
 *
 * + a specific layout (the current x and y positions of the
 *   children control their location) In addition, the
 *   application has control over both the spacing that occurs
 *   between each row and column and the margin spacing present
 *   between the edges of the RowColumn widget and any children
 *   that are placed against it. By default the RowColumn
 *   widget has no 3-D visuals associated with it; if an
 *   application wishes to have a 3-D shadow placed around this
 *   widget, it can create the RowColumn as a child of a Frame
 *   widget.
 *)
(# init::<
  (# WidgetClass::<
    (#
      do INNER;
      (if value=0 then xmRowColumnWidgetClass->value if)
    #)
  do INNER
  #);

  (* Resources *)
resizeWidth:
  (* Requests a new width if necessary, when set to True.
   * When set to False, THIS(RowColumn) does not request a new
   * width regardless of any changes to THIS(RowColumn) or its
   * children.
   *)
  BooleanResource(# resourceName::< XmNresizeWidth #);
resizeHeight:
  (* Requests a new height if necessary, when set to True.
```



```

    * When set to False, THIS(RowColumn) does not request a new
    * height regardless of any changes to THIS(RowColumn) or
    * its children.
    *)
    BooleanResource(# resourceName::< XmNresizeHeight #);
adjustLast:
    (* Extends the last row of children to the bottom edge of
    * THIS(RowColumn) (when orientation is XmHORIZONTAL) or
    * extends the last column to the right edge of
    * THIS(RowColumn) (when orientation is XmVERTICAL). This
    * feature is disabled by setting adjustLast to False.
    *)
    BooleanResource(# resourceName::< XmNadjustLast #);
marginWidth:
    (* Specifies the amount of blank space between the left
    * edge of THIS(RowColumn) and the first item in each row,
    * and the right edge of THIS(RowColumn) widget and the last
    * item in each row. The default value is 0 for Pulldown and
    * Popup MenuPanels, and three pixels for other RowColumn
    * types.
    *)
    ShortResource(# resourceName::< XmNmarginWidth #);
marginHeight:
    (* Specifies the amount of blank space between the top edge
    * of THIS(RowColumn) and the first item in each column, and
    * the bottom edge of THIS(RowColumn) and the last item in
    * each column. The default value is 0 for Pulldown and
    * Popup MenuPanels, and three pixels for other RowColumn
    * types.
    *)
    ShortResource(# resourceName::< XmNmarginHeight #);
isAligned:
    (* Specifies text alignment for each item within
    * THIS(RowColumn); this applies only to items that are
    * subpatterns of Label or LabelGadget. However, if the item
    * is a Label widget or gadget and its parent is either a
    * Popup MenuPanel or a Pulldown MenuPanel, alignment is not
    * performed; the Label is treated as the title within the
    * MenuPanel, and the alignment set by the application is not
    * overridden. entryAlignment controls the type of textual
    * alignment.
    *)
    BooleanResource(# resourceName::< XmNisAligned #);
adjustMargin:
    (* Specifies whether the INNER minor margins of all items
    * contained within THIS(RowColumn) are forced to the same
    * value. The INNER minor margin corresponds to the
    * marginLeft, marginRight, marginTop, and marginBottom
    * resources supported by Label and LabelGadget. A vertical
    * orientation causes marginLeft and marginRight for all
    * items in a particular column to be forced to the same
    * value; the value is the largest margin specified for one
    * of the Label items. This keeps all text within each row
    * or column lined up with all other text in its row or
    * column. If the rowColumnType is either XmMENU_POPUP or
    * XmMENU_PULLDOWN and this resource is True, only button
    * children have their margins adjusted.
    *)
    BooleanResource(# resourceName::< XmNadjustMargin #);
radioBehavior:
    (* Specifies a Boolean value that when True, indicates that
    * THIS(RowColumn) should enforce a RadioBox-type behavior
    * on all of its children that are ToggleButtons or
    * ToggleButtonGadgets. When the value of this resource is
    * True, the defaults for two resources for ToggleButton and
    * ToggleButtonGadget children change: indicatorType

```



```

* defaults to XmONE_OF_MANY and visibleWhenOff defaults to
* True.  RadioBox behavior dictates that when one toggle is
* selected and the user selects another toggle, the first
* toggle is unselected automatically.  THIS(RowColumn)
* usually does not enforce this behavior if the
* application, rather than the user, changes the state of a
* toggle.  THIS(RowColumn) does enforce this behavior if a
* toggle child is selected using the state pattern of
* ToggleButton(Gadget) with a notify argument of True.  The
* default value is False, except if THIS(RowColumn) is a
* RadioBox
*)
BooleanResource(# resourceName:< XmNradioBehavior #);
radioAlwaysOne:
(* If True, forces the active ToggleButton or
* ToggleButtonGadget to be automatically selected after
* having been unselected (if no other toggle was
* activated).  If False, the active toggle may be
* unselected.  The default value is True.  This resource is
* important only when radioBehavior is True.  The
* application can always add and subtract toggles from
* THIS(RowColumn) regardless of the selected/unselected
* state of the toggle.  The application can also manage and
* unmanage toggle children of THIS(RowColumn) at any time
* regardless of state.  Therefore, the application can
* sometimes create a RowColumn that has radioAlwaysOne set
* to True and none of the toggle children selected.  The
* result is undefined if the value of this resource is True
* and the application sets more than one ToggleButton at a
* time.
*)
BooleanResource(# resourceName:< XmNradioAlwaysOne #);
isHomogeneous:
(* Indicates if THIS(RowColumn) should enforce exact
* homogeneity among the items it contains; if True, only
* the widgets that are of the class indicated by entryClass
* are allowed as children of THIS(RowColumn).  This is most
* often used for MenuBars or RadioBoxes.  Attempting to
* insert a child that is not a member of the specified
* class generates a warning message.  The default value is
* False, except for a MenuBar or a RadioBox, when the
* default is True.
*)
BooleanResource(# resourceName:< XmNisHomogeneous #);
entryClass:
(* Specifies the only widget class that can be added to
* THIS(RowColumn); this resource is meaningful only when
* the isHomogeneous resource is set to True.  Both widget
* and gadget variants of the specified class may be added
* to THIS(RowColumn).  When rowColumnType is set to
* XmWORK_AREA and radioBehavior is True, the default value
* for entryClass is xmToggleButtonGadgetClass.  When
* rowColumnType is set to XmMENU_BAR, the value of
* entryClass is forced to xmCascadeButtonWidgetClass.
*)
IntegerResource(# resourceName:< XmNentryClass #);
menuHelpWidget:
(* Specifies the CascadeButton, which is treated as the
* Help widget if rowColumnType is set to XmMENU_BAR.  The
* MenuBar always places the Help widget at a lower corner
* of the menu.  If THIS(RowColumn) is any type other than
* XmMENU_BAR, this resource is not meaningful.
*)
IntegerResource(# resourceName:< XmNmenuHelpWidget #);
labelString:
(* A text string, which displays the label to one side of

```



```

* the selection area when rowColumnType is set to
* XmMENU_OPTION. This resource is not meaningful for all
* other RowColumn types. This resource needs to be set
* directly on the LabelGadget child, hence this special
* pattern.
*)
(# setText:
  (* Set THIS(labelString) as in MotifString.setText *)
  (# t: ^text
   enter t[]
   do ...
   #);
  getText:
  (* Get the value of THIS(labelString) as in
   * MotifString.getText
   *)
  (# t: ^text
   do ...
   exit t[]
   #);
  set:
  (* Set THIS(labelString) directly as in
   * MotifString.set
   *)
  (# value: @integer;
   enter value
   do ...
   #);
  get:
  (* Get the value of THIS(labelString) directly as in
   * MotifString.get
   *)
  (# value: @Integer
   do ...
   exit value
   #);
  resourceName: @XmNlabelString;
enter setText
exit getText
#);
subMenuId:
(* Specifies the Pulldown MenuPane to be associated with an
 * OptionMenu. This resource is useful only when
 * rowColumnType is set to XmMENU_OPTION. The default value
 * is 0.
 *)
IntegerResource(# resourceName:< XmNsubMenuId #);
menuHistory:
(* Specifies the last menu entry to be activated. It is
 * also useful for specifying the current selection for an
 * OptionMenu. If rowColumnType is set to XmMENU_OPTION,
 * the specified menu item is positioned under the cursor
 * when the menu is displayed. If THIS(RowColumn) has the
 * radioButton resource set to True, the widget field
 * associated with this resource contains the widget ID of
 * the last ToggleButton or ToggleButtonGadget to change
 * from unselected to selected. The default value is the
 * the first child in THIS(RowColumn).
 *)
IntegerResource(# resourceName:< XmNmenuHistory #);
popupEnabled:
(* Allows the menu system to enable keyboard input
 * (accelerators and mnemonics) defined for a Popup MenuPane
 * and any of its submenus. The Popup MenuPane needs to be
 * informed whenever its accessibility to the user changes
 * because posting of the Popup MenuPane is controlled by

```



```

* the application. The default value for this resource is
* True (keyboard input - accelerators and mnemonics -
* defined for the Popup MenuPane and any of its submenus is
* enabled).
*)
BooleanResource(# resourceName::< XmNpopupEnabled #);
numColumns:
(* Specifies the number of minor dimension extensions that
* are made to accommodate the entries; this attribute is
* meaningful only when packing is set to XmPACK_COLUMN.
* For vertically-oriented RowColumns, this attribute
* indicates how many columns are built; the number of
* entries per column is adjusted to maintain this number of
* columns, if possible. For horizontally-oriented
* RowColumn widgets, this attribute indicates how many rows
* are built. The default value is 1.
*)
ShortResource(# resourceName::< XmNnumColumns #);
entryAlignment:
(* Specifies the alignment type for children that are
* subclasses of Label or LabelGadget when isAligned is
* enabled. The following are textual alignment types:
*
* + XmALIGNMENT_BEGINNING - the default
*
* + XmALIGNMENT_CENTER
*
* + XmALIGNMENT_END
*)
CharResource(# resourceName::< XmNentryAlignment #);
entryVerticalAlignment:
(* Specifies the vertical alignment type for children that are
* subclasses of Label or LabelGadget when isAligned is
* enabled. The following are textual alignment types:
*
* + XmALIGNMENT_BEGINNING - the default
*
* + XmALIGNMENT_CENTER
*
* + XmALIGNMENT_END
*)
CharResource(# resourceName::< XmNentryVerticalAlignment #);
rowColumnType:
(* Specifies the type of RowColumn widget to be created. It
* is a non-standard resource that cannot be changed after
* it is set. The set of possible settings for this resource
* are:
*
* + XmWORK_AREA - the default
*
* + XmMENU_BAR
*
* + XmMENU_PULLDOWN
*
* + XmMENU_POPUP
*
* + XmMENU_OPTION This resource cannot be changed after
* the RowColumn widget is created.
*)
CharResource(# resourceName::< XmNrowColumnType #);
packing:
(* Specifies how to pack the items contained within
* THIS(RowColumn). This can be set to XmPACK_TIGHT,
* XmPACK_COLUMN or XmPACK_NONE. When a RowColumn widget
* packs the items it contains, it determines its major
* dimension using the value of the orientation resource.

```



```

* XmPACK_TIGHT indicates that given the current major
* dimension (for example, vertical if orientation is
* XmVERTICAL), entries are placed one after the other until
* the RowColumn widget must wrap. THIS(RowColumn) wraps
* when there is no room left for a complete child in that
* dimension. Wrapping occurs by beginning a new row or
* column in the next available space. Wrapping continues,
* as often as necessary, until all of the children are laid
* out. In the vertical dimension (columns), boxes are set
* to the same width; in the horizontal dimension (rows),
* boxes are set to the same depth. Each entry's position
* in the major dimension is left unaltered (for example, y
* is left unchanged when orientation is XmVERTICAL); its
* position in the minor dimension is set to the same value
* as the greatest entry in that particular row or column.
* The position in the minor dimension of any particular row
* or column is independent of all other rows or columns.
* XmPACK_COLUMN indicates that all entries are placed in
* identically sized boxes. The box is based on the largest
* height and width values of all the children widgets. The
* value of the numColumns resource determines how many
* boxes are placed in the major dimension, before extending
* in the minor dimension. XmPACK_NONE indicates that no
* packing is performed. The x and y attributes of each
* entry are left alone, and THIS(RowColumn) widget attempts
* to become large enough to enclose all entries. The
* default value is XmPACK_TIGHT except when building an
* OptionMenu or a RadioBox, when the default is
* XmPACK_COLUMN.
*)
CharResource(# resourceName::< XmNpacking #);

mnemonic:
(* This resource is useful only when rowColumnType is set
* to XmMENU_OPTION. Specifies a keysym for a key that, when
* pressed by the user along with the Alt modifier, posts
* the associated Pulldown MenuPane. The first character in
* the OptionMenu label string that exactly matches the
* mnemonic in the character set specified in
* mnemonicCharSet is underlined. The user can post the
* menu by pressing either the shifted or the unshifted
* mnemonic key. The default is no mnemonic.
*)
IntegerResource(# resourceName::< XmNmnemonic #);

mnemonicCharSet:
(* Specifies the character set of the mnemonic for an
* OptionMenu. The default is determined dynamically
* depending on the current language environment.
*)
IntegerResource(# resourceName::< XmNmnemonicCharset #);

whichButton:
(* Specifies the mouse button to which a menu system is
* sensitive. The default for XmMENU_POPUP is Menu. The
* default for XmMENU_OPTION, XmMENU_BAR, and XmWORK_AREA is
* Return. This resource is not useful for RowColumn
* widgets of type XmMENU_PULLDOWN. This resource is
* obsolete; it has been replaced by XmNmenuPost and is
* present for compatibility with older releases of
* OSF/Motif.
*)
IntegerResource(# resourceName::< XmNwhichButton #);

orientation:
(* Determines whether THIS(RowColumn) layouts are row-major
* or column-major. In a column-major layout, the children
* of THIS(RowColumn) are laid out in columns top to bottom
* within the THIS(RowColumn). In a row-major layout the
* children of THIS(RowColumn) are laid out in rows.

```



```

    * XmVERTICAL resource value selects a column-major layout.
    * XmHORIZONTAL resource value selects a row-major layout.
    * The default value is XmVERTICAL, except when creating a
    * MenuBar, when the default is XmHORIZONTAL.
    *)
CharResource(# resourceName::< XmNorientation #);
entryBorder:
    (* Imposes a uniform border width upon all children of
    * THIS(RowColumn). The default value is 0, which disables
    * the feature.
    *)
ShortResource(# resourceName::< XmNentryBorder #);
menuAccelerator:
    (* This resource is useful only when THIS(RowColumn) has
    * been configured to operate as a Popup MenuPane or a
    * MenuBar. The format of this resource is similar to the
    * left side specification of a translation string, with the
    * limitation that it must specify a key event. For a Popup
    * MenuPane, when the accelerator is typed by the user, the
    * Popup MenuPane is posted. For a MenuBar, when the
    * accelerator is typed by the user, the first item in the
    * MenuBar is highlighted, and traversal is enabled in the
    * MenuBar. The accelerator can be disabled by setting the
    * popupEnabled resource to False.
    *)
StringResource(# resourceName::< XmNmenuAccelerator #);
menuPost:
    (* Specifies an X event description indicating an event
    * that posts a menu system. The default for XmMENU_POPUP
    * is Menu Press. The default for XmMENU_OPTION,
    * XmMENU_BAR, and XmWORK_AREA is Return Press. This
    * resource is not useful for RowColumn widgets of type
    * XmMENU_PULLDOWN.
    *)
StringResource(# resourceName::< XmNmenuPost #);
entryCallback:
    (* Disables the activateCallback and valueChangedCallback
    * callbacks for all CascadeButton, DrawnButton, PushButton,
    * and ToggleButton widgets and gadgets contained within
    * THIS(RowColumn). If the application supplies this
    * resource, the activateCallback and valueChangedCallback
    * callbacks are then revectorized to the entryCallback. This
    * allows an application to supply a single callback pattern
    * for handling all items contained in THIS(RowColumn). The
    * callback reason is XmCR_ACTIVATE. If the application
    * does not supply this resource, the activateCallback and
    * valueChangedCallback callbacks for each item in
    * THIS(RowColumn) work as normal. The application must
    * supply this resource at startup is created. Changing
    * this resource after creation is not supported.
    *)
IntegerResource(# resourceName::< XmNentryCallback #);
spacing:
    (* Specifies the horizontal and vertical spacing between
    * items contained within THIS(RowColumn). The default
    * value is three pixels for XmOPTION_MENU and XmWORK_AREA
    * and 0 for other RowColumn types.
    *)
ShortResource(# resourceName::< XmNspacing #);

(* Utility patterns *)
getPostedFromWidget: IntegerValue
    (* Returns the widget from which a menu was posted. An
    * application can use this pattern during the activate
    * callback to determine the context in which the menu
    * callback should be interpreted.

```



```

    *)
    (# menu: @widget
    enter menu
    do ...
    #);

(* Callbacks *)
RowColumnCallback: MotifCallback
(* Prefix for RowColumn callbacks *)
(# callData::< XmRowColumnCallbackStruct do INNER #);

mapCallback:< RowColumnCallback
(* Invoked when the window associated with THIS(RowColumn)
* is about to be mapped. The callback reason is XmCR_MAP.
*);
unmapCallback:< RowColumnCallback
(* Called after the window associated with THIS(RowColumn)
* has been unmapped. The callback reason is XmCR_UNMAP.
*);
(* Inherited callbacks *)
helpCallback:< RowColumnCallback;

installCallbacks:< (* Private *)
(# do ...; INNER #);

<<SLOT RowColumnLib: attributes>>
#) (* RowColumn *);

PopupMenu: RowColumn
(* A PopupMenu is a RowColumn widget of type XmMENU_POPUP. When
* the Popup MenuPane is created, a MenuShell widget is
* automatically created as the parent of the MenuPane. The
* PopupMenu is used as the first MenuPane within a PopupMenu
* system; all other MenuPanes are of the Pulldown type. A Popup
* MenuPane displays a 3-D shadow, unless the feature is disabled
* by the application. The shadow appears around the edge of the
* MenuPane. A PopupMenu is "popped up" using manageChild, and
* "popped down" using unmanageChild. The menu can be positioned
* before being managed using the "position" pattern below.
*)
(# init:< (# WidgetClass:< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

(* Utility patterns *)
position:
(* Positions THIS(PopupMenu) using the information in the
* specified event. The x_root and y_root values in the
* specified event are used to determine the menu position.
*)
(# event: @XButtonPressedEvent
enter event
do ...
#);

<<SLOT PopupMenuLib: attributes>>
#) (* PopupMenu *);

PulldownMenu: RowColumn
(* A PulldownMenu is a RowColumn of type XmMENU_PULLDOWN. When
* creating a PulldownMenu, a MenuShell is automatically created
* as the parent of PulldownMenu. If the parent specified is a
* PopupMenu or a PulldownMenu, the MenuShell is created as a
* child of the parent's MenuShell; otherwise, it is created as a
* child of the specified parent widget. A PulldownMenu displays

```



```

* a 3-D shadow, unless the feature is disabled by the
* application. The shadow appears around the edge of the
* PulldownMenu. A PulldownMenu is used when creating submenus
* that are to be attached to a CascadeButton or a
* CascadeButtonGadget. This is the case for all MenuPanes that
* are part of a PulldownMenu system (a MenuBar), the MenuPane
* associated with an OptionMenu, and any MenuPanes that cascade
* from a Popup MenuPane. PulldownMenus that are to be
* associated with an OptionMenu must be created before the
* OptionMenu is created. The PulldownMenu must be attached to a
* CascadeButton or CascadeButtonGadget that resides in a
* MenuBar, a Popup MenuPane, a Pulldown MenuPane, or an
* OptionMenu. This is done by using the button resource
* subMenuId. To function correctly when incorporated into a
* menu, the PulldownMenu's hierarchy must be considered; this
* hierarchy depends on the type of menu system that is being
* built as follows:
*
* + If the PulldownMenu is to be pulled down from a MenuBar,
*   its parent must be the MenuBar.
*
* + If the PulldownMenu is to be pulled down from a PopupMenu
*   or another PulldownMenu, its parent must be that PopupMenu
*   or PulldownMenu.
*
* + If the PulldownMenu is to be pulled down from an
*   OptionMenu, its parent must be the same as the OptionMenu
*   parent.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do true -> doNotManageChild;
        INNER
    #);

    <<SLOT PulldownMenuLib: attributes>>
#) (* PulldownMenu *);

```

**MenuBar:** RowColumn

```

(* A MenuBar is a RowColumn widget of type XmMENU_BAR. A
* MenuBar is generally used for building a Pulldown menu system.
* Typically, a MenuBar is created and placed along the top of
* the application window, and several CascadeButtons are
* inserted as the children. Each of the CascadeButtons has a
* PulldownMenu associated with it. These PulldownMenus must have
* been created as children of the MenuBar. The user interacts
* with the MenuBar by using either the mouse or the keyboard. A
* MenuBar displays a 3-D shadow along its border. The
* application controls the shadow attributes using the
* visual-related resources supported by Manager. A MenuBar
* widget is homogeneous in that it accepts only children that
* are specializations of CascadeButton or CascadeButtonGadget.
* Attempting to insert a child of a different class results in a
* warning message. If a MenuBar does not have enough room to
* fit all of its subwidgets on a single line, the MenuBar
* attempts to wrap the remaining entries onto additional lines
* if allowed by the geometry manager of the parent widget.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do INNER
    #);

    <<SLOT MenuBarLib: attributes>>
#) (* MenuBar *);

```

**OptionMenu:** RowColumn

```

(* An OptionMenu is a RowColumn widget of type XmMENU_OPTION.

```



```

* An OptionMenu widget is a specialized RowColumn manager
* composed of a label, a selection area, and a single Pulldown
* MenuPane. When an application creates an OptionMenu widget,
* it supplies the label string and the PulldownMenu. In order
* to succeed, there must be a valid subMenuId resource set.
* When the OptionMenu is created, the PulldownMenu must have
* been created as a child of the OptionMenu's parent and must be
* specified. The LabelGadget and the selection area (a
* CascadeButtonGadget) are created by the OptionMenu. An
* OptionMenu is laid out with the label displayed on one side of
* the widget and the selection area on the other side. The
* selection area has a dual purpose; it displays the label of
* the last item selected from the associated PulldownMenu, and
* it provides the means for posting the PulldownMenu. An
* OptionMenu typically does not display any 3-D visuals around
* itself or the internal LabelGadget. By default, the internal
* CascadeButtonGadget has a visible 3-D shadow. The application
* may change this by getting the CascadeButtonGadget ID using
* the getButtonGadget pattern, and then calling XtSetValues
* using the standard visual-related resources. The PulldownMenu
* is posted by moving the mouse pointer over the selection area
* and pressing a mouse button defined by OptionMenu's RowColumn
* parent. The PulldownMenu is posted and positioned so that the
* last selected item is directly over the selection area. The
* mouse is then used to arm the desired menu item. When the
* mouse button is released, the armed menu item is selected and
* the label within the selection area is changed to match that
* of the selected item. The OptionMenu also operates by using
* the keyboard interface mechanism. If the application has
* established a mnemonic with the OptionMenu, typing Alt with
* the mnemonic causes the PulldownMenu to be posted with
* traversal enabled. The standard traversal keys can then be
* used to move within the Menu. Selection can occur as the
* result of pressing the Return key or typing a mnemonic or
* accelerator for one of the menu items. An application may use
* the menuHistory resource to indicate which item in the
* PulldownMenu should be treated as the current choice and have
* its label displayed in the selection area. By default, the
* first item in the PulldownMenu is used.
*)
(# init::< (# WidgetClass::< (# do ... #);
    subMenu:< IntegerValue
        (* Pulldown MenuPane to post - MUST be specified *);
    do INNER
    #);

    (* Utility patterns *)
getButtonGadget: IntegerValue
    (* Obtains the widget ID for the internally created
    * CascadeButtonGadget.
    *)
    (# do ... #);
getLabelGadget: IntegerValue
    (* Obtains the widget ID for the internally created
    * LabelGadget.
    *)
    (# do ... #);

    <<SLOT OptionMenuLib: attributes>>
#) (* OptionMenu *);

```

#### **RadioBox:** RowColumn

```

(* A RadioBox is a RowColumn widget of type XmWORK_AREA.
* Typically, this is a composite widget that contains multiple
* ToggleButtonGadgets. The RadioBox arbitrates and ensures that
* at most one ToggleButtonGadget is on at any time. This

```



```

* pattern provides initial values for several RowColumn
* resources. It initializes packing to XmPACK_COLUMN,
* radioButton to True, isHomogeneous to True, and entryClass
* to xmToggleButtonGadgetClass. In a RadioBox the ToggleButton
* or ToggleButtonGadget resource indicatorType defaults to
* XmONE_OF_MANY, and the ToggleButton or ToggleButtonGadget
* resource visibleWhenOff defaults to True.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do INNER
    #);

    <<SLOT RadioBoxLib: attributes>>
#) (* RadioBox *);

(* Aliases used in CompositeLib *)
mRowColumn: RowColumn (# #);
mMenuBar: MenuBar(# #);
mOptionMenu: OptionMenu(# #);
mPopupMenu: PopupMenu (# #);
mPulldownMenu: PulldownMenu(# #);
mRadioBox: RadioBox(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widgets within composites making the composite
* the default father of the widgets
*)
RowColumn: mRowColumn
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #);
PopupMenu: mPopupMenu
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #);
PulldownMenu: mPullDownMenu
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #);
MenuBar: MMenuBar
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #);
OptionMenu: MOptionMenu
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #);
RadioBox: MRadioBox
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #)

```

---



# Scrolledwindow Interface

```
ORIGIN 'manager';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

ScrolledWindow: Manager
(* The ScrolledWindow widget combines one or two ScrollBar
 * widgets and a viewing area to implement a visible window onto
 * some other (usually larger) data display. The visible part of
 * the window can be scrolled through the larger display by the
 * use of ScrollBars. A ScrolledWindow can be configured to
 * operate automatically so that it performs all scrolling and
 * display actions with no need for application program
 * involvement. It can also be configured to provide a minimal
 * support framework in which the application is responsible for
 * processing all user input and making all visual changes to the
 * displayed data in response to that input. When a
 * ScrolledWindow is performing automatic scrolling it creates a
 * clipping window. Conceptually, this window becomes the
 * viewport through which the user examines the larger underlying
 * data area. The application simply creates the desired data,
 * then makes that data the work area of the ScrolledWindow.
 * When the user moves the slider to change the displayed data,
 * the workspace is moved under the viewing area so that a new
 * portion of the data becomes visible. Sometimes it is
 * impractical for an application to create a large data space
 * and simply display it through a small clipping window. For
 * example, in a text editor, creating a single data area that
 * consisted of a large file would involve an undesirable amount
 * of overhead. The application needs to use a ScrolledWindow (a
 * small viewport onto some larger data), but needs to be
 * notified when the user scrolled the viewport so it could bring
 * in more data from storage and update the display area. For
 * these cases the ScrolledWindow can be configured so that it
 * provides only visual layout support. No clipping window is
 * created, and the application must maintain the data displayed
 * in the work area, as well as respond to user input on the
 * ScrollBars.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then
                xmScrolledWindowWidgetClass->value
            if)
        #)
    do INNER
    #);

    (* Resources *)
horizontalScrollBar: IntegerResource
    (* Specifies the horizontal ScrollBar. *)
    (# resourceName::< XmNhorizontalScrollBar #);
verticalScrollBar: IntegerResource
    (* Specifies the vertical ScrollBar *)
    (# resourceName::< XmNverticalScrollBar #);
```



```

workWindow: IntegerResource
    (* Specifies the viewing area. *)
    (# resourceName:< XmNworkWindow #);
clipWindow: IntegerResource
    (* Specifies the widget ID of the clipping area. This is
    * automatically created by THIS(ScrolledWindow) when the
    * visualPolicy resource is set to XmCONSTANT and can be
    * read only by the application. Any attempt to set this
    * resource to a new value causes a warning message to be
    * printed by the scrolled window. If the visualPolicy
    * resource is set to XmVARIABLE, this resource is set to
    * NULL, and no clipping window is created.
    *)
    (# resourceName:< XmNclipWindow #);
scrollingPolicy: CharResource
    (* Performs automatic scrolling of the work area with no
    * application interaction. If the value of this resource
    * is XmAUTOMATIC, THIS(ScrolledWindow) automatically
    * creates the ScrollBars; attaches callbacks to the
    * ScrollBars; sets the visual policy to XmCONSTANT; and
    * automatically moves the work area through the clip window
    * in response to any user interaction with the ScrollBars.
    * An application can also add its own callbacks to the
    * ScrollBars. This allows the application to be notified
    * of a scroll event without having to perform any layout
    * procedures. This resource cannot be changed after
    * creation.
    *)
    (# resourceName:< XmNscrollingPolicy #);
visualPolicy: CharResource
    (* Grows THIS(ScrolledWindow) to match the size of the work
    * area, or it can be used as a static viewport onto a
    * larger data space. If the visual policy is XmVARIABLE,
    * THIS(ScrolledWindow) forces the ScrollBar display policy
    * to XmSTATIC and allow the work area to grow or shrink at
    * any time and adjusts its layout to accommodate the new
    * size. When the policy is XmCONSTANT, the work area grows
    * or shrinks as requested, but a clipping window forces the
    * size of the visible portion to remain constant. The only
    * time the viewing area can grow is in response to a resize
    * from THIS(ScrolledWindow)'s parent. The default is
    * XmCONSTANT when scrollingPolicy is XmAUTOMATIC, and
    * XmVARIABLE otherwise. This resource cannot be changed
    * after creation.
    *)
    (# resourceName:< XmNvisualPolicy #);
scrollBarDisplayPolicy: CharResource
    (* Controls the automatic placement of the ScrollBars. If
    * it is set to XmAS_NEEDED and if scrollingPolicy is set to
    * XmAUTOMATIC, ScrollBars are displayed only if the
    * workspace exceeds the clip area in one or both
    * dimensions. A resource value of XmSTATIC causes
    * THIS(ScrolledWindow) to display the ScrollBars whenever
    * they are managed, regardless of the relationship between
    * the clip window and the work area. This resource must be
    * XmSTATIC when scrollingPolicy is XmAPPLICATION_DEFINED.
    * The default is XmAS_NEEDED when scrollingPolicy is
    * AUTOMATIC, and XmSTATIC otherwise.
    *)
    (# resourceName:< XmNscrollBarDisplayPolicy #);
scrollBarPlacement:
    (* Specifies the positioning of the ScrollBars in relation
    * to the work window. The following are the values:
    *
    * + XmTOP_LEFT - The horizontal ScrollBar is placed above
    *   the work window; the vertical ScrollBar to the left.

```



```

*
* + XmBOTTOM_LEFT - The horizontal ScrollBar is placed
*   below the work window; the vertical ScrollBar to the
*   left.
*
* + XmTOP_RIGHT - The horizontal ScrollBar is placed above
*   the work window; the vertical ScrollBar to the right.
*
* + XmBOTTOM_RIGHT - The horizontal ScrollBar is placed
*   below the work window; the vertical ScrollBar to the
*   right. The default value may depend on the value of
*   the stringDirection resource.
*)
CharResource(# resourceName:< XmNScrollBarPlacement #);
scrolledWindowMarginWidth ShortResource
(* Specifies the margin width on the right and left sides
* of THIS(ScrolledWindow).
*)
(# resourceName:< XmNscrolledWindowMarginWidth #);
scrolledWindowMarginHeight ShortResource
(* Specifies the margin height on the top and bottom of
* THIS(ScrolledWindow).
*)
(# resourceName:< XmNscrolledWindowMarginHeight #);
spacing ShortResource
(* Specifies the distance that separates the ScrollBars
* from the work window.
*)
(# resourceName:< XmNspacing #);

<<SLOT ScrolledWindowLib: attributes>>
#) (* ScrolledWindow *);

(* Alias used in CompositeLib *)
mScrolledWindow ScrolledWindow (# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
* the default father of the widget
*)
ScrolledWindow mScrolledWindow
(# init:< (# GetFatherWidget:< (# do THIS(Composite)->value #);
do INNER
#)
#)
#)

```

---

Scrolledwindow Interface

[Mjølner Informatics](#)



# Mainwindow Interface

```
ORIGIN 'scrolledwindow';
BODY 'private/mainwindowbody';

(*
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

MainWindow: ScrolledWindow
(* A MainWindow provides a standard layout for the primary
 * window of an application. This layout includes a MenuBar, a
 * CommandWindow, a work region, a MessageWindow, and ScrollBars.
 * Any or all of these areas are optional. The work region and
 * ScrollBars in the MainWindow behave identically to the work
 * region and ScrollBars in the ScrolledWindow widget. The user
 * can think of the MainWindow as an extended ScrolledWindow with
 * an optional MenuBar and optional CommandWindow and
 * MessageWindow. In a fully-loaded MainWindow, the MenuBar
 * spans the top of the window horizontally. The CommandWindow
 * spans the MainWindow horizontally just below the MenuBar, and
 * the work region lies below the CommandWindow. The
 * MessageWindow is below the work region. Any space
 * remaining below the MessageWindow is managed in a manner
 * identical to ScrolledWindow. The behavior of a ScrolledWindow
 * can be controlled by the ScrolledWindow resources. When
 * creating a MainWindow, create also the work region elements, a
 * MenuBar, a CommandWindow, a MessageWindow, and then invoke the
 * setAreas pattern with those widgets as arguments. A
 * MainWindow can also create three Separator widgets that
 * provide a visual separation of MainWindow's four components.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmMainWindowWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
commandWindow:
    (* Specifies the widget to be laid out as the
     * CommandWindow. This widget must have been previously
     * created and managed as a child of THIS(MainWindow).
     *)
    IntegerResource(# resourceName::< XmNcommandWindow #);
commandWindowLocation:
    (* Controls the position of the command window.
     * XmCOMMAND_ABOVE_WORKSPACE locates the command window
     * between the menu bar and the work window.
     * XmCOMMAND_BELOW_WORKSPACE locates the command window
     * between the work window and the message window.
     *)
    CharResource(# resourceName::< XmNcommandWindowLocation #);
theMenuBar:
    (* The menu bar attached to THIS(MainWindow), if any. It
     * should be named just "menuBar", but this would hide the
     * MenuBar widget class, so "theMenuBar" was chosen instead.
```



```

    *)
    IntegerResource(# resourceName::< XmNmenuBar #);
messageWindow:
    (* Specifies the widget to be laid out as the
    * MessageWindow. This widget must have been previously
    * created and managed as a child of THIS(MainWindow). The
    * MessageWindow is positioned at the bottom of
    * THIS(MainWindow). If this value is 0, no message window
    * is included in THIS(MainWindow).
    *)
    IntegerResource(# resourceName::< XmNmessageWindow #);
mainWindowMarginWidth:
    (* Specifies the margin width on the right and left sides
    * of THIS(MainWindow). This resource overrides any setting
    * of the ScrolledWindow resource scrolledWindowMarginWidth.
    *)
    ShortResource(# resourceName::< XmNmainWindowMarginWidth #);
mainWindowMarginHeight:
    (* Specifies the margin height on the top and bottom of
    * THIS(MainWindow). This resource overrides any setting of
    * the ScrolledWindow resource scrolledWindowMarginHeight.
    *)
    ShortResource(# resourceName::< XmNmainWindowMarginHeight #);
showSeparator:
    (* Displays separators between the components of
    * THIS(MainWindow) when set to True. If set to False, no
    * separators are displayed.
    *)
    BooleanResource(# resourceName::< XmNshowSeparator #);

    (* Utility patterns *)
setAreas:
    (* Used to identify manageable children. Each area is
    * optional, and may be specified as 0.
    *)
    (# menubar, command_window, h_scroll,
    v_scroll, work_region: @integer;
    enter (menubar, command_window, h_scroll, v_scroll,
    work_region)
    do ...
    #);
separator: IntegerValue
    (* Returns the widget ID of Separator number n in
    * THIS(MainWindow). The first Separator widget is located
    * between the MenuBar and the Command widget. The second
    * Separator widget is located between the Command widget
    * and the ScrolledWindow. The third Separator widget is
    * located between the message window and the widget above
    * it. These Separators are visible only when showSeparator
    * is True.
    *)
    (# n: @integer (* Number of separator; must be 1, 2, or 3 *)
    enter n
    do ...
    #);

    <<SLOT MainWindowLib: attributes>>
    #) (* MainWindow *);

    (* Alias used in CompositeLib *)
mMainWindow: MainWindow (# #);

    --- CompositeLib: attributes ---

    (* Redefinition of widget within composites making the composite
    * the default father of the widget

```



```
* )  
MainWindow: mMainWindow  
  ( # init::< ( # GetFatherWidget::< ( # do THIS(Composite)->value # );  
    do INNER  
    # )  
  # )
```

---

Mainwindow Interface

[Mjølner Informatics](#)



# Scale Interface

```
ORIGIN 'manager';
BODY 'private/scalebody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

Scale: Manager
(* A Scale is used by an application to indicate a value from
 * within a range of values, and it allows the user to input or
 * modify a value from the same range. A Scale has an elongated
 * rectangular region similar to a ScrollBar. A slider inside
 * this region indicates the current value along the Scale. The
 * user can also modify the Scale's value by moving the slider
 * within the rectangular region of the Scale. A Scale can also
 * include a label set located outside the Scale region. These
 * can indicate the relative value at various positions along the
 * scale. A Scale can be either input/output or output only. An
 * input/output Scale's value can be set by the application and
 * also modified by the user with the slider. An output-only
 * Scale is used strictly as an indicator of the current value of
 * something and cannot be modified interactively by the user.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmScaleWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
value:
    (* Specifies the slider's current position along the scale,
     * between minimum and maximum.
     *)
    IntegerResource(# resourceName::< XmNvalue #);
maximum:
    (* Specifies the slider's maximum value. *)
    IntegerResource(# resourceName::< XmNmaximum #);
minimum:
    (* Specifies the slider's minimum value. *)
    IntegerResource(# resourceName::< XmNminimum #);
orientation:
    (* Tells whether to display THIS(Scale) vertically or
     * horizontally. This resource can have values of
     * XmVERTICAL and XmHORIZONTAL
     *)
    CharResource(# resourceName::< XmNorientation #);
processingDirection:
    (* Specifies whether the value for maximum is on the right
     * or left side of minimum for horizontal Scales or above or
     * below minimum for vertical Scales. This resource can
     * have values of XmMAX_ON_TOP, XmMAX_ON_BOTTOM,
     * XmMAX_ON_LEFT, and XmMAX_ON_RIGHT. If the XmScale is
     * oriented vertically, the default value is XmMAX_ON_TOP.
     * If the XmScale is oriented horizontally, the default
```



```

    * value may depend on the value of the stringDirection
    * resource.
    *)
    CharResource(# resourceName::< XmNprocessingDirection #);
titleLabel:
    (* Specifies the title text string to appear in the window
    * of THIS(Scale).
    *)
    MotifStringResource(# resourceName::< XmNtitleLabel #);
fontList:
    (* Specifies the font list to use for the title text string
    * specified by titleLabel. If this value is 0 at
    * initialization, it is initialized by looking up the
    * parent hierarchy of the widget for an ancestor that is
    * specialization of BulletinBoard, VendorShell, or
    * MenuShell. If such an ancestor is found, the font list
    * is initialized to the appropriate default font list of
    * the ancestor widget (defaultFontList for VendorShell and
    * MenuShell, labelFontList for BulletinBoard)
    *)
    IntegerResource(# resourceName::< XmNfontList #);
showValue:
    (* Specifies whether a label for the current slider value
    * should be displayed next to the slider. If the value is
    * True, the current slider value is displayed.
    *)
    BooleanResource(# resourceName::< XmNshowValue #);
decimalPoints:
    (* Specifies the number of decimal points to shift the
    * slider value when displaying it. For example, a slider
    * value of 2,350 and an XmdecimalPoints value of 2 results
    * in a display value of 23.50
    *)
    ShortResource(# resourceName::< XmNdecimalPoints #);
scaleWidth:
    (* Specifies the width of the slider area. The value
    * should be in the specified unit type (the default is
    * pixels). If no value is specified a default width is
    * computed.
    *)
    ShortResource(# resourceName::< XmNscaleWidth #);
scaleHeight:
    (* Specifies the height of the slider area. The value
    * should be in the specified unit type (the default is
    * pixels). If no value is specified a default height is
    * computed.
    *)
    ShortResource(# resourceName::< XmNscaleHeight #);
scaleMultiple:
    (* Specifies the amount to move the slider when the user
    * takes an action that moves the slider by a multiple
    * increment. The default is (maximum - minimum) divided by
    * 10.
    *)
    IntegerResource(# resourceName::< XmNscaleMultiple #);
highlightOnEnter:
    (* Specifies whether the highlighting rectangle is drawn
    * when the cursor moves into THIS(Scale). If the shell's
    * focus policy is XmEXPLICIT, this resource is ignored, and
    * THIS(Scale) is highlighted when it has the focus. If the
    * shell's focus policy is XmPOINTER and if this resource is
    * True, the highlighting rectangle is drawn when the the
    * cursor moves into THIS(Scale). If the shell's focus
    * policy is XmPOINTER and if this resource is False, the
    * highlighting rectangle is not drawn when the the cursor
    * moves into THIS(Scale). The default is False.

```



```

    *)
    BooleanResource(# resourceName::< XmNhighlightOnEnter #);
highlightThickness:
    (* Specifies the size of the slider's border drawing
    * rectangle used for enter window and traversal highlight
    * drawing.
    *)
    ShortResource(# resourceName::< XmNhighlightThickness #);

    (* Callbacks *)
ScaleCallback: MotifCallback
    (* Prefix for callbacks to THIS(Scale) *)
    (# callData::< XmScaleCallbackStruct do INNER #);

dragCallback: < ScaleCallback
    (* Called when the slider position changes as the slider is
    * being dragged. The reason sent by the callback is
    * XmCR_DRAG.
    *);
valueChangedCallback: < ScaleCallback
    (* Called when the value of the slider has changed. The
    * reason sent by the callback is XmCR_VALUE_CHANGED.
    *);
    (* Inherited callbacks *)
helpCallback: < ScaleCallback;

installCallbacks: < (* Private *)
    (# do ...; INNER #);

    <<SLOT ScaleLib: attributes>>
    #) (* Scale *);

    (* Alias used in CompositeLib *)
mScale: Scale (# #);

    --- CompositeLib: attributes ---

    (* Redefinition of widget within composites making the composite
    * the default father of the widget
    *)
Scale: mScale
    (# init: < (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
    #)
    #)

```

---

## Scale Interface

[Mjølner Informatics](#)



# Panedwindow Interface

```
ORIGIN 'manager';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

PanedWindow: Manager
(* A PanedWindow is Composite that lays out children in a
 * vertically tiled format. Children appear in top-to-bottom
 * fashion, with the first child inserted appearing at the top of
 * the PanedWindow and the last child inserted appearing at the
 * bottom. The PanedWindow grows to match the width of its
 * widest child and all other children are forced to this
 * width. The height of the PanedWindow is equal to the sum of
 * the heights of all its children, the spacing between them, and
 * the size of the top and bottom margins. The user can also
 * adjust the size of the panes. To facilitate this adjustment,
 * a pane control sash is created for most children. The sash
 * appears as a square box positioned on the bottom of the pane
 * that it controls. The user can adjust the size of a pane by
 * using the mouse or keyboard. The PanedWindow is also a
 * Constraint, which means that it creates and manages a set of
 * constraints for each child. You can specify a minimum and
 * maximum size for each pane. The PanedWindow does not allow a
 * pane to be resized below its minimum size or beyond its
 * maximum size. Also, when the minimum size of a pane is equal
 * to its maximum size, no control sash is presented for that
 * pane or for the lowest pane. See the set of PanedWindow
 * Constraint resources at the end of this file. The default
 * insertPosition procedure for PanedWindow causes sashes to be
 * inserted at the end of the list of children and causes
 * non-sash widgets to be inserted after other non-sash children
 * but before any sashes.
 *)
(# init:<
    (# WidgetClass:<
        (#
            do INNER;
            (if value=0 then xmPanedWindowWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
marginWidth:
    (* Specifies the distance between the left and right edges
     * of THIS(PanedWindow) and its children.
     *)
    ShortResource(# resourceName:< XmNmarginWidth #);
marginHeight:
    (* Specifies the distance between the top and bottom edges
     * of the PanedWindow and its children.
     *)
    ShortResource(# resourceName:< XmNmarginHeight #);
spacing:
    (* Specifies the distance between each child pane. *)
    ShortResource(# resourceName:< XmNspacing #);
refigureMode:
```



```

(* Determines whether the panes' positions are recomputed
 * and repositioned when programmatic changes are being made
 * to THIS(PanedWindow). Setting this resource to True
 * resets the children to their appropriate positions.
 *)
BooleanResource(# resourceName:< XmNrefigureMode #);

separatorOn:
(* Determines whether a separator is created between each
 * of the panes. Setting this resource to True creates a
 * Separator at the midpoint between each of the panes.
 *)
BooleanResource(# resourceName:< XmNseparatorOn #);

sashIndent:
(* Specifies the horizontal placement of the sash along
 * each pane. A positive value causes the sash to be offset
 * from the near (left) side of THIS(PanedWindow), and a
 * negative value causes the sash to be offset from the far
 * (right) side of THIS(PanedWindow). If the offset is
 * greater than the width of THIS(PanedWindow) minus the
 * width of the sash, the sash is placed flush against the
 * near side of THIS(PanedWindow). Whether the placement
 * actually corresponds to the left or right side of the
 * PanedWindow may depend on the value of the
 * stringDirection resource.
 *)
ShortResource(# resourceName:< XmNsashIndent #);

sashWidth:
(* Specifies the width of the sash. *)
ShortResource(# resourceName:< XmNsashWidth #);

sashHeight:
(* Specifies the height of the sash. *)
ShortResource(# resourceName:< XmNsashHeight #);

sashShadowThickness:
(* Specifies the thickness of the shadows of the sashes. *)
ShortResource(# resourceName:< XmNsashShadowThickness #);

<<SLOT PanedWindowLib: attributes>>
#) (* PanedWindow *);

(* Alias used in CompositeLib *)
mPanedWindow: PanedWindow (# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
 * the default father of the widget
 *)
PanedWindow: mPanedWindow
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)
#)

```

---



# Selectionbox Interface

```
ORIGIN 'bulletinboard';
BODY 'private/selectionboxbody';

(*
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

SelectionBox: BulletinBoard
(* A SelectionBox is a general dialog widget that allows the
 * user to select one item from a list.  A SelectionBox includes
 * the following:
 *
 * + A scrolling list of alternatives
 *
 * + An editable text field for the selected alternative
 *
 * + Labels for the list and text field
 *
 * + Three or four buttons The default button labels are OK,
 *   Cancel, and Help.  By default an Apply button is also created;
 *   if the parent of the SelectionBox is a DialogShell it is
 *   managed, and otherwise it is unmanaged.  One additional
 *   WorkArea child may be added to the SelectionBox after
 *   creation.  The user can select an item in two ways: by
 *   scrolling through the list and selecting the desired item or
 *   by entering the item name directly into the text edit area.
 *   Selecting an item from the list causes that item name to
 *   appear in the selection text edit area.  The user may select a
 *   new item as many times as desired.  The item is not actually
 *   selected until the user presses the OK PushButton.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmSelectionBoxWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
textAccelerators:
    (* Specifies translations added to the Text widget child of
     * THIS(SelectionBox).  The default includes bindings for
     * the up and down keys for auto selection of list
     * items. This resource is ignored if accelerators is
     * initialized to a nondefault value.
     *)
    IntegerResource(# resourceName::< XmNtextAccelerators #);
selectionLabelString:
    (* Specifies the string label for the selection text edit
     * field.
     *)
    MotifStringResource
        (# resourceName::< XmNselectionLabelString #);
listLabelString:
    (* Specifies the string label to appear above the
     * SelectionBox list containing the selection items.
```



```

    *)
    MotifStringResource(# resourceName::< XmNlistLabelString #);
textColumns:
    (* Specifies the number of columns in the Text widget. *)
    ShortResource(# resourceName::< XmNtextColumns #);
textString:
    (* Specifies the text in the text edit selection field. *)
    MotifStringResource(# resourceName::< XmNtextString #);
listItems: MotifStringArrayResource
    (* Specifies the items in the SelectionBox list. *)
    (# resourceName::< XmNlistItems;
     counterName::< XmNlistItemCount;
    #);
listItemCount:
    (* Specifies the number of items in the SelectionBox list.
    *)
    IntegerResource(# resourceName::< XmNlistItemCount #);
listVisibleItemCount:
    (* Specifies the number of items displayed in the
    * SelectionBox list.
    *)
    IntegerResource(# resourceName::< XmNlistVisibleItemCount #);
okLabelString:
    (* Specifies the string label for the OK button. *)
    MotifStringResource(# resourceName::< XmNokLabelString #);
cancelLabelString:
    (* Specifies the string label for the Cancel button. *)
    MotifStringResource(# resourceName::< XmNcancelLabelString #);
helpLabelString:
    (* Specifies the string label for the Help button. *)
    MotifStringResource(# resourceName::< XmNhelpLabelString #);
applyLabelString:
    (* Specifies the string label for the Apply button. *)
    MotifStringResource(# resourceName::< XmNapplyLabelString #);
mustMatch:
    (* Specifies whether THIS(SelectionBox) should check if the
    * user's selection in the text edit field has an exact
    * match in the SelectionBox list when the OK button is
    * activated. If the selection does not have an exact
    * match, and mustMatch is True, the noMatchCallback
    * callback is called. If the selection does have an exact
    * match or if mustMatch is False, the okCallback callback
    * is called.
    *)
    BooleanResource(# resourceName::< XmNmustMatch #);
minimizeButtons:
    (* Sets the buttons to the width of the widest button and
    * height of the tallest button if False. If True, button
    * width and height are not modified.
    *)
    BooleanResource(# resourceName::< XmNminimizeButtons #);
dialogType:
    (* Determines the set of SelectionBox children widgets that
    * are created and managed at initialization. The following
    * are possible values:
    *
    * + XmDIALOG_PROMPT - all standard children except the
    *   list and list label are created, and all except the
    *   Apply button are managed.
    *
    * + XmDIALOG_COMMAND - only the list, the selection label,
    *   and the text field are created and managed.
    *
    * + XmDIALOG_SELECTION - all standard children are created
    *   and managed.
    *)

```



```

* + XmDIALOG_FILE_SELECTION - all standard children are
*   created and managed.
*
* + XmDIALOG_WORK_AREA - all standard children are
*   created, and all except the Apply button are managed.
*   If the parent of THIS(SelectionBox) is a DialogShell,
*   the default is XmDIALOG_SELECTION; otherwise, the
*   default is XmDIALOG_WORK_AREA. This resource cannot
*   be modified after creation.
*)
CharResource(# resourceName:< XmNdialogType #);

(* Utility patterns *)
getChild: IntegerValue
(* Used to access a component within THIS(SelectionBox).
* The child parameter specifies a component within
* THIS(SelectionBox). The following are legal values for
* this parameter:
*
* + XmDIALOG_APPLY_BUTTON
*
* + XmDIALOG_CANCEL_BUTTON
*
* + XmDIALOG_DEFAULT_BUTTON
*
* + XmDIALOG_HELP_BUTTON
*
* + XmDIALOG_LIST
*
* + XmDIALOG_LIST_LABEL
*
* + XmDIALOG_OK_BUTTON
*
* + XmDIALOG_SELECTION_LABEL
*
* + XmDIALOG_SEPARATOR
*
* + XmDIALOG_TEXT
*
* + XmDIALOG_WORK_AREA
*)
(# child: @integer;
enter child
do ...
#);

(* Callbacks *)
SelectionBoxCallback: MotifCallback
(* Prefix for SelectionBox callbacks *)
(# callData:< XmSelectionBoxCallbackStruct do INNER #);

applyCallback:< SelectionBoxCallback
(* Called when the user activates the Apply button. The
* callback reason is XmCR_APPLY.
*);
cancelCallback:< SelectionBoxCallback
(* Called when the user activates the Cancel button. The
* callback reason is XmCR_CANCEL.
*);
okCallback:< SelectionBoxCallback
(* Called when the user activates the OK button. The
* callback reason is XmCR_OK. If the selection text does
* not match a list item, and mustMatch is True, the
* noMatchCallback callback is called instead.
*);
noMatchCallback:< SelectionBoxCallback

```



```

    (* Called when the user makes a selection from the text
    * edit field that does not have an exact match with any of
    * the items in the list box. The callback reason is
    * XmCR_NO_MATCH. Called only if mustMatch is true
    * );
    (* Inherited callbacks *)
helpCallback::< SelectionBoxCallback;
focusCallback::< SelectionBoxCallback;
mapCallback::< SelectionBoxCallback;
unmapCallback::< SelectionBoxCallback;

installCallbacks::< (* Private *)
    (# do ...; INNER #);

<<SLOT SelectionBoxLib: attributes>>
#) (* SelectionBox *);

PromptDialog: SelectionBox
(* A PromptDialog consists of a DialogShell and an unmanaged
* SelectionBox child of the DialogShell. A PromptDialog prompts
* the user for text input. It includes a message, a text input
* region, and three managed buttons. The default button labels
* are OK, Cancel, and Help. An additional button, with Apply as
* the default label, is created unmanaged; it may be explicitly
* managed if needed. One additional WorkArea child may be added
* to the SelectionBox after creation.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

    <<SLOT PromptDialogLib: attributes>>
#) (* PromptDialog *);

SelectionDialog: SelectionBox
(* A SelectionDialog consists of a DialogShell and an unmanaged
* SelectionBox child of the DialogShell. A SelectionDialog
* offers the user a choice from a list of alternatives and gets
* a selection. It includes the following:
*
* + A scrolling list of alternatives
*
* + An editable text field for the selected alternative
*
* + Labels for the text field
*
* + Four buttons The default button labels are OK, Cancel,
*   Apply, and Help. One additional WorkArea child may be added
*   to the SelectionBox after creation.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

    <<SLOT SelectionDialogLib: attributes>>
#) (* SelectionDialog *);

(* Aliases used in CompositeLib *)
mSelectionBox: SelectionBox(# #);
mPromptDialog: PromptDialog(# #);
mSelectionDialog: SelectionDialog(# #);

--- CompositeLib: attributes ---

```



```

(* Redefinitions of widgets within composites making the composite
 * the default father of the widgets
 *)
SelectionBox: mSelectionBox
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
PromptDialog: mPromptDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
SelectionDialog: mSelectionDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Selectionbox Interface

[Mjølner Informatics](#)



# Fileselectionbox Interface

```
ORIGIN 'selectionbox';
BODY 'private/fileselectionboxbody';

(*
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

FileSelectionBox: SelectionBox
(* A FileSelectionBox traverses through directories, views the
 * files and subdirectories in them, and then selects files.
 *
 * A FileSelectionBox has five main areas:
 *
 * + A text input field for displaying and editing a directory
 *   mask used to select the files to be displayed
 *
 * + A scrollable list of filenames
 *
 * + A scrollable list of subdirectories
 *
 * + A text input field for displaying and editing a filename
 *
 * + A group of PushButtons, labeled OK, Filter, Cancel, and
 *   Help
 *
 * One additional WorkArea child may be added to the
 * FileSelectionBox after creation. The list of filenames, the
 * list of subdirectories, or both can be removed from the
 * FileSelectionBox after creation by using the patterns
 * unmanageFileNames and unmanageSubdirectories.
 *
 * The directory mask is a string specifying the base directory
 * to be examined and a search pattern. Ordinarily, the
 * directory list displays the subdirectories of the base
 * directory, as well as the base directory itself and its parent
 * directory. The file list ordinarily displays all files and/or
 * subdirectories in the base directory that match the search
 * pattern.
 *
 * The user can select a new directory to examine by scrolling
 * through the list of directories and selecting the desired
 * directory or by editing the directory mask. Selecting a new
 * directory from the directory list does not change the search
 * pattern. A user can select a new search pattern by editing
 * the directory mask. Double clicking on a directory in the
 * directory list initiates a search for files and subdirectories
 * in the new directory, using the current search pattern.
 *
 * The user can select a file by scrolling through the list of
 * filenames and selecting the desired file or by entering the
 * filename directly into the text edit area. Selecting a file
 * from the list causes that filename to appear in the file
 * selection text edit area.
 *
 * The user may select a new file as many times as desired. The
 * application is not notified until the user takes one of these
 * actions:
 *)
```



```

* + Selects the OK PushButton
*
* + Double clicks on an item in the file list The keyboard may
*   also be used to notify the application, see the relevant Motif
*   documentation for the machine used.
*
* A FileSelectionBox initiates a directory and file search when
* any of the following occurs:
*
* + The FileSelectionBox is initialized
*
* + The resources dirMask, directory, pattern, or fileTypeMask
*   are changed
*
* + The user activates the Filter PushButton
*
* + The user double clicks on an item in the directory list
*
* + The application invokes doSearch The keyboard may also be
*   used to initiate a search, see the relevant Motif
*   documentation for the machine used.
*
* When a file search is initiated, the FileSelectionBox takes
* the following actions:
*
* + Constructs an XmFileSelectionBoxCallbackStruct structure
*   with values appropriate for the action that initiated the
*   search
*
* + Calls the qualifySearchDataProc with the callback structure
*   as the data input argument
*
* + Sets directoryValid and listUpdated to False
*
* + Calls the dirSearchProc with the qualified data returned by
*   the qualifySearchDataProc
*
* If directoryValid is True, the FileSelectionBox takes these
* additional actions:
*
* + Sets listUpdated to False
*
* + Calls the fileSearchProc with the qualified data returned
*   by the qualifySearchDataProc
*
* + If listUpdated is True and the file list is empty, displays
*   the noMatchString in the file list and clears the selection
*   text and dirSpec
*
* + If listUpdated is True and the file list is not empty, sets
*   the selection text and dirSpec to the qualified dir
*   returned by the qualifySearchDataProc
*
* + Sets the directory mask text and dirMask to the qualified
*   mask returned by the qualifySearchDataProc
*
* + Sets directory to the qualified dir returned by the
*   qualifySearchDataProc
*
* + Sets pattern to the qualified pattern returned by the
*   qualifySearchDataProc
*)
(# init::<
  (# WidgetClass::<
    (#
      do INNER;

```



```

        (if value=0 then
            xmFileSelectionBoxWidgetClass->value
        if)
    #)
do INNER
#);

(* Resources *)
directory:
(* Specifies the base directory used in combination with
 * the pattern resource in determining the files and
 * directories to be displayed. The default value is
 * determined by the qualifySearchDataProc resource and
 * depends on the initial values of the dirMask, directory,
 * and pattern resources. If the default is empty, the
 * current working directory is used.
 *)
MotifStringResource(# resourceName:< XmNdirectory #);
directoryValid:
(* Specifies an attribute that is set only by the directory
 * search procedure. The value is set to True if the
 * directory passed to the directory search procedure can
 * actually be searched. If this value is False the file
 * search procedure is not called, and the dirMask,
 * directory, and pattern resources are not changed.
 *)
BooleanResource(# resourceName:< XmNdirectoryValid #);
dirListItems: MotifStringArrayResource
(* Specifies the items in the directory list. *)
(# resourceName:< XmNdirListItems;
 counterName:< XmNdirListItemCount;
 #);
dirListItemCount:
(* Specifies the number of items in the directory list. *)
IntegerResource(# resourceName:< XmNdirListItemCount #);
dirListLabelString:
(* Specifies the label string of the directory list. *)
MotifStringResource(# resourceName:< XmNdirListLabelString #);
dirMask:
(* Specifies the directory mask used in determining the
 * files and directories to be displayed. The default value
 * is determined by the qualifySearchDataProc and depends on
 * the initial values of the dirMask, directory, and pattern
 * resources.
 *)
MotifStringResource(# resourceName:< XmNdirMask #);
dirSearchProc:
(* Specifies a directory search procedure to replace the
 * default directory-search procedure. FileSelectionBox's
 * default directory-search procedure fulfills the needs of
 * most applications. Because it is impossible to cover the
 * requirements of all applications, you can replace the
 * default search procedure. The directory search procedure
 * is called with two arguments: the FileSelectionBox widget
 * and a pointer to an XmFileSelectionBoxCallbackStruct
 * structure. The callback structure is generated by the
 * qualifySearchDataProc and contains all information
 * required to conduct a directory search, including the
 * directory mask and a qualified base directory and search
 * pattern. Once called, it is up to the search routine to
 * generate a new list of directories and update the
 * FileSelectionBox widget by using XtSetValues. The search
 * procedure must set the directoryValid and listUpdated
 * resources. If it generates a new list of directories, it
 * must also set the dirListItems and dirListItemCount
 * resources. If the search procedure cannot search the

```



```

* specified directory, it must warn the user and set the
* directoryValid and listUpdated resources to False, unless
* it prompts and subsequently obtains a valid directory.
* If the directory is valid but is the same as the current
* directory, the search procedure must set the
* directoryValid resource to True, but it may elect not to
* generate a new list of directories. In this case it must
* set the listUpdated resource to False.
*
* If the search procedure generates a new list of
* directories, it must set the dirListItems resource to the
* new list of directories and the dirListItemCount resource
* to the number of items in the list. If there are no
* directories, it sets the dirListItems resource to NULL
* and the dirListItemCount resource to 0. In either case
* it must set the directoryValid and listUpdated resources
* to True. The search procedure ordinarily should not
* change the callback struct. But if the original
* directory is not valid, the search procedure may obtain a
* new directory from the user. In this case it should set
* the dir member of the callback struct to the new
* directory, call the qualifySearchDataProc with the
* callback struct as the input argument, and copy the
* qualified data returned by the qualifySearchDataProc into
* the callback struct.
*)
ProcResource(# resourceName::< XmNdirSearchProc #);
dirSpec:
(* Specifies the full file path specification. This
* resource overrides the textString resource in
* SelectionBox. The default value is determined by the
* FileSelectionBox after conducting the initial directory
* and file search.
*)
MotifStringResource(# resourceName::< XmNdirSpec #);
fileListItems: MotifStringArrayResource
(* Specifies the items in the file list. This resource
* overrides the listItems resource in SelectionBox.
*)
(# resourceName::< XmNfileListItems;
  counterName::< XmNfileListItemCount;
#);
fileListItemCount:
(* Specifies the number of items in the file list. This
* resource overrides the listItemCount resource in
* SelectionBox.
*)
IntegerResource(# resourceName::< XmNfileListItemCount #);
fileListLabelString:
(* Specifies the label string of the file list. This
* resource overrides the listLabelString resource in
* SelectionBox.
*)
MotifStringResource
  (# resourceName::< XmNfileListLabelString #);
fileSearchProc:
(* Specifies a file search procedure to replace the default
* file-search procedure. FileSelectionBox's default
* file-search procedure fulfills the needs of most
* applications. Because it is impossible to cover the
* requirements of all applications, you can replace the
* default search procedure. The file search procedure is
* called with two arguments:
* THIS(FileSelectionBox).thewidget and a pointer to an
* XmFileSelectionBoxCallbackStruct structure. The callback
* structure is generated by the qualifySearchDataProc and

```



```

* contains all information required to conduct a file
* search, including the directory mask and a qualified base
* directory and search pattern. Once called, it is up to
* the search routine to generate a new list of files and
* update the FileSelectionBox widget by using XtSetValues.
* The search procedure must set the listUpdated resource.
* If it generates a new list of files, it must also set the
* fileListItemCount and fileListItemCount resources. The
* search procedure is recommended always to generate a new
* list of files. If the mask member of the callback struct
* is the same as the mask member of the callback struct in
* the preceding call to the search procedure, the procedure
* may elect not to generate a new list of files. In this
* case it must set the listUpdated resource to False. If
* the search procedure generates a new list of files, it
* must set the fileListItemCount resource to the new list of
* files and the fileListItemCount resource to the number of
* items in the list. If there are no files, it sets
* fileListItemCount to NULL and fileListItemCount to 0. In
* either case it must set the listUpdated resource to True.
* In constructing the list of files, the search procedure
* should include only files of the types specified by
* THIS(FileSelectionBox).fileTypeMask. Setting the dirSpec
* resource is optional, but recommended. Set this
* attribute to the full file specification of the directory
* searched. The directory specification is displayed below
* the directory and file lists.
*)
ProcResource(# resourceName:< XmNfileSearchProc #);
fileTypeMask:
(* Specifies the type of files listed in the file list.
* Following are the possible values:
*
* + XmFILE_REGULAR restricts the file list to contain only
*   regular files.
*
* + XmFILE_DIRECTORY restricts the file list to contain
*   only directories.
*
* + XmFILE_ANY_TYPE allows the list to contain all file
*   types including directories.
*)
CharResource(# resourceName:< XmNfileTypeMask #);
filterLabelString:
(* Specifies the label string for the text entry field for
* the directory mask
*)
MotifStringResource(# resourceName:< XmNfilterLabelString #);
listUpdated:
(* Specifies an attribute that is set only by the directory
* and file search procedures. Set to True if the search
* procedure updated the directory or file list.
*)
BooleanResource(# resourceName:< XmNlistUpdated #);
noMatchString:
(* Specifies a string to be displayed in the file list if
* the list of files is empty.
*)
MotifStringResource(# resourceName:< XmNnoMatchString #);
pattern:
(* Specifies the search pattern used in combination with
* directory in determining the files and directories to be
* displayed. The default value is determined by the
* qualifySearchDataProc and depends on the initial values
* of the dirMask, directory, and pattern resources. If the
* default is empty, a pattern that matches all files is

```



```

* used.
*)
MotifStringResource(# resourceName::< XmNpattern #);
qualifySearchDataProc:
(* Specifies a search data qualification procedure to
* replace the default data qualification procedure.
* THIS(FileSelectionBox)'s default data qualification
* procedure fulfills the needs of most applications.
* Because it is impossible to cover the requirements of all
* applications, you can replace the default procedure. The
* data qualification procedure is called to generate a
* qualified directory mask, base directory, and search
* pattern for use by the directory and file search
* procedures. It is called with three arguments:
* THIS(FileSelectionBox).thewidget and pointers to two
* XmFileSelectionBoxCallbackStruct structures. The first
* callback struct contains the input data. The second
* callback struct contains the output data, to be filled in
* by the data qualification procedure. If the input dir
* and pattern members are not NULL, the procedure must copy
* them to the corresponding members of the output callback
* struct. If the input dir is NULL, the procedure
* constructs the output dir as follows: If the input mask
* member is NULL, the procedure uses
* THIS(FileSelectionBox).directory as the output dir;
* otherwise, it extracts the output dir from the input
* mask. If the resulting output dir is empty, the
* procedure uses the current working directory instead. If
* the input pattern is NULL, the procedure constructs the
* output pattern as follows: If the input mask member is
* NULL, the procedure uses the
* THIS(FileSelectionBox).pattern as the output pattern;
* otherwise, it extracts the output pattern from the input
* mask. If the resulting output pattern is empty, the
* procedure uses a pattern that matches all files instead.
* The data qualification procedure constructs the output
* mask from the output dir and pattern. The procedure must
* ensure that the output dir, pattern, and mask are fully
* qualified. If the input value member is not NULL, the
* procedure must copy it to the output value member;
* otherwise, the procedure must copy
* THIS(FileSelectionBox).dirSpec to the output value. The
* data qualification procedure must calculate the lengths
* of the output value, mask, dir, and pattern members and
* must fill in the corresponding length members of the
* output callback struct. The data qualification procedure
* must copy the input reason and event members to the
* corresponding output members.
*)
ProcResource(# resourceName::< XmNqualifySearchDataProc #);

(* Utility patterns *)
getChild:
(* XmFileSelectionBoxGetChild is used to access a
* component within a FileSelectionBox. The parameters
* given to the function are the FileSelectionBox widget and
* a value indicating which child to access. The child
* parameter specifies a component within
* THIS(FileSelectionBox). The following are legal values
* for this parameter:
*
* + XmDIALOG_APPLY_BUTTON
*
* + XmDIALOG_CANCEL_BUTTON
*
* + XmDIALOG_DEFAULT_BUTTON

```



```

*
* + XmDIALOG_DIR_LIST
*
* + XmDIALOG_DIR_LIST_LABEL
*
* + XmDIALOG_FILTER_LABEL
*
* + XmDIALOG_FILTER_TEXT
*
* + XmDIALOG_HELP_BUTTON
*
* + XmDIALOG_LIST
*
* + XmDIALOG_LIST_LABEL
*
* + XmDIALOG_OK_BUTTON
*
* + XmDIALOG_SELECTION_LABEL
*
* + XmDIALOG_SEPARATOR
*
* + XmDIALOG_TEXT
*
* + XmDIALOG_WORK_AREA
*)
(# child: @integer;
enter child
do ...
#);

unmanageFileNames:
(* Unmanages (removes) the fileList child. Useful if a
* FileSelectionBox without the list of filenames is wanted.
*)
(# do ... #);

unmanageSubdirectories
(* Unmanages (removes) the dirList child. Useful if a
* FileSelectionBox without the list of directories is
* wanted.
*)
(# do ... #);

unManageHelp:
(* Unmanages (removes) the Help button. Useful if a
* FileSelectionBox without the Help button is wanted.
*)
(# do ... #);

doSearch:
(* initiates a directory and file search in
* THIS(FileSelectionBox). For a description of the actions
* that THISFileSelectionBox) takes when doing a search, see
* above. The dirmask parameter specifies the directory
* mask used in determining the directories and files
* displayed in the FileSelectionBox lists. This value is
* used as the mask member of the input data
* XmFileSelectionBoxCallbackStruct structure passed to the
* THIS(FileSelectionBox).qualifySearchDataProc. The dir
* and pattern members of that structure are NULL.
*)
(# dirMask: @MotifString;
enter dirMask
do ...
#);

(* Callbacks *)
FileSelectionBoxCallback SelectionBoxCallback
(* Prefix for callbacks to THIS(FileSelectionBox) *)
(# callData::< XmFileSelectionBoxCallbackStruct do INNER #);

```



```

(* Inherited callbacks *)
helpCallback::< FileSelectionBoxCallback;
focusCallback::< FileSelectionBoxCallback;
mapCallback::< FileSelectionBoxCallback;
unmapCallback::< FileSelectionBoxCallback;
applyCallback::< FileSelectionBoxCallback;
cancelCallback::< FileSelectionBoxCallback;
okCallback::< FileSelectionBoxCallback;
noMatchCallback::< FileSelectionBoxCallback;

<<SLOT FileSelectionBoxLib: attributes>>
#) (* FileSelectionBox *);

FileSelectionDialog: FileSelectionBox
(* A FileSelectionDialog consists of a DialogShell and an
 * unmanaged FileSelectionBox child of the DialogShell. A
 * FileSelectionDialog selects a file. It includes the
 * following:
 *
 * + An editable text field for the directory mask
 *
 * + A scrolling list of filenames
 *
 * + An editable text field for the selected file
 *
 * + Labels for the list and text fields
 *
 * + Four buttons The default button labels are: OK, Filter,
 *   Cancel, and Help. One additional WorkArea child may be added
 *   to the FileSelectionBox after creation.
 *)
(# init::< (# WidgetClass::< (# do ... #);
  do true -> doNotManageChild;
  INNER
  #);

  <<SLOT FileSelectionDialogLib: attributes>>
#) (* FileSelectionDialog *);

(* Aliases used in CompositeLib *)
mFileSelectionBox: FileSelectionBox (# #);
mFileSelectionDialog: FileSelectionDialog(# #);

--- CompositeLib: attributes ---

(* Redefinitions of widgets within composites making the composite
 * the default father of the widgets
 *)
FileSelectionBox: mFileSelectionBox
  (# init::<(# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
FileSelectionDialog: mFileSelectionDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->Value #);
    do INNER
    #)
  #)

```

---



# Command Interface

```
ORIGIN 'selectionbox';
BODY 'private/commandbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

Command: SelectionBox
(* A Command is a special-purpose composite widget for command
 * entry that provides a built-in command-history mechanism.
 * Command includes a command-line text-input field, a
 * command-line prompt, and a command- history list region. One
 * additional WorkArea child may be added to the Command after
 * creation. Whenever a command is entered, it is automatically
 * added to the end of the command-history list and made visible.
 * This does not change the selected item in the list, if there
 * is one. Many of the new resources specified for Command are
 * actually SelectionBox resources that have been renamed for
 * clarity and ease of use.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmCommandWidgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
command:
    (* Contains the current command-line text. This is the
     * textString resource in SelectionBox, renamed for
     * Command. This resource can also be modified via the
     * setValue and appendValue patterns. The command area is a
     * Text widget.
     *)
    MotifStringResource(# resourceName::< XmNcommand #);
historyItems: MotifStringArrayResource
    (* The items that make up the contents of the history list.
     * This is the listItems resource in SelectionBox, renamed
     * for Command.
     *)
    (# resourceName::< XmNhistoryItems;
        countername::< XmNhistoryItemCount;
    #);
historyItemCount:
    (* Specifies the number of MotifStrings in historyItems.
     * This is the listItemCount resource in SelectionBox,
     * renamed for Command.
     *)
    IntegerResource(# resourceName::< XmNhistoryItemCount #);
historyMaxItems:
    (* Specifies the maximum number of items allowed in the
     * history list. Once this number is reached, an existing
     * list item must be removed before a new item can be added
     * to the list. For each command entered, the first list
     * item is removed from the list, so the new command can be
```



```

    * added to the list.
    *)
IntegerResource(# resourceName:< XmNhistoryMaxItems #);
historyVisibleItemCount:
(* Specifies the number of items in the history list that
 * should be visible at one time. In effect, it sets the
 * height (in lines) of the history list window. This is
 * the visibleItemCount resource in SelectionBox, renamed
 * for Command.
 *)
IntegerResource
    (# resourceName:< XmNhistoryVisibleItemCount #);
promptString:
(* Specifies a prompt for the command line. This is the
 * selectionLabelString resource in SelectionBox, renamed
 * for Command. The default may vary depending on the value
 * of the stringDirection resource.
 *)
MotifStringResource(# resourceName:< XmNpromptString #);

(* Utility patterns *)
getChild: IntegerValue
(* Exit a component of THIS(Command). This child argument
 * is used to determine which child to access; it can be,
 * XmDIALOG_COMMAND_TEXT, XmDIALOG_PROMPT_LABEL, or
 * XmDIALOG_HISTORY_LIST
 *)
(# child: @integer;
enter child
do ...
#);
setValue:
(* Replaces the string displayed in the command area of
 * THIS(Command) with the passed MotifString
 *)
(# command: @MotifString
enter command
do ...
#);
appendValue:
(* Appends the passed MotifString to the end of the string
 * displayed in the command area of THIS(Command)
 *)
(# command: @MotifString
enter command
do ...
#);
error:
(* Displays an error message in the history area of
 * THIS(Command). The MotifString error is displayed until
 * the next command entered occurs.
 *)
(# error: @MotifString
enter error
do ...
#);

(* Callbacks *)
CommandCallback: SelectionBoxCallback
(* Prefix for Command callbacks *)
(# callData:< XmCommandCallbackStruct do INNER #);

commandEnteredCallback < CommandCallback
(* Called when a command is entered in THIS(Command). The
 * callback reason is XmCR_COMMAND_ENTERED.
 *)

```



```

commandChangedCallback < CommandCallback
(* Called when the value of THIS(Command) changes. The
 * callback reason is XmCR_COMMAND_CHANGED. This is
 * equivalent to the valueChangedCallback of the Text
 * widget, except that the callData is an
 * XmCommandCallbackStruct, and the structure's value member
 * contains the MotifString.
 *);
(* Inherited callbacks *)
helpCallback:< CommandCallback;
focusCallback:< CommandCallback;
mapCallback:< CommandCallback;
unmapCallback:< CommandCallback;
applyCallback:< CommandCallback;
cancelCallback:< CommandCallback;
okCallback:< CommandCallback;
noMatchCallback:< CommandCallback;

installCallbacks:< (* Private *)
  (# do ...; INNER #);

<<SLOT CommandLib: attributes>>
#) (* Command *);

(* Alias used in CompositeLib *)
mCommand: Command (# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
 * the default father of the widget
 *)
Command: mCommand
  (# init:< (# GetFatherWidget:< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)
#)

```

---

Command Interface

[Mjølner Informatics](#)



# Messagebox Interface

```
ORIGIN 'bulletinboard';
BODY 'private/messageboxbody';
```

```
(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)
```

```
-- XtEnvLib: attributes --
```

**MessageBox:** BulletinBoard

```
(* MessageBox is a dialog pattern used for creating simple
 * message dialogs. Specializations based on MessageBox are
 * provided for several common interaction tasks, which include
 * giving information, asking questions, and reporting errors. A
 * MessageBox dialog is typically transient in nature, displayed
 * for the duration of a single interaction. MessageBox is a
 * subpattern of BulletinBoard and depends on it for much of its
 * general dialog behavior. A MessageBox can contain a message
 * symbol, a message, and up to three standard default
 * PushButtons: OK, Cancel, and Help. It is laid out with the
 * symbol and message on top and the PushButtons on the bottom.
 * The help button is positioned to the side of the other push
 * buttons. You can localize the default symbols and button
 * labels for MessageBox convenience dialogs.
 *)
```

```
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmMessageBoxWidgetClass->value if)
        #)
    do INNER
    #);
```

```
(* Resources *)
```

**dialogType:**

```
(* Specifies the type of MessageBox dialog, which
 * determines the default message symbol. The following are
 * the possible values for this resource:
```

```
*
 * + XmDIALOG_ERROR - indicates an ErrorDialog
 *
 * + XmDIALOG_INFORMATION - indicates an InformationDialog
 *
 * + XmDIALOG_MESSAGE - indicates a MessageDialog. This is
 * the default MessageBox dialog type. The default
 * message symbol is empty.
 *
 * + XmDIALOG_QUESTION - indicates a QuestionDialog
 *
 * + XmDIALOG_WARNING - indicates a WarningDialog
 *
 * + XmDIALOG_WORKING - indicates a WorkingDialog If this
 * resource is changed, the symbol bitmap is modified to the
 * new dialog type bitmap unless the symbolPixmap resource
 * is also changed.
 *)
```

```
CharResource(# resourceName::< XmNdialogType #);
```

**minimizeButtons:**

```
(* Sets the buttons to the width of the widest button and
```



```

    * height of the tallest button if False.  If True, button
    * width and height are set to the preferred size of each
    * button.
    *)
BooleanResource(# resourceName::< XmNminimizeButtons #);
defaultButtonType:
(* Specifies the default PushButton.  The following are
 * valid types:
 *
 * + XmDIALOG_CANCEL_BUTTON
 *
 * + XmDIALOG_OK_BUTTON
 *
 * + XmDIALOG_HELP_BUTTON
 *)
CharResource(# resourceName::< XmNdefaultButtonType #);
messageString:
(* Specifies the string to be used as the message. *)
MotifStringResource(# resourceName::< XmNmessageString #);
messageAlignment:
(* Controls the alignment of the message Label.  Possible
 * values include the following:
 *
 * + XmALIGNMENT_BEGINNING - the default
 *
 * + XmALIGNMENT_CENTER
 *
 * + XmALIGNMENT_END
 *)
CharResource(# resourceName::< XmNmessageAlignment #);
symbolPixmap:
(* Specifies the pixmap label to be used as the message
 * symbol.
 *)
IntegerResource(# resourceName::< XmNsymbolPixmap #);
okLabelString:
(* Specifies the string label for the OK button. *)
MotifStringResource(# resourceName::< XmNokLabelString #);
cancelLabelString:
(* Specifies the string label for the Cancel button. *)
MotifStringResource(# resourceName::< XmNcancelLabelString #);
helpLabelString:
(* Specifies the string label for the help button. *)
MotifStringResource(# resourceName::< XmNhelpLabelString #);

(* Utility patterns *)
unManageCancel:
(* Unmanages (removes) the Cancel button. Useful if a
 * MessageBox with only an OK button is wanted.
 *)
(# do ... #);
unManageHelp:
(* Unmanages (removes) the Help button. Useful if a
 * MessageBox with only an OK button is wanted.
 *)
(# do ... #);

(* Callbacks *)
okCallback::< MotifCallback
(* Specifies the list of callbacks that is called when the
 * user clicks on the OK button.  The reason sent by the
 * callback is XmCR_OK.
 *);
cancelCallback::< MotifCallback
(* Called when the user clicks on the Cancel button.  The
 * reason sent by the callback is XmCR_CANCEL.

```



```

    *);
    (* Inherited callbacks *)
helpCallback:< MotifCallback;
focusCallback:< MotifCallback;
mapCallback:< MotifCallback;
unmapCallback:< MotifCallback;

installCallbacks:< (* Private *)
    (# do ...; INNER #);

<<SLOT MessageBoxLib: attributes>>
#) (* MessageBox *);

```

**ErrorDialog:** MessageBox

```

(* An ErrorDialog consists of a DialogShell and an unmanaged
 * MessageBox child of the DialogShell. An ErrorDialog warns the
 * user of an invalid or potentially dangerous condition. It
 * includes a symbol, a message, and three buttons. The default
 * symbol is an octagon with a diagonal slash. The default
 * button labels are OK, Cancel, and Help.
 *)
(# init:< (# WidgetClass:< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

<<SLOT ErrorDialogLib: attributes>>
#) (* ErrorDialog *);

```

**InformationDialog:** MessageBox

```

(* An InformationDialog consists of a DialogShell and an
 * unmanaged MessageBox child of the DialogShell. An
 * InformationDialog warns the user of an invalid or potentially
 * dangerous condition. It includes a symbol, a message, and
 * three buttons. The default symbol is lower case i. The
 * default button labels are OK, Cancel, and Help.
 *)
(# init:< (# WidgetClass:< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

<<SLOT InformationDialogLib: attributes>>
#) (* InformationDialog *);

```

**MessageDialog:** MessageBox

```

(* A MessageDialog consists of a DialogShell and an unmanaged
 * MessageBox child of the DialogShell. A MessageDialog warns
 * the user of an invalid or potentially dangerous condition. It
 * includes a symbol, a message, and three buttons. By default
 * there is no symbol. The default button labels are OK, Cancel,
 * and Help.
 *)
(# init:< (# WidgetClass:< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

<<SLOT MessageDialogLib: attributes>>
#) (* MessageDialog *);

```

**QuestionDialog:** MessageBox

```

(* A QuestionDialog consists of a DialogShell and an unmanaged
 * MessageBox child of the DialogShell. A QuestionDialog warns
 * the user of an invalid or potentially dangerous condition. It
 * includes a symbol, a message, and three buttons. The default

```



```

* symbol is a question mark. The default button labels are OK,
* Cancel, and Help.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

    <<SLOT QuestionDialogLib: attributes>>
#) (* QuestionDialog *);

```

**WarningDialog: MessageBox**

```

(* A WarningDialog consists of a DialogShell and an unmanaged
* MessageBox child of the DialogShell. A WarningDialog warns
* the user of an invalid or potentially dangerous condition. It
* includes a symbol, a message, and three buttons. The default
* symbol is an exclamation point. The default button labels are
* OK, Cancel, and Help.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

    <<SLOT WarningDialogLib: attributes>>
#) (* WarningDialog *);

```

**WorkingDialog: MessageBox**

```

(* A WorkingDialog consists of a DialogShell and an unmanaged
* MessageBox child of the DialogShell. A WorkingDialog warns
* the user of an invalid or potentially dangerous condition. It
* includes a symbol, a message, and three buttons. The default
* symbol is an hourglass. The default button labels are OK,
* Cancel, and Help.
*)
(# init::< (# WidgetClass::< (# do ... #);
    do true -> doNotManageChild;
    INNER
    #);

    <<SLOT WorkingDialogLib: attributes>>
#) (* WorkingDialog *);

```

```
(* Aliases used in CompositeLib *)
```

```

mMessageBox: MessageBox(# #);
mErrorDialog: ErrorDialog(# #);
mQuestionDialog: QuestionDialog(# #);
mInformationDialog: InformationDialog(# #);
mMessageDialog: MessageDialog(# #);
mWarningDialog: WarningDialog(# #);
mWorkingDialog: WorkingDialog(# #);

```

```
--- CompositeLib: attributes ---
```

```

(* Redefinitions of widgets within composites making the composite
* the default father of the widgets
*)

```

**MessageBox: mMessageBox**

```

(# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
#);

```

**ErrorDialog: mErrorDialog**

```

(# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)

```



```

#);
QuestionDialog: mQuestionDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
InformationDialog: mInformationDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
MessageDialog: mMessageDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
WarningDialog: mWarningDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #);
WorkingDialog: mWorkingDialog
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Messagebox Interface

[Mjølner Informatics](#)



# MenuShell Interface

```
ORIGIN 'basics';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

MenuShell: OverrideShell
(* The MenuShell widget is a custom OverrideShell widget.  An
 * OverrideShell widget bypasses the window manager when
 * displaying itself.  It is designed specifically to contain
 * Popup or Pulldown MenuPanes.  Most application writers never
 * encounter this widget if they use the patterns PopupMenu or
 * PulldownMenu, which automatically create a MenuShell widget as
 * the parent of the MenuPane.  However, if these patterns are
 * not used, the application programmer must create the required
 * MenuShell.  In this case, it is important to note that the
 * type of parent of the MenuShell depends on the type of menu
 * system being built.
 *
 * + If the MenuShell is for the top-level Popup MenuPane, the
 *   MenuShell must be created as a child of the widget from
 *   which the Popup MenuPane is popped up.
 *
 * + If the MenuShell is for a MenuPane that is pulled down from
 *   a Popup or another Pulldown MenuPane, the MenuShell must be
 *   created as a child of the Popup or Pulldown MenuPane.
 *
 * + If the MenuShell is for a MenuPane that is pulled down from
 *   a MenuBar, the MenuShell must be created as a child of the
 *   MenuBar.
 *
 * + If the MenuShell is for a Pulldown MenuPane in an
 *   OptionMenu, the MenuShell's parent must be the OptionMenu.
 *)
(# init::<
  (# WidgetClass::<
    (#
      do INNER;
      (if value=0 then xmMenuShellWidgetClass->value if)
    #)
    do INNER
  #);

  <<SLOT MenuShellLib: attributes>>
#) (* MenuShell *);

(* Alias used in CompositeLib *)
mMenuShell: MenuShell(# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
 * the default father of the widget
 *)
MenuShell: mMenuShell
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
  #)
```



# )

---

Menushell Interface

[Mjølner Informatics](#)



# Dialogshell Interface

```
ORIGIN 'basics';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

DialogShell: TransientShell
(* Modal and modeless dialogs use DialogShell as the Shell
 * parent. DialogShell widgets cannot be iconified. Instead,
 * all secondary DialogShell widgets associated with an
 * ApplicationShell widget are iconified and de-iconified as a
 * group with the primary widget. A client indirectly
 * manipulates a DialogShell via the different dialog patterns,
 * and it can directly manipulate its BulletinBoard-derived
 * child. Much of the functionality of DialogShell assumes that
 * its child is a BulletinBoard, although it can potentially
 * stand alone.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmDialogShellWidgetClass->value if)
        #)
        do INNER #);

    <<SLOT DialogShellLib: attributes>>
#) (* DialogShell *);

(* Alias used in CompositeLib *)
mDialogShell: DialogShell(# #);

--- CompositeLib: attributes ---

(* Redefinition of widget within composites making the composite
 * the default father of the widget
 *)
DialogShell: mDialogShell
(# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
#)

#)
```

---



# Vendorshell Interface

```
ORIGIN 'basics';

( *
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 * )

-- VendorShellLib: attributes --

( * VendorShell is a Motif widget class used as a supporting
 * superclass for all shell classes that are visible to the window
 * manager and that are not override redirect. It contains the
 * subresources that describe the MWM-specific look and feel (Motif
 * Window manager). It also manages the MWM-specific communication
 * needed by all VendorShell subclasses. Notice that a dummy
 * VendorShell widget is part of XtEnv, and that the attributes in
 * this fragment form are just add-ons for that widget.
 * )

defaultFontList: IntegerResource
( * Specifies a default font list for children of
 * THIS(VendorShell). This font list is used whenever a font
 * list is not specifically set for a Text, Label or Button child
 * of THIS(VendorShell).
 * )
( # resourceName::< XmNdefaultFontList # );

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

keyboardFocusPolicy: CharResource
( * Determines allocation of keyboard focus within the widget
 * hierarchy rooted at THIS(VendorShell). The X keyboard focus
 * must be directed to somewhere in the hierarchy for this
 * client-side focus management to take effect. Possible values
 * are XmEXPLICIT, specifying a click-to-type policy, and
 * XmPOINTER, specifying a pointer-driven policy.
 * )
( # resourceName::< XmNkeyboardFocusPolicy # );

mwmDecorations: IntegerResource
( * Includes the decoration flags (specific decorations to add or
 * remove from the window manager frame) for MWM_HINTS.
 * )
( # resourceName::< XmNmwmDecorations # );

mwmFunctions: IntegerResource
( * Includes the function flags (specific window manager
 * functions to include or exclude from the system menu) for
 * MWM_HINTS.
 * )
( # resourceName::< XmNmwmFunctions # );

mwmInputMode: IntegerResource
( * Includes the input mode flag (application modal or system
 * modal input focus constraints) for MWM_HINTS.
 * )
( # resourceName::< XmNmwmInputMode # );

mwmMenu: StringResource
```



```

(* Specifies the menu items that the Motif window manager should
 * add to the end of the system menu. The string contains a list
 * of items separated by \n with the following format:
 *
 *      label [mnemonic] [ accelerator] function
 *
 * If more than one item is specified, the items should be
 * separated by a newline character.
 *)
(# resourceName::< XmNmwmMenu #);
shellUnitType: CharResource
(* Determines geometric resource interpretation. The following
 * values are allowed:
 *
 * + XmPIXELS - all values provided to THIS(Primitive) are
 *   treated as normal pixel values.
 *
 * + Xm100TH_MILLIMETERS - all values provided to
 *   THIS(Primitive) are treated as 1/100 millimeter.
 *
 * + Xm1000TH_INCHES - all values provided to THIS(Primitive)
 *   are treated as 1/1000 inch.
 *
 * + Xm100TH_POINTS - all values provided to THIS(Primitive) are
 *   treated as 1/100 point. A point is a unit used in text
 *   processing applications and is defined as 1/72 inch.
 *
 * + Xm100TH_FONT_UNITS - all values provided to the widget are
 *   treated as 1/100 of a font unit. See the Motif
 *   documentation for details on using font units.
 *)
(# resourceName::< XmNshellUnitType #);
useAsyncGeometry: IntegerResource
(* Specifies whether the geometry manager should wait for
 * confirmation of a geometry request to the window manager.
 * When the value of this resource is True, the geometry manager
 * forces waitForWm to False and wmTimeout to 0, and it relies on
 * asynchronous notification. When the value of this resource is
 * False, waitForWm and wmTimeout are unaffected. The default is
 * False.
 *)
(# resourceName::< XmNuseAsyncGeometry #)

```

---

Vendorshell Interface

[Mjølner Informatics](#)



# Gadget Interface

```
ORIGIN 'basics';
BODY 'private/gadgetbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

Gadget: RectObj
(* Gadget is a pattern used as a supporting superpattern for
 * other gadget patterns. It handles shadow-border drawing and
 * highlighting, traversal activation and deactivation, and
 * various callbacks needed by gadgets. The color and pixmap
 * resources defined by Manager are directly used by gadgets. If
 * one of these resources for is changed for a Manager widget,
 * all of the gadget children within the Manager also change.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmGadgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
traversalOn:
    (* Specifies if traversal is activated for THIS(Gadget) *)
    BooleanResource(# resourceName::< XmNtraversalOn #);
highLightOnEnter:
    (* Specifies if the highlighting rectangle is drawn when
    * the cursor moves into THIS(Gadget). If the shell's focus
    * policy is XmEXPLICIT, this resource is ignored, and
    * THIS(Gadget) is highlighted when it has the focus. If
    * the shell's focus policy is XmPOINTER and if this
    * resource is True, the highlighting rectangle is drawn
    * when the cursor moves into THIS(Gadget). If the shell's
    * focus policy is XmPOINTER and if this resource is False,
    * the highlighting rectangle is not drawn when the the
    * cursor moves into THIS(Gadget). The default is False.
    *)
    BooleanResource(# resourceName::< XmNhighLightOnEnter #);
navigationType:
    (* Controls whether THIS(Gadget) is a navigation group.
    *
    * + XmNONE indicates that THIS(Gadget) is not a navigation
    *   group.
    *
    * + XmTAB_GROUP indicates that THIS(Gadget) is included
    *   automatically in keyboard navigation, unless
    *   XmAddTabGroup has been called.
    *
    * + XmSTICKY_TAB_GROUP indicates that THIS(Gadget) is
    *   included automatically in keyboard navigation, even if
    *   XmAddTabGroup has been called.
    *
    * + XmEXCLUSIVE_TAB_GROUP indicates that THIS(Gadget) is
    *   included explicitly in keyboard navigation by the
```



```

*      application.  With XmEXCLUSIVE_TAB_GROUP, traversal of
*      widgets within the group is based on the order of
*      children.  If THIS(Gadget)'s parent is a shell, the
*      default is XmTAB_GROUP; otherwise, the default is
*      XmNONE.
*)
IntegerResource(# resourceName::< XmNnavigationType #);
unitType:
(* Provides the basic support for resolution independence.
* It defines the type of units THIS(Gadget) uses with
* sizing and positioning resources.  If THIS(Gadget)'s
* parent is a specialization of Manager and if the unitType
* resource is not explicitly set, it defaults to the unit
* type of the parent widget.  If THIS(Gadget)'s parent is
* not a specialization of Manager, the resource has a
* default unit type of XmPIXELS unitType can have the
* following values:
*
* + XmPIXELS - all values provided to THIS(Gadget) are
*   treated as normal pixel values.
*
* + Xm100TH_MILLIMETERS - all values provided to
*   THIS(Gadget) are treated as 1/100 millimeter.
*
* + Xm1000TH_INCHES - all values provided to THIS(Gadget)
*   are treated as 1/1000 inch.
*
* + Xm100TH_POINTS - all values provided to THIS(Gadget)
*   are treated as 1/100 point.  A point is a unit used
*   in text processing applications and is defined as
*   1/72 inch.
*
* + Xm100TH_FONT_UNITS - all values provided to the widget
*   are treated as 1/100 of a font unit.  See the Motif
*   documentation for details on using font units.
*)
CharResource(# resourceName::< XmNunitType #);
highlightThickness:
(* Specifies the thickness of the highlighting rectangle. *)
ShortResource(# resourceName::< XmNhighlightThickness #);
shadowThickness:
(* Specifies the size of the drawn border shadow. *)
ShortResource(# resourceName::< XmNshadowThickness #);
userData:
(* Allows the application to attach any necessary specific
* data to THIS(Gadget).  This is an internally unused
* resource.
*)
IntegerResource(# resourceName::< XmNuserData #);

(* Callback definitions.  Only the helpCallback pattern is
* actually used in Gadget, the rest is used in various
* specializations.  But since many specializations define
* callbacks with identical names, for convenience, most of
* the corresponding patterns are declared here.
*)
helpCallback:< CallbackProc
(* Called when the help key is pressed.  The reason sent by
* the callback is XmCR_HELP
*);

activateCallback:< CallbackProc;
armCallback:< CallbackProc;
disarmCallback:< CallbackProc;
cascadingCallback:< CallbackProc;
exposeCallback:< CallbackProc;

```



```

resizeCallback:< CallbackProc;
valueChangedCallback:< CallbackProc;
incrementCallback:< CallbackProc;
decrementCallback:< CallbackProc;
pageincrementCallback:< CallbackProc;
pagedecrementCallback:< CallbackProc;
toTopCallback:< CallbackProc;
toBottomCallback:< CallbackProc;
dragCallback:< CallbackProc;
focusCallback:< CallbackProc;
losingFocusCallback:< CallbackProc;
modifyVerifyCallback:< CallbackProc;
motionVerifyCallback:< CallbackProc;

installCallbacks:< (* Private *)
    (# do ...; INNER #);

<<SLOT GadgetLib: attributes>>
#)

```

---

Gadget Interface

[Mjølner Informatics](#)



# Labelgadget Interface

```
ORIGIN 'gadget';

( *
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 * )

-- XtEnvLib: attributes --

LabelGadget: Gadget
( * A LabelGadget can contain either a MotifString or a pixmap.
 * When a LabelGadget is insensitive, its text is stippled, or
 * the user-supplied insensitive pixmap is displayed
 * )
( # init: <
    ( # WidgetClass: <
        ( #
            do INNER;
                ( if value=0 then xmLabelGadgetClass->value if )
            # )
        do INNER # );

    ( * Resources * )
accelerator:
    ( * Sets the accelerator on a button widget in a menu, which
    * activates a visible or invisible button from the
    * keyboard. This resource is a string that describes a set
    * of modifiers and the key that may be used to select the
    * button. The format of this string is identical to that
    * used by the translations manager, with the exception that
    * only a single event may be specified and only KeyPress
    * events are allowed. Accelerators for buttons are
    * supported only for PushButton and ToggleButton in
    * Pulldown and Popup MenuPanels.
    * )
    StringResource( # resourceName: < XmNaccelerator # );
acceleratorText:
    ( * Specifies the text displayed for the accelerator. The
    * text is displayed to the side of the label string or
    * pixmap. Accelerator text for buttons is displayed only
    * for PushButtons and ToggleButtons in Pulldown and Popup
    * Menus.
    * )
    MotifStringResource( # resourceName: < XmNacceleratorText # );
alignment:
    ( * Specifies the label alignment for text or pixmap.
    *
    * + XmALIGNMENT_BEGINNING (left alignment) - causes the
    *   left sides of the lines of text to be vertically
    *   aligned with the left edge of the widget window.
    *   For a pixmap, its left side is vertically aligned
    *   with the left edge of the widget window.
    *   XmALIGNMENT_CENTER (center alignment) - causes the
    *   centers of the lines of text to be vertically
    *   aligned in the center of the widget window. For a
    *   pixmap, its center is vertically aligned with the
    *   center of the widget window.
    *
    * + XmALIGNMENT_END (right alignment) - causes the right
    *   sides of the lines of text to be vertically aligned
    *   with the right edge of the widget window. For a
```



```

*    pixmap, its right side is vertically aligned with
*    the right edge of the widget window. The above
*    descriptions for text are correct when
*    stringDirection is XmSTRING_DIRECTION_L_TO_R. When
*    that resource is XmSTRING_DIRECTION_R_TO_L, the
*    descriptions for XmALIGNMENT_BEGINNING and
*    XmALIGNMENT_END are switched.
*)
CharResource(# resourceName::< XmNalignment #);
labelType:
(* Specifies the label type: XmSTRING - text displays
*    labelString. XmPIXMAP - icon data in pixmap displays
*    labelPixmap or labelInsensitivePixmap.
*)
CharResource(# resourceName::< XmNlabelType #);
marginWidth:
(* Specifies the amount of spacing between the left side of
*    THIS(LabelGadget) (specified by marginLeft) and the right
*    edge of the left shadow, and the amount of spacing
*    between the right side of THIS(LabelGadget) (specified by
*    marginRight) and the left edge of the right shadow.
*)
ShortResource(# resourceName::< XmNmarginWidth #);
marginHeight:
(* Specifies the amount of spacing between the top of
*    THIS(LabelGadget) (specified by marginTop) and the bottom
*    edge of the top shadow, and the amount of spacing between
*    the bottom of THIS(LabelGadget) (specified by
*    marginBottom) and the top edge of the bottom shadow.
*)
ShortResource(# resourceName::< XmNmarginHeight #);
marginLeft:
(* Specifies the amount of spacing between the left edge of
*    the label text and the right side of the left margin
*    (specified by marginWidth). This may be modified in
*    subpatterns. For example, ToggleButton may increase this
*    field to make room for the toggle indicator and for
*    spacing between the indicator and label. Whether this
*    actually applies to the left or right side of the label
*    may depend on the value of stringDirection.
*)
ShortResource(# resourceName::< XmNmarginLeft #);
marginRight:
(* Specifies the amount of spacing between the right edge
*    of the label text and the left side of the right margin
*    (specified by marginWidth). This may be modified in
*    subpatterns. For example, CascadeButton may increase
*    this field to make room for the cascade pixmap. Whether
*    this actually applies to the left or right side of the
*    label may depend on the value of stringDirection.
*)
ShortResource(# resourceName::< XmNmarginRight #);
marginBottom:
(* Specifies the amount of spacing between the bottom of
*    the label text and the top of the bottom margin
*    (specified by marginHeight). This may be modified in
*    subpatterns. For example, CascadeButton may increase
*    this field to make room for the cascade pixmap
*)
ShortResource(# resourceName::< XmNmarginBottom #);
marginTop:
(* Specifies the amount of spacing between the top of the
*    label text and the bottom of the top margin (specified by
*    marginHeight). This may be modified in subpatterns. For
*    example, CascadeButton may increase this field to make
*    room for the cascade pixmap.

```



```

*)
ShortResource(# resourceName:< XmNmarginTop #);
fontList:
(* Specifies the font of the text used in
 * THIS(LabelGadget). If this resource is unspecified at
 * initialization, it is initialized by looking up the
 * parent hierarchy of THIS(LabelGadget) for an ancestor
 * that is a specialization of BulletinBoard, VendorShell,
 * or MenuShell. If such an ancestor is found, the font
 * list is initialized to the appropriate default font list
 * of the ancestor widget (defaultFontList for VendorShell
 * and XmMenuShell, labelFontList or buttonFontList for
 * BulletinBoard). Use the MotifFontList pattern, if you
 * want to set this resource from the program.
 *)
IntegerResource(# resourceName:< XmNfontList #);
labelPixmap:
(* Specifies the pixmap when labelType is XmPIXMAP *)
IntegerResource(# resourceName:< XmNlabelPixmap #);
labelInsensitivePixmap
(* Specifies a pixmap used as the button face if labelType
 * is XmPIXMAP and the button is insensitive.
 *)
IntegerResource(# resourceName:< XmNlabelInsensitivePixmap #);
labelString:
(* Specifies the compound string when the labelType is
 * XmSTRING. If this resource is not set, it is initialized
 * by converting the name of the widget to a MotifString.
 *)
MotifStringResource(# resourceName:< XmNlabelString #);
mnemonic:
(* Provides the user with an alternate means of selecting a
 * button. A button in a MenuBar, a Popup MenuPane, or a
 * Pulldown MenuPane can have a mnemonic. This resource
 * contains a keysym as listed in the X11 keysym table. The
 * first character in the label string that exactly matches
 * the mnemonic in the character set specified in
 * mnemonicCharSet is underlined when the button is
 * displayed. When a mnemonic has been specified, the user
 * activates the button by pressing the mnemonic key while
 * the button is visible. If the button is a CascadeButton
 * in a MenuBar and the MenuBar does not have the focus, the
 * user must use the Alt modifier while pressing the
 * mnemonic. The user can activate the button by pressing
 * either the shifted or the unshifted mnemonic key.
 *)
IntegerResource(# resourceName:< XmNmnemonic #);
mnemonicCharSet:
(* Specifies the character set of the mnemonic for
 * THIS(LabelGadget). The default is determined dynamically
 * depending on the current language environment.
 *)
IntegerResource(# resourceName:< XmNmnemonicCharSet #);
recomputeSize:
(* Indicates whether THIS(LabelGadget) attempts to be big
 * enough to contain the label. If True, setting a new
 * labelString or pixmap, accelerator text, margins, font,
 * or label type causes THIS(LabelGadget) to shrink or
 * expand to exactly fit the new labelString or pixmap. If
 * False, THIS(LabelGadget) never attempts to change size on
 * its own.
 *)
BooleanResource(# resourceName:< XmNrecomputeSize #);
stringDirection:
(* Specifies the direction in which the string is to be
 * drawn. The following are the values:

```



```

* XmSTRING_DIRECTION_L_TO_R - left to right
* XmSTRING_DIRECTION_R_TO_L - right to left The default for
* this resource is determined at creation time. If no
* value is specified for this resource and
* THIS(LabelGadget)'s parent is a manager, the value is
* inherited from the parent; otherwise, it defaults to
* XmSTRING_DIRECTION_L_TO_R.
*)
IntegerResource(# resourceName:< XmNstringDirection #);

<<SLOT LabelGadgetLib: attributes>>
#) (* labelgadget *);

(* Alias used in CompositeLib *)
mLabelGadget: LabelGadget(# #);

--- CompositeLib: attributes ---

(* Redefinition of gadget within composites making the composite
* the default father of the gadget
*)
LabelGadget: mLabelGadget
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Labelgadget Interface

[Mjølner Informatics](#)



# Cascadebuttongadget Interface

```
ORIGIN 'labelgadget';
BODY 'private/cascadebutgadgetbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

CascadeButtonGadget: LabelGadget
(* A CascadeButtonGadget links two MenuPanels or a MenuBar to a
 * MenuPane. It is used in menu systems and must have a
 * RowColumn parent with its rowColumnType resource set to
 * XmMENU_BAR, XmMENU_POPUP or XmMENU_PULLDOWN. It is the only
 * widget that can have a Pulldown MenuPane attached to it as a
 * submenu. The submenu is displayed when this widget is
 * activated within a MenuBar, a PopupMenu, or a PulldownMenu.
 * Its visuals can include a label or pixmap and a cascading
 * indicator when it is in a Popup or Pulldown MenuPane; or, it
 * can include only a label or a pixmap when it is in a MenuBar.
 *)
(# init::<
  (# WidgetClass::<
    (#
      do INNER;
      (if value=0 then xmCascadeButtonGadgetClass->value if)
    #)
    do INNER
  #);

  (* Resources *)
subMenuId:
  (* Specifies the widget ID for the Pulldown MenuPane to be
   * associated with THIS(CascadeButtonGadget). The specified
   * MenuPane is displayed when THIS(CascadeButtonGadget)
   * becomes armed. The MenuPane must have been created with
   * the appropriate parentage depending on the type of menu
   * used.
   *)
  IntegerResource(# resourceName::< XmNsubMenuId #);
cascadePixmap:
  (* Specifies the cascade pixmap displayed on one end of
   * THIS(CascadeButtonGadget) when a CascadeButton is used
   * within a Popup or Pulldown MenuPane and a submenu is
   * attached. The Label resources marginBottom, marginLeft,
   * marginRight, and marginTop may be modified to ensure that
   * room is left for the cascade pixmap. The default cascade
   * pixmap is an arrow pointing to the side of the menu where
   * the submenu will appear.
   *)
  IntegerResource(# resourceName::< XmNcascadePixmap #);
mappingDelay:
  (* Specifies the amount of time, in milliseconds, between
   * when THIS(CascadeButtonGadget) becomes armed and when it
   * maps its submenu. This delay is used only when the
   * widget is within a Popup or Pulldown MenuPane.
   *)
  IntegerResource(# resourceName::< XmNmappingDelay #);

  (* Utility patterns *)
```



```

highlight:
(* Draws or erases the shadow highlight around
 * THIS(CascadeButtonGadget): If draw if true, the shadow is
 * drawn, otherwise it is erased.
 *)
(# draw: @boolean;
 enter draw do ...
 #);

(* Callbacks *)
CascadeButtonGadgetCallback
(* Prefix for CascadeButtonGadget callbacks *)
MotifCallback(# do INNER #);

activateCallback:< CascadeButtonGadgetCallback
(* Called when the user activates
 * THIS(CascadeButtonGadget), and there is no submenu
 * attached to pop up. The activation occurs by releasing a
 * mouse button or by typing the mnemonic associated with
 * the widget. The specific mouse button depends on
 * information in the RowColumn parent. The reason sent by
 * the callback is XmCR_ACTIVATE.
 *);

cascadingCallback:< CascadeButtonGadgetCallback
(* Called just prior to the mapping of the submenu
 * associated with THIS(CascadeButtonGadget). The reason
 * sent by the callback is XmCR_CASCADING.
 *);
(* Inherited callbacks *)
helpCallback:< CascadeButtonGadgetCallback;

installCallbacks:< (* Private *)
(# do ...; INNER #);

<<SLOT CascadeButtonGadgetLib: attributes>>
#) (* CascadeButtonGadget *);

(* Alias used in CompositeLib *)
mCascadeButtonGadget: CascadeButtonGadget (# #);

--- CompositeLib: attributes ---

(* Redefinition of gadget within composites making the composite
 * the default father of the gadget
 *)
CascadeButtonGadget: mCascadeButtonGadget
(# init:< (# GetFatherWidget:< (# do THIS(Composite)->value #);
 do INNER
 #)
 #)
#)

```

---



# Pushbuttongadget Interface

```
ORIGIN 'labelgadget';
BODY 'private/pushbuttongadgetbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

PushButtonGadget: LabelGadget
(* A PushButtonGadget issues commands within an application. It
 * consists of a text label or pixmap surrounded by a border
 * shadow. When a PushButtonGadget is selected, the shadow
 * changes to give the appearance that it has been pressed in.
 * When a PushButtonGadget is unselected, the shadow changes to
 * give the appearance that it is out.
 *)
(# init::<
  (# WidgetClass::<
    (#
      do INNER;
      (if value=0 then xmPushButtonGadgetClass->value if)
    #)
  do INNER #);

(* Resources *)
fillOnArm:
(* Forces THIS(PushButtonGadget) to fill the background
 * with the color specified by armColor when
 * THIS(PushButtonGadget) is armed and when this resource is
 * set to True. If False, only the top and bottom shadow
 * colors are switched. When THIS(PushButtonGadget) is in a
 * menu, this resource is ignored and assumed to be False.
 *)
BooleanResource(# resourceName::< XmNfillOnArm #);

ArmColor:
(* Specifies the color with which to fill the armed button.
 * fillOnArm must be set to True for this resource to have
 * an effect. The default for a color display is a color
 * between the background and the bottom shadow color. For
 * a monochrome display, the default is set to the
 * foreground color, and any text in the label appears in
 * the background color when the button is armed.
 *)
IntegerResource(# resourceName::< XmNarmColor #);

armPixmap:
(* Specifies the pixmap to be used as the button face if
 * labelType is XmPIXMAP and THIS(PushButtonGadget) is
 * armed. This resource is disabled when
 * THIS(PushButtonGadget) is in a menu.
 *)
IntegerResource(# resourceName::< XmNarmPixmap #);

defaultButtonShadowThickness
(* This resource specifies the width of the default button
 * indicator shadow. If this resource is zero, the width of
 * the shadow comes from the value of the showAsDefault
 * resource. If this resource is greater than zero, the
 * showAsDefault resource is only used to specify whether
 * THIS(PushButton) is the default.
 *)
```



```

ShortResource(# resourceName:< XmNarmPixmap #);
multiClick:
(* If a button click is followed by another button click
 * within the time span specified by the display's multi-click
 * time, and this resource is set to XmMULTICLICK_DISCARD, do * not process the second click
 * XmMULTICLICK_KEEP, process the event and increment
 * click_count in the callback structure. When
 * THIS(PushButton) is not in a menu, the default value is
 * XmMULTICLICK_KEEP.
 *)
CharResource(# resourceName:< XmNmultiClick #);
ShowAsDefault:
(* If defaultButtonShadowThickness is greater than zero, a
 * value greater than zero in this resource specifies to
 * mark THIS(PushButtonGadget) as the default button. If
 * defaultButtonShadowThickness is zero, a value greater
 * than zero in this resource specifies to mark
 * THIS(PushButtonGadget) as the default button with the
 * shadow thickness specified by this resource. The space
 * between the shadow and the default shadow is equal to the
 * sum of both shadows. The default value is zero. When
 * this value is not zero, the Label resources marginLeft,
 * marginRight, marginTop, and marginBottom may be modified
 * to accommodate the second shadow. This resource is
 * disabled when THIS(PushButtonGadget) is in a menu.
 *)
ShortResource(# resourceName:< XmNshowAsDefault #);

(* Callbacks *)
PushButtonGadgetCallback MotifCallback
(* Prefix for callbacks to THIS(PushButtonGadget) *)
(# callData:< XmPushButtonCallbackStruct do INNER #);

activateCallback:< PushButtonGadgetCallback
(* Called when THIS(PushButtonGadget) is activated.
 * THIS(PushButtonGadget) is activated when the user presses
 * and releases the active mouse button while the pointer is
 * inside THIS(PushButtonGadget). Activating
 * THIS(PushButtonGadget) also disarms it. For this
 * callback the reason is XmCR_ACTIVATE.
 *);
armCallback:< PushButtonGadgetCallback
(* Called when THIS(PushButtonGadget) is armed.
 * THIS(PushButtonGadget) is armed when the user presses the
 * active mouse button while the pointer is inside
 * THIS(PushButtonGadget). For this callback the reason is
 * XmCR_ARM.
 *);
disarmCallback:< PushButtonGadgetCallback
(* Called when THIS(PushButtonGadget) is disarmed.
 * THIS(PushButton) is disarmed when the user presses and
 * releases the active mouse button while the pointer is
 * inside THIS(PushButtonGadget). For this callback, the
 * reason is XmCR_DISARM.
 *);
(* Inherits callbacks *)
helpCallback:< PushButtonGadgetCallback;

installCallbacks:< (* Private *)
(# do ...; INNER #);

<<SLOT PushButtonGadgetLib: attributes>>
#) (* PushButtonGadget *);

(* Alias used in CompositeLib *)
mPushButtonGadget: PushButtonGadget(# #);

```



```

--- CompositeLib: attributes ---

(* Redefinition of gadget within composites making the composite
 * the default father of the gadget
 *)
PushButtonGadget: mPushButtonGadget
  (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Pushbutton gadget Interface

[Mjølner Informatics](#)



# Togglebuttongadget Interface

```
ORIGIN 'labelgadget';
BODY 'private/togglebuttongadgetbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

ToggleButtonGadget: LabelGadget
(* A ToggleButtonGadget sets nontransitory state data within an
 * application.  Usually this widget consists of an indicator
 * (square or diamond) with either text or a pixmap on one side
 * of it.  However, it can also consist of just text or a pixmap
 * without the indicator.  The toggle graphics display a
 * 1-of-many or N-of-many selection state.  When a toggle
 * indicator is displayed, a square indicator shows an N-of-many
 * selection state and a diamond indicator shows a 1-of-many
 * selection state.  A ToggleButtonGadget implies a selected or
 * unselected state.  In the case of a label and an indicator, an
 * empty indicator (square or diamond shaped) indicates that
 * ToggleButtonGadget is unselected, and a filled indicator shows
 * that it is selected.  In the case of a pixmap toggle,
 * different pixmaps are used to display the selected/unselected
 * states.
 *)
(# init::<
    (# WidgetClass::<
        (#
            do INNER;
                (if value=0 then xmToggleButtonGadgetClass->value if)
            #)
        do INNER
        #);

    (* Resources *)
indicatorSize:
    (* Sets the size of the indicator.  A value of
     * XmINVALID_DIMENSION causes the indicator to be set to the
     * size of the font of the label string.
     *)
    ShortResource(# resourceName::< XmNindicatorSize #);
indicatorType:
    (* Specifies if the indicator is a 1-of or N-of indicator.
     * For the 1-of indicator, the value is XmONE_OF_MANY.  For
     * the N-of indicator, the value is XmN_OF_MANY.  The
     * N-of-many indicator is square. The 1-of-many indicator is
     * diamond shaped.  This resource specifies only the visuals
     * and does not enforce the behavior.  When
     * THIS(ToggleButtonGadget) is in a RadioBox, the default is
     * XmONE_OF_MANY; otherwise, the default is XmN_OF_MANY.
     *)
    CharResource(# resourceName::< XmNindicatorType #);
visibleWhenOff:
    (* Indicates that the toggle indicator is visible in the
     * unselected state when the Boolean value is True.  When
     * THIS(ToggleButtonGadget) is in a menu, the default value
     * is False.  When THIS(ToggleButtonGadget) is in a
     * RadioBox, the default value is True.
     *)
    BoolResource(# resourceName::< XmNvisibleWhenOff #);
```



```

    BooleanResource(# resourceName::< XmNvisibleWhenOff #);
spacing:
    (* Specifies the amount of spacing between the toggle
       * indicator and the toggle label (text or pixmap).
       *)
    ShortResource(# resourceName::< XmNspacing #);
selectPixmap:
    (* Specifies the pixmap to be used as the button face if
       * labelType is XmPIXMAP and THIS(ToggleButtonGadget) is
       * selected.  When THIS(ToggleButtonGadget) is unselected,
       * the pixmap specified in Label's labelPixmap is used.
       *)
    IntegerResource(# resourceName::< XmNselectPixmap #);
selectInsensitivePixmap:
    (* Specifies a pixmap used as the button face when
       * THIS(ToggleButtonGadget) is selected and the button is
       * insensitive if the Label resource labelType is set to
       * XmPIXMAP.  If the ToggleButton is unselected and the
       * button is insensitive, the pixmap in
       * labelInsensitivePixmap is used as the button face.
       *)
    IntegerResource
        (# resourceName::< XmNselectInsensitivePixmap #);
set:
    (* Displays the button in its selected state if set to
       * True.  This shows some conditions as active when a set of
       * buttons first appears.
       *)
    BooleanResource(# resourceName::< XmNset #);
indicatorOn:
    (* Specifies that a toggle indicator is drawn to one side
       * of the toggle text or pixmap when set to True.  When set
       * to False, no space is allocated for the indicator, and it
       * is not displayed.  If indicatorOn is True, the indicator
       * shadows are switched when the button is selected or
       * unselected, but, any shadows around the entire widget are
       * not switched.  However, if indicatorOn is False, any
       * shadows around the entire widget are switched when the
       * toggle is selected or unselected.
       *)
    BooleanResource(# resourceName::< XmNindicatorOn #);
fillOnSelect:
    (* Fills the indicator with the color specified in
       * selectColor and switches the top and bottom shadow colors
       * when set to True.  Otherwise, it switches only the top
       * and bottom shadow colors.
       *)
    BooleanResource(# resourceName::< XmNfillOnSelect #);
selectColor:
    (* Allows the application to specify what color fills the
       * center of the square or diamond-shaped indicator when it
       * is set.  If this color is the same as either the top or
       * the bottom shadow color of the indicator, a
       * one-pixel-wide margin is left between the shadows and the
       * fill; otherwise, it is filled completely.  This
       * resource's default for a color display is a color between
       * the background and the bottom shadow color.  For a
       * monochrome display, the default is set to the foreground
       * color.  The meaning of this resource is undefined when
       * indicatorOn is False.
       *)
    IntegerResource(# resourceName::< XmNselectColor #);

    (* Utility patterns *)
state:
    (* Used to manipulate the state of THIS(ToggleButtonGadget)

```



```

*          directly
*)
(# value: @boolean;
get:
  (* Returns True if THIS(ToggleButtonGadget) is
   * selected and False if THIS(ToggleButtonGadget) is
   * unselected.
   *)
  (#
  do ...;
  exit value
  #);
set:
  (* Enters a Boolean value that indicates whether
   * THIS(ToggleButtonGadget) state should be selected or
   * unselected. If True, the button state is selected;
   * if False, the button state is unselected. The
   * notify parameter indicates whether the
   * valueChangedCallback is called; it can be either
   * True or False. When this argument is True and
   * THIS(ToggleButtonGadget) is a child of a RowColumn
   * widget whose radioBehavior is True, selecting
   * THIS(ToggleButtonGadget) causes other
   * ToggleButtonGadget and ToggleButtonGadgetGadget
   * children of the RowColumn to be unselected.
   *)
  (# notify: @boolean
  enter (value, notify)
  do ...;
  #)
enter set
exit get
#);

(* Callbacks *)
ToggleButtonGadgetCallback MotifCallback
  (* Prefix for ToggleButton callbacks *)
  (# callData:< XmToggleButtonCallbackStruct do INNER #);

armCallback:< ToggleButtonGadgetCallback
  (* Called when THIS(ToggleButtonGadget) is armed. To arm
   * THIS(ToggleButtonGadget), press the active mouse button
   * while the pointer is inside THIS(ToggleButtonGadget). The
   * reason sent by this callback is XmCR_ARM.
   *);
disarmCallback:< ToggleButtonGadgetCallback
  (* Called when THIS(ToggleButtonGadget) is disarmed. To
   * disarm THIS(ToggleButtonGadget), press and release the
   * active mouse button while the pointer is inside the
   * THIS(ToggleButtonGadget). THIS(ToggleButtonGadget) is
   * also disarmed when the user moves out of
   * THIS(ToggleButtonGadget) and releases the mouse button
   * when the pointer is outside THIS(ToggleButtonGadget).
   * For this callback, the reason is XmCR_DISARM.
   *);
valueChangedCallback:< ToggleButtonGadgetCallback
  (* Called when THIS(ToggleButtonGadget) value is changed.
   * To change the value, press and release the active mouse
   * button while the pointer is inside
   * THIS(ToggleButtonGadget). This action also causes
   * THIS(ToggleButtonGadget) to be disarmed. For this
   * callback, the reason is XmCR_VALUE_CHANGED.
   *);
(* Inherited callbacks *)
helpCallback:< ToggleButtonGadgetCallback;

```



```

installCallbacks: < (* Private *)
  (# do ...; INNER #);

  <<SLOT ToggleButtonGadgetLib: attributes>>
  #) (* ToggleButtonGadget *);

(* Alias used in CompositeLib *)
mToggleButtonGadget: ToggleButtonGadget(# #);

--- CompositeLib: attributes ---

(* Redefinition of gadget within composites making the composite
 * the default father of the gadget
 *)
ToggleButtonGadget: mToggleButtonGadget
  (# init: < (# GetFatherWidget: < (# do THIS(Composite)->value #);
    do INNER
    #)
  #)

```

---

Togglebuttongadget Interface

[Mjølner Informatics](#)



# Arrowbuttongadget Interface

```
ORIGIN 'gadget';
BODY 'private/arrowbuttongadgetbody';

(*
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

-- XtEnvLib: attributes --

ArrowButtonGadget: Gadget
(* An ArrowButtonGadget consists of a directional arrow
 * surrounded by a border shadow. When it is selected, the
 * shadow changes to give the appearance that the ArrowButton has
 * been pressed in. When the ArrowButtonGadget is unselected,
 * the shadow reverts to give the appearance that the
 * ArrowButtonGadget is released, or out.
 *)
(# <<SLOT arrowButtonGadgetLib: attributes>>;

init::<
    (# WidgetClass::<
        (#
            do INNER;
            (if value=0 then xmArrowButtonGadgetClass->value if)
        #)
        do INNER
    #);

    (* Resources *)
arrowDirection:
    (* Sets the arrow direction. Can be one of XmARROW_UP,
     * XmARROW_DOWN, XmARROW_LEFT, and XmARROW_RIGHT.
     *)
    CharResource(# resourceName::< XmNarrowDirection #);
multiClick:
    (* If a button click is followed by another button click
     * within the time span specified by the display's multi-click
     * time, and this resource is set to XmMULTICLICK_DISCARD, do * not process the second c
     * XmMULTICLICK_KEEP, process the event and increment
     * click_count in the callback structure. When the button is
     * not in a menu, the default value is XmMULTICLICK_KEEP.
     *)
    CharResource(# resourceName::< XmNmultiClick #);

    (* Callbacks *)
ArrowButtonGadgetCallback MotifCallback
    (* Prefix for ArrowButton callbacks *)
    (# callData::< XmArrowButtonCallbackStruct do INNER #);

activateCallback::< ArrowButtonGadgetCallback
    (* Called when THIS(ArrowButtonGadget) is activated.
     * Activating THIS(ArrowButtonGadget) also disarms it. The
     * reason sent by this callback is XmCR_ACTIVATE.
     *);
armCallback::< ArrowButtonGadgetCallback
    (* Called when THIS(ArrowButtonGadget) is armed. The reason
     * sent by this callback is XmCR_ARM.
     *);
disarmCallback::< ArrowButtonGadgetCallback
    (* Called when THIS(ArrowButtonGadget) is disarmed. The
```



```

        * reason for this callback is XmCR_DISARM.
        *);
(* Inherited callbacks *)
helpCallback::< ArrowButtonGadgetCallback;

installCallbacks::< (* Private *)
    (# do ...; INNER #);

<<SLOT ArrowButtonGadgetLib: attributes>>;
#) (* ArrowButtonGadget *);

(* Alias used in CompositeLib *)
mArrowButtonGadget: ArrowButtonGadget(# #);

--- CompositeLib: attributes ---

(* Redefinition of gadget within composites making the composite
 * the default father of the gadget
 *)
ArrowButtonGadget: mArrowButtonGadget
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #)

```

---

Arrowbuttongadget Interface

[Mjølner Informatics](#)



# SeparatorGadget Interface

```
ORIGIN 'gadget';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- XtEnvLib: attributes --

SeparatorGadget: Gadget
(* A SeparatorGadget is a gadget that separates items in a
 * display. Several different line drawing styles are provided,
 * as well as horizontal or vertical orientation.
 *)
(# init: <
    (# WidgetClass: <
        (#
            do INNER;
            (if value=0 then xmSeparatorGadgetClass->value if)
        #)
    do INNER
    #);

    (* Resources *)
    separatorType:
    (* Specifies the type of line drawing to be done in
     * THIS(SeparatorGadget):
     *
     * + XmSINGLE_LINE - single line.
     *
     * + XmDOUBLE_LINE - double line.
     *
     * + XmSINGLE_DASHED_LINE - single-dashed line.
     *
     * + XmDOUBLE_DASHED_LINE - double-dashed line.
     *
     * + XmNO_LINE - no line.
     *
     * + XmSHADOW_ETCHED_IN - double line giving the effect of
     * a line etched into the window. The thickness of the
     * double line is equal to the value of shadowThickness.
     * For horizontal orientation, the top line is drawn in
     * topShadowColor and the bottom line is drawn in
     * bottomShadowColor. For vertical orientation, the left
     * line is drawn in topShadowColor and the right line is
     * drawn in bottomShadowColor.
     *
     * + XmSHADOW_ETCHED_OUT - double line giving the effect of
     * an etched line coming out from the window. The
     * thickness of the double line is equal to the value of
     * shadowThickness. For horizontal orientation, the top
     * line is drawn in bottomShadowColor and the bottom
     * line is drawn in topShadowColor. For vertical
     * orientation, the left line is drawn in
     * bottomShadowColor and the right line is drawn in
     * topShadowColor.
     *)
    CharResource(# resourceName: < XmNseparatorType #);
    margin:
    (* For horizontal orientation, specifies the space on the
     * left and right sides between the border of
```



```

    * THIS(SeparatorGadget) and the line drawn. For vertical
    * orientation, specifies the space on the top and bottom
    * between the border of the THIS(SeparatorGadget) and the
    * line drawn.
    *)
    ShortResource(# resourceName::< XmNmargin #);
orientation:
    (* Displays THIS(SeparatorGadget) vertically or
    * horizontally. This resource can have values of
    * XmVERTICAL and XmHORIZONTAL.
    *)
    CharResource(# resourceName::< XmNOrientation #);

    <<SLOT SeparatorGadgetLib: attributes>>
    #) (* SeparatorGadget *);

    (* Alias used in CompositeLib *)
mSeparatorGadget: SeparatorGadget(# #);

    --- CompositeLib: attributes ---

    (* Redefinition of gadget within composites making the composite
    * the default father of the gadget
    *)
SeparatorGadget: mSeparatorGadget
    (# init::< (# GetFatherWidget::< (# do THIS(Composite)->value #);
        do INNER
        #)
    #)

```

---

SeparatorGadget Interface

[Mjølner Informatics](#)



# Motifenv Interface

```
ORIGIN 'xtenv';
INCLUDE 'motif/basics';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

(* This fragment group simply defines the MotifEnv prefix for
 * applications using the BETA interface to Motif. For each
 * widget/gadget wanted in the application, the fragment group in
 * the sub-directory 'motif', defining the BETA interface to it,
 * must be explicitly included. For ease of use, the fragment group
 * 'allmotif' includes the interface to all the widgets/gadgets,
 * and may be used instead of this fragment group at the price of a
 * slightly bigger executable.
 *)

-- LIB: attributes --

MotifEnv: XtEnv
  (# <<SLOT MotifEnvLib: attributes>>
  do INNER
  #)
```

---

Motifenv Interface

[Mjolner Informatics](#)



# Allmotif Interface

```
ORIGIN 'motifenv';

( *
  * COPYRIGHT
  *   Copyright Mjolner Informatics, 1992-97
  *   All rights reserved.
  *)

( * This fragment group includes all of the BETA interface to
  * Motif.  If separate inclusion - which may give smaller
  * executables - is wanted, use the fragment group 'motifenv'
  * instead.
  *)

INCLUDE 'motif/primitives';
INCLUDE 'motif/shells';
INCLUDE 'motif/labels';
INCLUDE 'motif/texts';
INCLUDE 'motif/managers';
INCLUDE 'motif/lists';
INCLUDE 'motif/dialogs';
INCLUDE 'motif/gadgets';
```

---

Allmotif Interface

[Mjolner Informatics](#)



# Awenv Interface

```
ORIGIN 'xtenv';
BODY 'private/awenvbody';
INCLUDE 'athena/awlib';
(
 * The BETA interface to the Athena widget set. The Athena widget set
 * is build on top of Xt and it contains user interface elements like
 * scrollbars, commands buttons, menus, etc.
 *
 * COPYRIGHT
 *     Copyright Mjolner Informatics, 1992-97
 *     All rights reserved.
 *)

--- xtenvlib: attributes ---

Sme: RectObj
(
 * The Sme (simple menu entry) gadget is the base class for all
 * menu entries. While this pattern is mainly intended to be a
 * superpattern, it may be used in a menu to add blank space between
 * menu entries.
 *)
(# <<SLOT smeLIB: attributes>>;

  init::<
    (# widgetClass::<
      (#
        do INNER;
        (if value=0 then SmeObjectClass -> value if)
      #)
    do INNER
    #);
  #) (* Sme *);

SmeBSB: sme
(
 * The SmeBSB (simple menu entry composed of a bitmap, a string and
 * a bitmap) gadget is used to create a menu entry that contains a
 * string, and optional bitmaps in its left and right margins. Since
 * each menu entry is an independent object, the application is able
 * to change the font, colour, height, and other attributes of the
 * menu entries, on an entry by entry basis.
 *)
(# <<SLOT smeBSBLIB: attributes>>;

  init::<
    (# widgetClass::<
      (#
        do INNER;
        (if value=0 then SmeBSBObjectClass -> value if)
      #)
    do INNER;
    #);

    (* SmeBSB resources *)
    label:
      (* This is a the string that will be displayed in the menu
       * entry. The exact location of this string within the bounds
       * of the menu entry is controlled by the leftMargin,
       * right-Margin, vertSpace, and justify attributes.
       *)
      StringResource(# resourceName::< XtNlabel #);
    font: (* The font used to display the text label *)
      IntegerResource(# resourceName::< XtNFont #);
```



**foreground:**

```
( * A pixel value which indexes the SimpleMenu's colormap to
 * derive the foreground colour of the menu entry's window.
 * This colour is also used to render all 1's in the left and
 * right bitmaps.
 *)
```

```
IntegerResource(# resourceName::< XtNForeground #);
```

**justify:**

```
( * How the label is to be rendered between the left and right
 * margins when the space is wider than the actual text. This
 * attribute may be specified with the values XtJustifyLeft,
 * XtJustifyCenter, or XtJustifyRight.
 *)
```

```
IntegerResource(# resourceName::< XtNJustify #);
```

**leftBitmap:**

```
( * The bitmap to display in the left margin of the menu entry.
 * All 1's in the bitmap will be rendered in the foreground
 * colour, and all 0's will be drawn in the background colour
 * of the SimpleMenu widget. It is the programmers
 * responsibility to make sure that the menu entry is tall
 * enough, and the appropriate margin wide enough to accept the
 * bitmap. If care is not taken the bitmap may extend into
 * another menu entry, or into this entry's label. See also
 * fileToBitmap.
 *)
```

```
IntegerResource(# resourceName::< XtNleftBitmap #);
```

**rightBitmap:**

```
( * The bitmap to display in the right margin of the menu
 * entry. See also leftBitmap
 *)
```

```
IntegerResource(# resourceName::< XtNrightBitmap #);
```

**leftMargin:**

```
( * The amount of space (in pixels) that will be left between
 * the left edge of the menu entry and the label string.
 *)
```

```
ShortResource(# resourceName::< XtNleftMargin #);
```

**rightMargin:**

```
( * The amount of space (in pixels) that will be left between
 * the right edge of the menu entry and the label string.
 *)
```

```
ShortResource(# resourceName::< XtNrightMargin #);
```

**vertSpace:**

```
( * The amount of vertical padding, expressed as a percentage
 * of the height of the font, that is to be placed around the
 * label of a menu entry. The label and bitmaps are always
 * centered vertically within the menu. The default value for
 * this attribute (25) causes the default height to be 125% of
 * the height of the font.
 *)
```

```
IntegerResource(# resourceName::< XtNVertSpace #);
```

**installcallbacks: <**

```
( * Install the "callback" virtual, so that it is invoked when
 * the user has selected THIS(SmeBSB).
 *)
```

```
(# do ...; INNER #);
```

```
#) (* SmeBSB *);
```

**SmeLine: sme**

```
( * The SmeLine gadget is used to add a horizontal line or menu
 * separator to a menu. Since each menu entry is an independent
 * object, the application is able to change the colour, height, and
 * other attributes of the menu entries, on an entry by entry
 * basis. THIS(SmeLine) is not selectable, and will not highlight
 * when the pointer cursor is over it.
 *)
```



```

(# <<SLOT smeLineLIB: attributes>>;

init::<
  (# widgetClass::<
    (# do INNER;
      (if value=0 then SmeLineObjectClass -> value if)
    #)
  do INNER
  #);

(* SmeLine resources *)
foreground:
  (* A pixel value which indexes the colormap of the SimpleMenu
   * of which THIS(SmeLine) is a child to derive the foreground
   * colour used to draw the separator line.
   *)
  IntegerResource(# resourceName::< XtNForeGround #);
linewidth:
  (* The width of the horizontal line that is to be
   * displayed. I.e., it is actually the height.
   *)
  IntegerResource(# resourceName::< XtNLineWidth #);
stipple:
  (* If a bitmap is specified for this attribute, the line will
   * be stippled through it. This allows the menu separator to be
   * rendered as something more exciting than just a line. For
   * instance, if you define a stipple that is a chain link, then
   * your menu separators will look like chains.
   *)
  IntegerResource(# resourceName::< XtNStipple #);
#) (* SmeLine *);

SmeCascade: smeBSB
(* The SmeCascade object is used to add a cascading submenu to
 * another menu. When the user points inside THIS(SmeCascade), the
 * submenu is popped up to the right of the item. The user can then
 * select among the items of the cascaded menu.
 *
 * Since each menu entry is an independent object, the application
 * is able to change the colour, height, and other attributes of the
 * menu entries, on an entry by entry basis.
 *)
(# <<SLOT smeCascadeLIB: attributes>>;

init::<
  (# widgetClass::<
    (#
      do INNER;
        (if value=0 then SmeCascadeObjectClass -> value if)
    #)
  do INNER
  #);

subMenu:
  (* The submenu that is popped up when THIS(SmeCascade) is
   * selected.
   *)
  IntegerResource(# resourceName::< XtNSubMenu #);
#) (* SmeCascade *);

SimpleMenu: OverrideShell
(* The SimpleMenu widget is a container for the menu entries. This
 * is the only part of the menu that actually contains a window. The
 * SimpleMenu serves as the glue to bind the individual menu entries
 * together into a menu. The SimpleMenu widget is itself a direct
 * subclass of OverrideShell, therefore no other shell is necessary

```



```

* when creating a menu. The children of a SimpleMenu must be
* subclasses of Sme (Simple Menu Entry). See also MenuButton.
*)
(# <<SLOT simpleMenuLIB: attributes>>;

(* Convenient aliases for menu entries *)
item: smeBSB(# #);
blank: sme(# #);
line: smeLine(# #);
cascade: smeCascade(# #);

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then simpleMenuWidgetClass->value if)
    #);
  do INNER;
  #);

(* SimpleMenu resources *)
backingStore:
  IntegerResource(# resourceName::< XtNBackingStore #);
bottomMargin:
  (* The amount of space between the bottom of the menu and the
  * menu entry closest to that edge.
  *)
  ShortResource(# resourceName::< XtNBottomMargin #);
topMargin:
  (* The amount of space between the top of the menu and the
  * menu entry closest to that edge.
  *)
  ShortResource(# resourceName::< XtNTopMargin #);
cursor:
  (* The shape of the mouse pointer whenever it is in this
  * widget.
  *)
  IntegerResource(# resourceName::< XtNCursor #);
label:
  (* This label will be placed at the top of THIS(SimpleMenu),
  * and will not be highlighted. menuLabel will return the
  * corresponding widget object.
  *)
  IntegerResource(# resourceName::< XtNLabel #);
menuLabel: IntegerValue
  (* The widget showing the label *)
  (# do (thewidget, 'menuLabel') -> XtNameToWidget -> value #);
labelClass:
  (* Specifies the type of Sme object created as the menu
  * label. Can be smeObjectClass, smeBSBObjectClass or
  * SmeLineObjectClass
  *)
  IntegerResource(# resourceName::< XtNlabelClass #);
menuOnScreen:
  (* If this attribute is True then when the menu is on the
  * screen it will always be fully visible.
  *)
  BooleanResource(# resourceName::< XtNmenuOnScreen #);
popupOnEntry:
  (* When the menu is popped up, it will by default pop it up
  * with the first item selected. To popup the menu under
  * another entry, set this attribute to the menu entry that
  * should be under the pointer, when it is popped up. This
  * allows the application to offer the user a default menu
  * entry, that can be selected without moving the pointer.
  *)

```



```

IntegerResource(# resourceName::< XtNpopupOnEntry #);
rowHeight:
(* If this attribute is 0 (default) then each menu entry will
 * be given its desired height. Otherwise all menu entries will
 * be forced to be rowheight pixels high.
 *)
ShortResource(# resourceName::< XtNrowHeight #);

(* Redclarations of Sme and subpatterns to use THIS(SimpleMenu)
 * as father, if no father is specified in init
 *)
SmeBSB: bsmeBSB
  (# init::<
    (# getFatherWidget::< (# do THIS(SimpleMenu).theWidget->value #);
    do INNER
    #)
    #);
Sme: bsme
  (# init::<
    (# getFatherWidget::< (# do THIS(SimpleMenu).theWidget->value #);
    do INNER
    #)
    #);
SmeLine: bsmeLine
  (# init::<
    (# getFatherWidget::< (# do THIS(SimpleMenu).theWidget->value #);
    do INNER
    #)
    #);
SmeCascade: bsmeCascade
  (# init::<
    (# getFatherWidget::< (# do THIS(SimpleMenu).theWidget->value #);
    do ...; INNER
    #)
    #);
#) (* SimpleMenu *);

MenuButton: Command
  (# <<SLot menuButtonLIB: attributes>>;

  init::<
    (# widgetClass::<
      (#
        do INNER;
        (if value=0 then menuButtonWidgetClass -> value if)
      #)
    do ...;
    INNER
    #);

  menuName:
    StringResource(# resourceName::< XtNMenuName #);

  setMenu:
    (# m: ^simpleMenu
    enter m[]
    do m.name -> menuName
    #);
#) (* MenuButton *);

Simple: Core
(* The simple widget is not very useful by itself, as it has no
 * semantics of its own. Its main purpose is to be used as a common
 * superclass for the other simple Athena widgets. Provides a
 * rectangular area with a settable mouse cursor and special border.
 *)

```



```

(# <<SLOT simpleLIB: attributes>>;

init:<
  (# widgetClass:<
    (#
      do INNER;
      (if value=0 then simpleWidgetClass -> value if)
    #)
  do INNER
  #);
cursor:
  (* The image that will be displayed as the pointer cursor
   * whenever it is in THIS(Simple).
   *)
  IntegerResource(# resourceName:< XtNCursor #);
insensitiveBorder:
  (* This may contain a pixmap, that will be tiled into the
   * widget's border, if the widget becomes insensitive.
   *)
  IntegerResource(# resourceName:< XtNInsensitiveBorder #);
#) (* Simple *);

StripChart: Simple
(* The StripChart widget is used to provide a real time graphical
 * chart of a single value. This widget is used by xload to provide
 * the load graph. It will read data from an application, and
 * update the chart at the update interval specified.
 *)
(# <<SLOT stripChartLIB: attributes>>;

init:<
  (# widgetClass:<
    (#
      do INNER;
      (if value=0 then stripChartWidgetClass->value if)
    #)
  do INNER;
  #);
foreground:
  (* A pixel value which indexes the widget's colormap to derive
   * the color that will be used to draw the graph.
   *)
  IntegerResource(# resourceName:< XtNForeground #);
highlight:
  (* A pixel value which indexes the widget's colormap to derive
   * the color that will be used to draw the scale lines on the
   * graph.
   *)
  IntegerResource(# resourceName:< XtNHighLight #);
jumpScroll:
  (* When the graph reaches the right edge of the window it must
   * be scrolled to the left. This attribute specifies the
   * number of pixels it will jump. Smooth scrolling can be
   * achieved by setting this attribute to 1.
   *)
  IntegerResource(# resourceName:< XtNJumpScroll #);
minScale:
  (* The minimum scale for the graph. The number of divisions
   * on the graph will always be greater than or equal to this
   * value.
   *)
  IntegerResource(# resourceName:< XtNMinScale #);
update:
  (* The number of seconds between graph updates. Each update
   * is represented on the graph as a 1 pixel wide line. Every
   * update seconds the getValue virtual pattern will be called

```



```

    * to get a new graph point, and this point will be added to
    * the right end of THIS(StripChart).
    *)
IntegerResource(# resourceName:< XtNUpdate #);

getValue:<
(* This virtual method will be called every "update" seconds
 * to get a new graph point. This pattern must be further bound
 * by the application to compute the new value. The value is an
 * integer-quotient that should be between 0 and 1. The value
 * is set by a call to the local pattern setQuotient. E.g. the
 * new value is set to 1/2 by the call (1,2) -> setQuotient
 *)
(# valueAddr: @integer;
  setQuotient:
    (# q,r: @integer
      enter (q,r)
      do (valueAddr,q,r) -> setDoubleFromQuot
      #)
  enter valueAddr
  do INNER
  #);
installCallbacks:< (* Install the getValue virtual *)
  (# do ...; INNER #);
#) (* StripChart *);

```

**Toggle:** Command

```

(* The Toggle widget is an area, often rectangular, containing a
 * text label or bitmap image. This widget maintains a Boolean
 * state (e.g. True/False or On/Off) and changes state whenever it
 * is selected. When the pointer is on the button, the button may
 * become highlighted by drawing a rectangle around its perimeter.
 * This highlighting indicates that the button is ready for
 * selection. When pointer button 1 is pressed and released, the
 * Toggle widget indicates that it has changed state by reversing
 * its foreground and background colors, and its virtual method
 * callback is invoked. If the pointer is moved out of the widget
 * before the button is released, the widget reverts to its normal
 * foreground and background colors, and releasing the button has no
 * effect. This behavior allows the user to cancel an action.
 *)
* Toggle buttons may also be part of a radio group. A radio group
* is a list of at least two Toggle buttons in which no more than
* one Toggle may be set at any time. A radio group is identified
* by any one of its members.
*)
(# <<SLOT toggleLIB: attributes>>;

```

```

init:<
  (# widgetClass:<
    (#
      do INNER;
      (if value=0 then toggleWidgetClass -> value if)
    #)
  do INNER
  #);

```

**radiodata:**

```

(* Specifies the data that will be returned by GetCurrent when
 * this is the currently set widget in the radio group. This
 * value is also used to identify the Toggle that will be set
 * by a call to SetCurrent. The value 0 will be returned by
 * GetCurrent if no widget in a radio group is currently set.
 * Programmers must not specify 0 as radioData.
 *)
IntegerResource(# resourceName:< XtNRadioData #);

```



**radiogroup:**

```
( * Specifies another Toggle widget that is in the radio group
 * to which this Toggle widget should be added. A radio group
 * is a group of at least two Toggle widgets, only one of which
 * may be set at a time. If this value is NULL (the default)
 * then the Toggle will not be part of any radio group and can
 * change state without affecting any other Toggle widgets. If
 * the widget specified in this attribute is not already in a
 * radio group then a new radio group will be created
 * containing these two Toggle widgets. No Toggle widget can
 * be in multiple radio groups. The behavior of a radio group
 * of one toggle is undefined.
 *)
```

```
IntegerResource(# resourceName:< XtNRadioGroup #);
```

**state:**

```
( * Specifies whether the Toggle widget is set (True) or unset
 * (False).
 *)
```

```
BooleanResource(# resourceName:< XtNState #);
```

**getCurrent:** IntegerValue

```
( * Use this pattern to get the radiodata of the current set
 * Toggle in the group. If no toggle in the radio group is
 * selected then 0 is returned.
 *)
```

```
(# do theWidget -> XawToggleGetCurrent -> value #);
```

**setCurrent:**

```
( * To change the Toggle that is currently set in a radiogroup
 * use this pattern. It takes as enter-parameter the radiodata
 * of the Toggle to set.
 *)
```

```
(# data: @integer
```

```
enter data
```

```
do (theWidget,data) -> xawToggleSetCurrent
```

```
#);
```

**unsetCurrent:**

```
( * Unsets all Toggle widgets in a radiogroup *)
```

```
(# do theWidget -> XawToggleUnsetCurrent #);
```

```
#) (* Toggle *);
```

**Label:** Simple

```
( * A Label widget is a text string or bitmap displayed within a
 * rectangular region of the screen. The label may contain multiple
 * lines of characters. The Label widget will allow its string to
 * be left, right, or center justified. Normally, this widget can be
 * neither selected nor directly edited by the user. It is intended
 * for use as an output device only.
 *)
```

```
(# <<SLOT labelLIB: attributes>>;
```

**init:**<

```
(# widgetClass:<
```

```
(#
```

```
do INNER;
```

```
(if value=0 then labelWidgetClass -> value if)
```

```
#)
```

```
do INNER
```

```
#);
```

```
( * Label resources *)
```

**label:**

```
( * A text that specifies the text to be displayed in the
 * widget's window if no bitmap is specified. The default is
 * the name of the widget. Newline characters will become
 * newlines.
 *)
```



```

StringResource(# resourceName::< XtNlabel #);
bitmap:
(* A bitmap to display instead of the label. The default size
 * of the widget will be just large enough to contain the
 * bitmap and the widget's internal width and height.
 *)
IntegerResource(# resourceName::< XtNBitmap #);
font: (* The text font to use when displaying the label. *)
IntegerResource(# resourceName::< XtNFont #);
foreground:
(* A pixel value which indexes the widget's colormap to derive
 * the foreground colour of the widgets window.
 *)
IntegerResource(# resourceName::< XtNForeground #);
internalHeight:
(* The minimum amount of space to leave between the label or
 * bitmap and the horizontal edges of the window
 *)
ShortResource(# resourceName::< XtNinternalHeight #);
internalWidth:
(* The minimum amount of space to leave between the label or
 * bitmap and the vertical edges of the window
 *)
ShortResource(# resourceName::< XtNinternalWidth #);
justify:
(* Specifies left, center, or right alignment of label within
 * the widget. This value may be specified with the symbolic
 * constants XtJustifyLeft, XtJustifyCenter, or
 * XtJustifyRight. This value only has noticeable effect when
 * the width of the widget is larger than necessary to display
 * the label.
 *)
IntegerResource(# resourceName::< XtNjustify #);
resize:
(* A boolean which specifies whether the widget should attempt
 * to resize to its preferred dimensions whenever one of it
 * attribute are modified. This attempt to resize may be denied
 * by the parent of this widget. The parent is always free to
 * resize the widget regardless of the state of resize.
 *)
BooleanResource(# resourceName::< XtNresize #);
#) (* label *);

```

**Grip:** Simple

```

(* The Grip widget provides a small rectangular region in which
 * user input events (such as ButtonPress or ButtonRelease) may be
 * handled. The most common use for the Grip widget is as an
 * attachment point for visually repositioning an object, such as
 * the pane border in a Paned widget.
 *)
(# <<SLOT gripLIB: attributes>>;

```

```

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then gripWidgetClass -> value if)
    #)
  do INNER
  #);

```

```

foreground:
(* A pixel value which indexes the widget's colormap to derive
 * the colour used to flood fill the entire Grip widget.
 *)
IntegerResource(# resourceName::< XtNForeground #);

```



```

installCallbacks:<
  (* Install the "callback" virtual. "callback" will be called
   * when a user event occurs.
   *)
  (# do ...; INNER #);
#) (* Grip *);

```

**Scrollbar:** Simple

```

(* The Scrollbar widget is a rectangular area containing a slide
 * region and a thumb (also known as a slide bar). A Scrollbar can
 * be used alone, as a value generator, or it can be used within a
 * composite widget (for example, a Viewport). A Scrollbar can be
 * oriented either vertically or horizontally.
 *
 * When a Scrollbar is created, it is drawn with the thumb in a
 * contrasting color. The thumb is normally used to scroll client
 * data and to give visual feedback on the percentage of the client
 * data that is visible.
 *
 * Each pointer button invokes a specific action. That is, given
 * either a vertical or horizontal orientation, the pointer button
 * actions will scroll or return data as appropriate for that
 * orientation. Pointer buttons 1 and 3 do not move the thumb
 * automatically. Instead, they return the pixel position of the
 * cursor on the scroll region. When pointer button 2 is clicked,
 * the thumb moves to the current pointer position. When pointer
 * button 2 is held down and the pointer is moved, the thumb follows
 * the pointer.
 *
 * The pointer cursor in the scroll region changes depending on the
 * current action. When no pointer button is pressed, the cursor
 * appears as a double-headed arrow that points in the direction
 * that scrolling can occur. When pointer button 1 or 3 is pressed,
 * the cursor appears as a single-headed arrow that points in the
 * logical direction that the client will move the data. When
 * pointer button 2 is pressed, the cursor appears as an arrow that
 * points to the thumb.
 *
 * While scrolling is in progress, the application receives
 * notification through callback virtual patterns. For both
 * discrete scrolling actions, the callback gives the pixel position
 * of the pointer when the button was released. For continuous
 * scrolling, the callback routine gives the current relative
 * position of the thumb. When the thumb is moved using pointer
 * button 2, the callback procedure is invoked continuously. When
 * either button 1 or 3 is pressed, the callback procedure is
 * invoked only when the button is released and the further binding
 * of the callback procedure is responsible for moving the thumb.
 *)
(# <<SLOT scrollbarLIB: attributes>>;

init:<
  (# widgetClass:<
    (#
      do INNER;
      (if value=0 then scrollbarWidgetClass->value if)
    #)
    do INNER
  #);
resolution:< (* The resolution to use in floatResource *)
  IntegerObject(# do 100 -> value ; INNER #);

  (* Scrollbar resources *)
floatResource:
  (# set: (* Set THIS(floatResource) to value/resolution *)

```



```

        (# value: @integer;
        enter value
        do ...;
        #);
get: (* Exits resolution*value of THIS(floatResource) *)
    (# value: @integer
    do ...;
    exit value
    #);
    resourceName:< IntegerObject;
enter set
exit get
#);
cursor:
    IntegerResource(# resourceName:< XtNCursor #);
foreground:
    (* A pixel value which indexes the widget's colormap to derive
    * the color used to draw the thumb.
    *)
    IntegerResource(# resourceName:< XtNforeGround#);
length:
    (* The height of a vertical scrollbar or the width of a
    * horizontal scrollbar.
    *)
    ShortResource(# resourceName:< XtNlength #);
minimumThumb:
    (* The smallest size, in pixels, to which the thumb can
    * shrink.
    *)
    ShortResource(# resourceName:< XtNMinimumThumb #);
scrollDCursor:
    (* This cursor is used when scrolling backward in a vertical
    * scrollbar.
    *)
    IntegerResource(# resourceName:< XtNScrollDCursor #);
scrollHCursor:
    (* This cursor is used when a horizontal scrollbar is
    * inactive.
    *)
    IntegerResource(# resourceName:< XtNScrollHCursor #);
scrollLCursor:
    (* This cursor is used when scrolling forward in a horizontal
    * scrollbar.
    *)
    IntegerResource(# resourceName:< XtNScrollLCursor #);
scrollRCursor:
    (* This cursor is used when scrolling backward in a horizontal
    * scrollbar, or when thumbing a vertical scrollbar.
    *)
    IntegerResource(# resourceName:< XtNScrollRCursor #);
scrollUCursor:
    (* This cursor is used when scrolling forward in a vertical
    * scrollbar, or when thumbing a horizontal scrollbar.
    *)
    IntegerResource(# resourceName:< XtNScrollUCursor #);
scrollVCursor:
    (* This cursor is used when a vertical scrollbar is inactive.
    *)
    IntegerResource(# resourceName:< XtNScrollVCursor #);
thumb:
    (* This pixmap is used to tile (or stipple) the thumb of the
    * scrollbar. If no tiling is desired, then set this attribute
    * to 0. This resource will accept either a bitmap or a pixmap
    * that is the same depth as the window.
    *)
    IntegerResource(# resourceName:< XtNThumb #);

```



```

thickness:
(* The width of a vertical scrollbar or the height of a
 * horizontal scrollbar.
 *)
ShortResource(# resourceName::< XtNThickNess #);

orientation:
(* The orientation is the direction that the thumb will be
 * allowed to move. This value can be either XtOrientVertical
 * or XtOrientHorizontal. The "vertical" pattern provides a
 * more convenient interface.
 *)
IntegerResource(# resourceName::< XtNOrientation #);

shown:
(* This is the size of the thumb, expressed as a percentage
 * (0-100) of the length of the scrollbar.
 *)
floatResource(# resourceName::< XtNShown #);

topOfThumb:
(* The location of the top of the thumb, as a percentage
 * (0-100) of the length of the scrollbar.
 *)
floatResource(# resourceName::< XtNTopOfThumb #);

vertical:
(* A boolean that specifies whether the direction of the
 * scrollbar is vertical or horizontal.
 *)
(# isVertical: @boolean
enter isVertical
do (if isVertical then
    XtOrientVertical -> orientation
    else
    XtOrientHorizontal -> orientation
    if)
#);

jumpProc:<
(* This pattern is invoked on each new position when button 2
 * is down and moves the thumb interactively.
 *)
(# percent: @integer enter percent do INNER #);

scrollProc:<
(* This pattern is invoked when pointer button 1 or 3 are
 * pressed for incremental scrolling. The local attribute
 * position is a signed integer. Button 1 returns a positive
 * value, and button 3 returns a negative value. In both cases,
 * the magnitude of the value is the distance of the pointer in
 * pixels from the top (or left) of the Scrollbar. The value
 * will never be greater than the length of the Scrollbar.
 *)
(# position: @integer enter position do INNER #);

installCallbacks:<
(* Install the jumpProc and scrollProc virtuals *)
(# do ...; INNER #);
#) (* Scrollbar *);

```

**Command:** Label

```

(* The Command widget is an area, often rectangular, that contains
 * a text label or bitmap image. Those selectable areas are often
 * referred to as "buttons". When the pointer cursor is on a button,
 * it becomes highlighted by drawing a rectangle around its
 * perimeter. This highlighting indicates that the button is ready
 * for selection. When pointer button 1 is pressed, the Command
 * widget indicates that it has been selected by reversing its
 * foreground and background colours. When the button is released,
 * the Command widget's virtual pattern callback will be invoked. If
 * the pointer is moved out of the widget before the button is

```



```

* released, the widget reverts to its normal foreground and
* background colours, and releasing the button has no effect. This
* behaviour allows the user to cancel an action.
*)
(# <<SLOT commandLIB: attributes>>;

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then commandWidgetClass -> value if)
    #)
  do INNER
  #);

  (* Constants used for specifying shapeStyle *)
rectangle: (# exit XmuShapeRectangle #);
oval: (# exit XmuShapeOval #);
ellipse: (# exit XmuShapeEllipse #);
roundedRectangle: (# exit XmuShapeRoundedRectangle #);

shapeStyle: IntegerResource
  (* Nonrectangular buttons may be created using this pattern.
  * Nonrectangular buttons are supported only on a X Window
  * server that supports the Shape Extension. If nonrectangular
  * buttons are specified for a server lacking this extension,
  * the shape is ignored and the widgets will be
  * rectangular. The following shapes are currently supported:
  * rectangle, oval, ellipse, and roundedRectangle.
  *)
  (# resourceName::< XtNshapeStyle #);

cornerRoundPercent: ShortResource
  (* When a ShapeStyle of roundedRectangle is used, this
  * attribute controls the radius of the rounded corner. The
  * radius of the rounded corners is specified as a percentage
  * of the length of the shortest side of the widget.
  *)
  (# resourceName::< XtNCornerRoundPercent #);

highlightThickness:
  (* The thickness of the rectangle that is used to highlight
  * the internal border of this widget, alerting the user that
  * it is ready to be selected. The default value is 2 pixels if
  * the shapeStyle is rectangle, and 0 pixels (no highlighting)
  * otherwise.
  *)
  ShortResource(# resourceName::< XtNHighlightThickness #);

installCallbacks::<
  (* Install the "callback" virtual *)
  (# do ...; INNER #);
#) (* Command *)

```

#### **Form:** Constraint

```

(* The Form widget can contain an arbitrary number of children or
* subwidgets. The Form provides geometry management for its
* children, which allows individual control of the position of each
* child. Any combination of children can be added to a Form. The
* initial positions of the children may be computed relative to the
* positions of other children. When the Form is resized, it
* computes new positions and sizes for its children. This
* computation is based upon information provided for each child.
*
* The default width of the Form is the minimum width needed to
* enclose the children after computing their initial layout, with a
* margin of defaultDistance at the right and bottom edges. If a
* width and height is assigned to the Form that is too small for

```



```

* the layout, the children will be clipped by the right and bottom
* edges of the Form.
*)
(# <<SLOT FormLIB: attributes>>;

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then formWidgetClass -> value if)
    #)
  do INNER
  #);

defaultDistance:
  (* The default internal spacing for the children. This is the
   * default value for the attributes horizDistance and
   * vertDistance for the children.
   *)
  IntegerResource(# resourceName::< XtNDefaultDistance #);

(* Convenience routines for chaining the children *)
FixChildSize:
  (* Chain vertical edges to left, horizontal edges to top *)
  (# child: ^THIS(xtenv).core;
  enter child[]
  do ...;
  #);

(* Constants used for specifying constraints on children *)
chainTop: (# exit XawChainTop #);
chainLeft: (# exit XawChainLeft #);
chainBottom: (# exit XawChainBottom #);
chainRight: (# exit XawChainRight #);
rubber: (# exit XawRubber #);

doLayout:
  (* Force or defer re-layout of THIS(form). Takes a boolean
   * parameter that specifies if changes to the children should
   * result in a new layout. When making several changes to the
   * children of a Form widget after the Form has been realized,
   * it is a good idea to disable relayout until after all
   * changes have been made.
   *)
  (# doIt: @boolean
  enter doIt
  do (if doIt then
      (thewidget,1) -> xawFormDoLayout
    else
      (thewidget,0) -> xawFormDoLayout
    if);
  #);
#) (* Form *);

Paned: Constraint
  (* The Paned widget manages children in a vertically or
   * horizontally tiled fashion. The panes may be dynamically resized
   * by the user by using the grips that appear near the right or
   * bottom edge of the border between two panes.
   *
   * The Paned widget may accept any widget class as a pane except
   * Grip. Grip widgets have a special meaning for the Paned widget,
   * and adding a Grip as its own pane will confuse the Paned widget.
   *
   * The grips allow the panes to be resized by the user. The
   * semantics of how these panes resize is somewhat complicated, and

```



```

* warrants further explanation here. When the mouse pointer is
* positioned on a grip and pressed, an arrow is displayed that
* indicates the pane that is to be to be resized. While keeping the
* mouse button down, the user can move the grip up and down (or
* left and right). This, in turn, changes the size of the pane. The
* size of the Paned widget will not change. Instead, it chooses
* another pane (or panes) to resize.
*)
(# <<SLOT panedLIB: attributes>>;

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then panedWidgetClass -> value if)
    #)
  do INNER
  #);

cursor:
  (* The cursor to use when the mouse pointer is over the Paned
  * widget, but not in any of its children (children may also
  * inherit this cursor). It should be noted that the internal
  * borders are actually part of the Paned widget, not the
  * children.
  *)
  IntegerResource(# resourceName::< XtNcursor #);

gripCursor:
  (* The cursor to use when the grips are not active. The
  * default value is verticalGripCursor or horizontalGripCursor
  * depending on the orientation of the Paned widget.
  *)
  IntegerResource(# resourceName::< XtNGripCursor #);

horizontalBetweenCursor:
  (* The cursor to be used for the grip when changing the
  * boundary between two panes. This attribute allows the
  * cursors to be different depending on the orientation of the
  * Paned widget.
  *)
  IntegerResource(# resourceName::<XtNhorizontalBetweenCursor #);

verticalBetweenCursor:
  (* The cursor to be used for the grip when changing the
  * boundary between two panes. This attribute allows the
  * cursors to be different depending on the orientation of the
  * Paned widget.
  *)
  IntegerResource(# resourceName::< XtNverticalBetweenCursor #);

horizontalGripCursor:
  (* The cursor to be used for the grips when they are not
  * active. This attribute allows the cursors to be different
  * depending on the orientation of the Paned widget.
  *)
  IntegerResource(# resourceName::< XtNhorizontalGripCursor #);

verticalGripCursor:
  (* The cursor to be used for the grips when they are not
  * active. This attribute allows the cursors to be different
  * depending on the orientation of the Paned widget.
  *)
  IntegerResource(# resourceName::< XtNverticalGripCursor #);

gripIndent:
  (* The amount of space left between the right (or bottom) edge
  * of the Paned widget and all the grips.
  *)
  ShortResource(# resourceName::< XtNGripIndent #);

gripTranslation:
  (* Translation table that will be applied to all grips *)

```



```

IntegerResource(# resourceName::< XtNGripTranslation #);
internalBorderWidth:
(* The width of the internal borders. This is the amount of
 * space left between the panes.
 *)
ShortResource(# resourceName::< XtNInternalBorderWidth #);
internalBorderColor:
(* A pixel value which indexes the widget's colormap to derive
 * the internal border colour of the widget's window.
 *)
IntegerResource(# resourceName::< XtNInternalBorderColor #);
leftCursor:
(* The cursor used to indicate that the left pane is the
 * important one to resize when the Paned widget is oriented
 * horizontally.
 *)
IntegerResource(# resourceName::< XtNLeftCursor #);
rightCursor:
(* The cursor used to indicate that the right pane is the
 * important one to resize when the Paned widget is oriented
 * horizontally.
 *)
IntegerResource(# resourceName::< XtNRightCursor #);
lowerCursor:
(* The cursor used to indicate that the pane below is the
 * important one to resize when the Paned widget is oriented
 * vertically.
 *)
IntegerResource(# resourceName::< XtNLowerCursor #);
upperCursor:
(* The cursor used to indicate that the pane above is the
 * important one to resize when the Paned widget is oriented
 * vertically.
 *)
IntegerResource(# resourceName::< XtNUpperCursor #);
orientation:
(* The orientation to stack the panes. This value can be
 * either XtOrientVertical or XtOrientHorizontal. The
 * "vertical" pattern provides a more convenient interface.
 *)
IntegerResource(# resourceName::< XtNOrientation #);
refigureMode:
(* This attribute allows pane layout to be suspended. If the
 * value is False, then no layout actions will be taken. This
 * may improve efficiency when adding or removing more than one
 * pane from the Paned widget.
 *)
BooleanResource(# resourceName::< XtNRefigureMode #);
vertical:
(* A boolean that specifies the orientation of the stack of
 * panes.
 *)
(# isVertical: @boolean
enter isVertical
do (if isVertical then
    XtOrientVertical -> orientation
    else
    XtOrientHorizontal -> orientation
    if)
#);

GetNumSub: IntegerValue
(* Exits the number of panes in THIS(Paned) *)
(# do theWidget -> XawPanedGetNumSub -> value #);
#) (* Paned *);

```



**Dialog:** Form

```
( * The Dialog widget implements a commonly used interaction
 * semantic to prompt for auxiliary input from a user. For example,
 * you can use a Dialog widget when an application requires a small
 * piece of information, such as a filename, from the user. A Dialog
 * widget, which is simply a special case of the Form widget,
 * provides a convenient way to create a preconfigured Form.
 *
 * The typical Dialog widget contains three areas. The first line
 * contains a description of the function of the Dialog widget, for
 * example, the string 'Filename:'. The second line contains an area
 * into which the user types input. The third line can contain
 * buttons that let the user confirm or cancel the Dialog input. Any
 * of these areas may be omitted by the application.
 *)
(# <<SLOT dialogLIB: attributes>>;

  init::<
    (# widgetClass::<
      (#
        do INNER;
        (if value=0 then dialogWidgetClass -> value if)
      #)
    do INNER
    #);

  label: ( * A string to be displayed at the top of THIS(Dialog) *)
    StringResource(# resourceName::< XtNLabel #);

  value:
    ( * An initial value for the string field that the user will
    * enter text into. By default, no text entry field is
    * available to the user. Specifying an initial value for
    * value activates the text entry field. If string input is
    * desired, but no initial value is to be specified then set
    * this attribute to '' (empty text).
    *)
    StringResource(# resourceName::< XtNValue #);

  icon:
    ( * A pixmap image to be displayed immediately to the left of
    * the label of THIS(Dialog).
    *)
    IntegerResource(# resourceName::< XtNIcon #);

  getValue: ( * Returns the text in the value field *)
    (# value: ^text; do ... exit value[] #);

  labelWidget: IntegerValue
    ( * The subwidget holding the label *)
    (# do (thewidget, 'label') -> XtNameToWidget -> value #);

  iconWidget: IntegerValue
    ( * The subwidget holding the icon *)
    (# do (thewidget, 'icon') -> XtNameToWidget -> value #);

  valueWidget: IntegerValue
    ( * The subwidget holding the value *)
    (# do (thewidget, 'value') -> XtNameToWidget -> value #);

  #) ( * Dialog *);
```

**Box:** Composite

```
( * The Box widget provides geometry management of arbitrary widgets
 * in a box of a specified dimension. The children are rearranged
 * when resizing events occur either on the Box or its children, or
 * when children are managed or unmanaged. The Box widget always
 * attempts to pack its children as tightly as possible within the
 * geometry allowed by its parent.
 *
 * Box widgets are commonly used to manage a related set of buttons
 * and are often called ButtonBox widgets, but the children are not
 * limited to buttons. The Box's children are arranged on a
```



```

* background that has its own specified dimensions and colour.
*)
(# <<SLOT boxLIB: attributes>>;

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then boxWidgetClass -> value if)
    #)
  do INNER
  #);

  (* Box resources *)
hSpace:
  (* The amount of space, in pixels, to leave between the
   * children. This attribute also specifies the amount of space
   * left between the outermost children and the vertical edge of
   * THIS(Box).
   *)
  ShortResource(# resourceName::< XtNHSpace #);
vSpace:
  (* The amount of space, in pixels, to leave between the
   * children. This attribute also specifies the amount of space
   * left between the outermost children and the horizontal edge
   * of THIS(Box).
   *)
  ShortResource(# resourceName::< XtNVSpace #);
orientation:
  (* Specifies whether the preferred shape of THIS(Box) is tall
   * and narrow (XawOrientVertical) or short and wide
   * (XawOrientHorizontal). The "vertical" pattern provides a
   * more convenient interface.
   *)
  IntegerResource(# resourceName::< XtNOrientation #);
vertical:
  (* A boolean that specifies whether the preferred shape of
   * THIS(Box) is tall and narrow or short and wide. When
   * THIS(Box) is a child of a parent which enforces width
   * constraints, it is usually better to specify true (the
   * default). When the parent enforces height constraints, it is
   * usually better to specify false.
   *)
  (# isVertical: @boolean
  enter isVertical
  do (if isVertical then
      XtOrientVertical-> orientation
    else
      XtOrientHorizontal -> orientation
    if)
  #);
#) (* box *);

```

**ViewPort: Form**

```

(* The ViewPort widget consists of a frame window, one or two
 * Scrollbars, and an INNER window. The size of the frame window is
 * determined by the viewing size of the data that is to be
 * displayed and the dimensions to which the ViewPort is
 * created. The INNER window is the full size of the data that is to
 * be displayed and is clipped by the frame window. The ViewPort
 * widget controls the scrolling of the data directly. No
 * application code are required for the scrolling.
 *
 * When the geometry of the frame window is equal in size to the
 * INNER window, or when the data does not require scrolling, the
 * ViewPort widget automatically removes any scrollbars. The

```



```

* forceBar option causes the ViewPort widget to display all
* scrollbars permanently.
*)
(# <<SLOT ViewPortLIB: attributes>>;

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then ViewPortWidgetClass -> value if)
    (#)
  do INNER
  #);

allowHoriz:
  (* If this attributes is False then THIS(ViewPort) will never
  * create a horizontal scrollbar. If it is True then the
  * scrollbar will only appear when it is needed, unless
  * forceBars is True.
  *)
  BooleanResource(# resourceName::< XtNAllowHoriz #);
allowVert:
  (* If this attributes is False then THIS(ViewPort) will never
  * create a vertical scrollbar. If it is True then the
  * scrollbar will only appear when it is needed, unless
  * forceBars is True.
  *)
  BooleanResource(# resourceName::< XtNAllowVert #);
forceBars:
  (* When True the scrollbars that have been allowed will always
  * be visible on the screen. If False the scrollbars will be
  * visible only when the INNER window is larger than the frame.
  *)
  BooleanResource(# resourceName::< XtNForceBars #);
useBottom:
  (* By default the scrollbars appear on the left and top of the
  * screen. This attribute allows the horizontal scrollbar to
  * be placed on the bottom edge of THIS(ViewPort).
  *)
  BooleanResource(# resourceName::< XtNuseBottom #);
useRight:
  (* By default the scrollbars appear on the left and top of the
  * screen. This attribute allows the vertical scrollbar to be
  * placed on the right edge of THIS(ViewPort).
  *)
  BooleanResource(# resourceName::< XtNuseRight #);
#) (* ViewPort *);

```

#### ListWidget: Simple

```

(* The List widget contains a list of strings formatted into rows
* and columns. When one of the strings is selected, it is
* highlighted, and the List widget's virtual pattern callback is
* invoked. Only one string may be selected at a time.
*)
(# <<SLOT listWidgetLIB: attributes>>;

init::< (# widgetClass::< (# do ... #);
  do INNER;
  #);
strings:< StringArray
  (* Further bind to specify initial contents *)
  (# #);
getStrings:
  (* The current content of THIS(ListWidget). Updated by "init"
  * and "change".
  *)

```



```

    (# s: ^StringArray
    do ...
    exit s[]
    #);

(* ListWidget resources *)
list:
    (* An array of text strings displayed in THIS(ListWidget). *)
    StringArrayResource(# resourceName:< XtNList #);
columnSpacing:
    (* The amount of space, in pixels, between each of the columns
    * in THIS(ListWidget).
    *)
    ShortResource(# resourceName:< XtNColumnSpacing #);
rowSpacing:
    (* The amount of space, in pixels, between each of the rows in
    * THIS(ListWidget).
    *)
    ShortResource(# resourceName:< XtNRowSpacing #);
defaultColumns:
    (* The default number of columns. This value is used when
    * neither the width nor the height of THIS(ListWidget) is
    * specified or when forceColumns is True.
    *)
    IntegerResource(# resourceName:< XtNDefaultColumns #);
forceColumns:
    (* If this boolean is true it forces the default number of
    * columns to be used regardless of THIS(ListWidget)'s current
    * size.
    *)
    BooleanResource(# resourceName:< XtNForceColumns #);
foreground:
    (* A pixel value which indexes the widget's colormap to derive
    * the colour used to paint the text of the list elements.
    *)
    IntegerResource(# resourceName:< XtNForeground #);
pasteBuffer:
    (* If this attribute is set to True then the name of the
    * currently selected list element will be put into the
    * clipboard of the X Window System.
    *)
    BooleanResource(# resourceName:< XtNPasteBuffer #);
internalHeight:
    (* The minimum amount of space to leave between edges of the
    * list and the horizontal edges of THIS(ListWidget)
    *)
    IntegerResource(# resourceName:< XtNInternalHeight #);
internalWidth:
    (* The minimum amount of space to leave between edges of the
    * list and the vertical edges of THIS(ListWidget)
    *)
    IntegerResource(# resourceName:< XtNInternalWidth #);
longest:
    (* Specifies the width, in pixels, of the longest string in
    * the current list. THIS(ListWidget) will compute this value
    * if zero (the default) is specified.
    *)
    IntegerResource(# resourceName:< XtNlongest #);
numberStrings:
    (* The number of strings in the current list. If a value of
    * zero (the default) is specified, the List widget will
    * compute it.
    *)
    IntegerResource(# resourceName:< XtNNumberStrings #);

change:

```



```

(* To change the elements of THIS(ListWidget) this method must
 * be called. It takes as enter parameter a StringArray (list)
 * and a boolean (resize). List specifies the new list to
 * display. If resize is true, it specifies that
 * THIS(ListWidget) should try to resize itself after making
 * the change.
 *)
(# list: ^StringArray;
  resize: @boolean
  enter (list[],resize)
  do ...;
  #);

callback::<
  (* This pattern is invoked when an item in the list have been
   * selected. Current.listindex is an integer which contains
   * the index of the item that have been selected. The indexing
   * starts from 0, e.g. if the second item of the list have been
   * selected, current.listIndex will be 1. Current.string exits
   * the string in the selected item.
   *)
  (# current: @XawListReturnStruct
  do theWidget -> XawListShowCurrent -> current.ptr;
  INNER;
  #);
installcallbacks::< (* Install the "callback" virtual *)
  (# do ...; INNER #);
private: @...;
#) (* ListWidget *);

AsciiText: Simple
(* The AsciiText widget provides a window that will allow an
 * application to display and edit one or more lines of
 * text. Options are provided to allow the user to add Scrollbars to
 * its window, search for a specific string, and modify the text in
 * the buffer.
 *)
(# <<SLOT asciiTextLIB: attributes>>;

init::<
  (# widgetClass::<
    (#
      do INNER;
      (if value=0 then asciiTextWidgetClass -> value if)
    #)
  do INNER
  #);
installCallbacks::<
  (* Install the "callback" virtual. It will be called every
   * time the text buffer changes
   *)
  (# do ...; INNER #);

  (* AsciiText resources *)
string:
  (* If type is AsciiString then this string contains the buffer
   * to be displayed in the widget. If type is AsciiFile then the
   * string contains the name of the file to be displayed.
   *)
  StringResource(# resourceName::< XtNString #);
editType:
  (* This is the type of editing that will be allowed in
   * THIS(AsciiText). Legal values are read, edit, and append.
   *)
  IntegerResource(# resourceName::< XtNEditType #);
autoFill:

```



```

(* If this attribute is True THIS(AsciiText) will
 * automatically break a line when the user attempts to type
 * into the right margin. The attribute has no effect on files
 * or text inserted into the asciiText widget. It only checks
 * to see if the action should be taken when a user enters a
 * new character.
 *)
BooleanResource(# resourceName:< XtNAutoFill #);

bottomMargin:
(* The amount of space, in pixels, between the bottom edge of
 * the window and the bottom edge of the text within the
 * window. If there is a scrollbar active on this edge, then
 * this is the space between the text and the scrollbar.
 *)
ShortResource(# resourceName:< XtNBottomMargin #);

topMargin:
(* The amount of space, in pixels, between the top edge of the
 * window and the top edge of the text within the window. If
 * there is a scrollbar active on this edge, then this is the
 * space between the text and the scrollbar.
 *)
ShortResource(# resourceName:< XtNTopMargin #);

rightMargin:
(* The amount of space, in pixels, between the right edge of
 * the window and the right edge of the text within the
 * window. If there is a scrollbar active on this edge, then
 * this is the space between the text and the scrollbar.
 *)
ShortResource(# resourceName:< XtNRightMargin #);

leftMargin:
(* The amount of space, in pixels, between the left edge of
 * the window and the left edge of the text within the
 * window. If there is a scrollbar active on this edge, then
 * this is the space between the text and the scrollbar.
 *)
ShortResource(# resourceName:< XtNLeftMargin #);

displayPosition:
(* The position in the text buffer of the character that is
 * currently displayed in the upper left hand corner of the
 * text display.
 *)
IntegerResource(# resourceName:< XtNdisplayPosition #);

insertPosition:
(* This is the location of the insert point. It is expressed
 * in characters from the beginning of the text. The cursor
 * will always be forced to be on the screen. This attribute
 * may therefore be used to scroll the text display to a
 * certain character position.
 *)
IntegerResource(# resourceName:< XtNinsertPosition #);

resize:
(* Controls whether or not THIS(AsciiText) attempts to resize
 * itself when it is no longer able to display the full text
 * buffer in the associated window. Any attempt by
 * THIS(AsciiText) to resize itself is always subject to the
 * constraints imposed by its parent. The values resizeNever,
 * resizeWidth, resizeHeight, and resizeBoth are all acceptable
 * for this attribute.
 *)
BooleanResource(# resourceName:< XtNresize #);

scrollHorizontal:
(* This attribute controls the placement of scrollbar on the
 * left edge of THIS(AsciiText). The values accepted are
 * scrollAlways, scrollWhenNeeded, and scrollNever. If
 * scrollWhenNeeded is specified, the appropriate scrollbar
 * will only appear when there is text in the buffer that is

```



```

    * not able to fit within the bounds of the current window.
    * The scrollbar will disappear when the text once again fits
    * within the window.
    *)
IntegerResource(# resourceName::< XtNscrollHorizontal #);
scrollVertical:
(* This attribute controls the placement of scrollbar on the
 * bottom edge of THIS(AsciiText). The values accepted are
 * scrollAlways, scrollWhenNeeded, and scrollNever. If
 * scrollWhenNeeded is specified, the appropriate scrollbar
 * will only appear when there is text in the buffer that is
 * not able to fit within the bounds of the current window.
 * The scrollbar will disappear when the text once again fits
 * within the window.
 *)
IntegerResource(# resourceName::< XtNscrollVertical #);
selectTypes:
(* Specifies the selection type array that is used when
 * multi-click is activated. Only for the advanced programmer.
 *)
IntegerResource(# resourceName::< XtNselectTypes #);
textSink:
(* The TextSink subwidget used by THIS(AsciiText). Only to be
 * used by the advanced programmer
 *)
IntegerResource(# resourceName::< XtNtextSink #);
textSource:
(* The TextSource subwidget used by THIS(AsciiText). Only to
 * be used by the advanced programmer
 *)
IntegerResource(# resourceName::< XtNtextSource #);
wrap:
(* When the text in any one line is wider than the window
 * there are several possible actions. This attribute allows
 * the user to decide what will happen. The accepted values for
 * this attribute are wrapNever, wrapLine, and wrapWord. With
 * wrapLine all text that is beyond the right edge of the
 * window will be displayed on the next line. With wrapWord the
 * same action occurs but the text is broken at a word boundary
 * if possible. If no wrapping is enabled then the text will
 * extend off the edge of the window, and a small rectangle
 * will be painted in the right margin to alert the user that
 * this line is too long.
 *)
IntegerResource(# resourceName::< XtNwrap #);
dataCompression:
(* The AsciiText uses an algorithm that may cause the text
 * buffer to grow to about twice the size of the actual text
 * over time, as the text is edited. On systems where CPU
 * cycles are cheaper than memory, it is helpful to spend some
 * extra time to compress this buffer back to its minimum
 * size. If this attribute is True, the asciiText will compress
 * its data to the minimum size required every time the text
 * string is saved, or the value of the string is queried.
 *)
BooleanResource(# resourceName::< XtNdataCompression #);
echo:
(* Whether or not to echo characters to the screen. The buffer
 * can still be edited, but nothing is displayed. This mode can
 * be useful for entering passwords and other sensitive
 * information.
 *)
IntegerResource(# resourceName::< XtNecho #);
font:
(* This is font for rendering all text and must be a character
 * cell font

```



```

    *)
    IntegerResource(# resourceName:< XtNfont #);
foreground:
    (* A pixel value which indexes the asciiText widget's colormap
    * to derive the foreground colour.
    *)
    IntegerResource(# resourceName:< XtNForeground #);
length:
    (* If the useStringInPlace resource is False (default), this
    * resource has no effect. If that resource is true, however,
    * then the length resource specifies the length of the buffer
    * passed to the text widget in the string resource.
    *)
    IntegerResource(# resourceName:< XtNlength #);
pieceSize:
    (* This is the size of the internal chunks into which the text
    * buffer is broken down for memory management. The larger this
    * value the less segmented your memory will be, but the slower
    * your editing will be. THIS(AsciiText) will always allocate
    * a chunk of memory this size to stuff the string into, so
    * when using small strings, having this buffer large can waste
    * memory.
    *)
    IntegerResource(# resourceName:< XtNpieceSize #);
type:
    (* This attribute may be either asciiString or asciiFile. The
    * value of this attribute determines whether the string
    * attribute contains the name of a file to be opened or a
    * buffer to be displayed by THIS(AsciiText).
    *)
    IntegerResource(# resourceName:< XtNtype #);
useStringInPlace:
    (* Setting this resource to Trus will disable the memoru
    * management provided by the Text widget, updating the string
    * resource in place. Using the string in place can be much
    * more efficient for text widgets that display static data, or
    * when the programmer wishes to impose strict constraints on
    * the contents of the string. When using the string in place
    * be sure that: The length of the string is specified by
    * setting the length resource, the type of THIS(AsciiText) is
    * asciiString, and that the string exists for the lifetime of
    * THIS(AsciiText), or until it has been reset.
    *)
    IntegerResource(# resourceName:< XtNUseStringInPlace #);
displayNonPrinting:
    (* If this attributes is True, THIS(AsciiText) will display
    * all non-printable characters as the string ^@. If False,
    * THIS(AsciiText) will just leave a blank space where a
    * non-printable character exists in the text buffer.
    *)
    BooleanResource(# resourceName:< XtNdisplayNonprinting #);

(* Text selection *)
selection: @
    (* The text that is selected by the AsciiText widget. Is
    * expressed as a start- and an end-position. The first
    * character in the text buffer is at position 0
    *)
    (# set:
        (# begin,end: @integer
        enter (begin,end)
        do (theWidget,begin,end) -> XawTextSetSelection;
        #);
        unset: (* Unhighlight the previous highlighted text *)
        (# do theWidget -> XawTextUnsetSelection #);
        get:

```



```

        (# begin,end: @XawTextPosition;
        do (theWidget,begin[],end[]) -> xawTextGetSelectionPos;
        exit (begin,end)
        #);
enter set
exit get
#);

(* Conversion between text buffer positions and line and column
 * position
 *)
posToLineAndColumn:
(* Convert a text buffer position to the line number and the
 * column number on that line. Ranges: The position of the
 * first character in the buffer is 0. The line number for the
 * first line is 1. The first character on a line is at column
 * 1. If the position entered is negative, 0 is used, and if
 * it is larger than the last position in the text buffer, the
 * last position is used instead. The column number may be
 * incorrect if TABs are present on the line.
 *)
(# position: @integer;
 line, column: @integer;
 enter position
 do ...;
 exit (line, column)
 #);
lineAndColumnToPos:
(* Converts a line number and a column number to a text buffer
 * position. Ranges: The position of the first character in
 * the buffer is 0. The line number for the first line is 1.
 * The first character on a line is at column 1. If the
 * entered line or column number is not positive, 1 is used
 * instead. If the entered line number is larger than the
 * number of the last line in the text buffer, the number of
 * the last line is used instead. If the entered column number
 * is larger than the number of columns on the line, the column
 * of the last character on the line is used instead. The text
 * buffer position may be incorrect if TABs are present on the
 * line.
 *)
(# line, column: @integer;
 position: @integer;
 enter (line,column)
 do ...
 exit position
 #);
displayCaret:
(* Make the caret be visible/invisible *)
(# displayIt: @boolean;
 enter displayIt
 do (if displayIt then
      (theWidget, 1) -> XawTextDisplayCaret;
    else
      (theWidget, 0) -> XawTextDisplayCaret;
    if)
 #);
save:
(* If type is asciiFile, this operation saves the contents of
 * the text buffer in the file specified in the string
 * resource.
 *)
(# do ... #);
saveAs:
(* Saves the contents of the text buffer in the file with the
 * name entered

```



```

    *)
    (# filename: ^text;
    enter filename[]
    do ...
    #);
changed:
    (* Exits a boolean indicating if the text buffer has been
    * changed since it was last saved or queried for changes.
    *)
    (# value: @boolean do ... exit value #);

    (* Constants for setting various resources *)

read: (# exit XawTextRead #);
edit: (# exit XawTextEdit #);
append: (# exit XawTextAppend #);

resizeNever: (# exit XawTextResizeNever #);
resizeWidth: (# exit XawTextResizeWidth #);
resizeHeight: (# exit XawTextResizeHeight #);
resizeBoth: (# exit XawTextResizeBoth #);

scrollAlways: (# exit XawTextScrollAlways #);
scrollWhenNeeded: (# exit XawTextScrollWhenNeeded #);
scrollNever: (# exit XawTextScrollNever #);

wrapNever: (# exit XawTextwrapNever #);
wrapLine: (# exit XawTextwrapLine #);
wrapWord: (# exit XawTextwrapWord #);

asciiFile: (# exit XawAsciiFile #);
asciiString: (# exit XawAsciiString #);

#) (* AsciiText *);

(* Alises to use in Composite *)
blabel: label(# #);
bcommand: command(# #);
bmenuButton: menuButton(# #);
btoggle: toggle(# #);
bsimple: simple(# #);
bstripChart: stripChart(# #);
bgrip: grip(# #);
bscrollbar: scrollbar(# #);
bform: form(# #);
bpaned: paned(# #);
bdialog: dialog(# #);
bbox: box(# #);
bviewport: ViewPort(# #);
blistWidget: listWidget(# #);
basciiText: asciiText(# #);
bsme: sme(# #);
bsmeBSB: smeBSB(# #);
bsmeLine: smeLine(# #);
bsmeCascade: smeCascade(# #);

--- compositeLib: attributes ---

(* Redefinition of patterns within Composites: Use the Composite as
* default father
*)
Label: bLabel
    (# init:<
        (# getFatherWidget:< (# do THIS(composite).theWidget->value #);
        do INNER #)
    #);

```



```

Command: bCommand
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Toggle: bToggle
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
MenuButton: bMenuButton
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Simple: bSimple
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
StripChart: bStripChart
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Grip: bGrip
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Scrollbar: bScrollbar
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Form: bForm
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Paned: bPaned
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Dialog: bDialog
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
Box: bBox
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
ViewPort: bViewPort
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
ListWidget: bListWidget
  (# init::<
    (# getFatherWidget::< (# do THIS(composite).theWidget->value #);
    do INNER #)
  #);
AsciiText: bAsciiText

```



```

    (# init::<
      (# getFatherWidget::<(# do THIS(composite).theWidget->value #);
      do INNER #)
    #);

--- CoreLIB: attributes ---

(* Extra Constraint Attributes for children of paned. Each child of
 * the Paned widget has a set of operations available to control the
 * layout. These attributes allow the Paned widget's children to
 * specify individual layout requirements.
 *)
allowResize:
  (* If this value is False, the Paned widget will disallow all
   * geometry requests from this child.
   *)
  BooleanResource(# resourceName::< XtNAllowResize #);
minSize:
  (* The absolute minimum size for this pane. This value will never
   * be overridden by the Paned widget. This may case some panes to
   * pushed off the bottom (or right) edge of the paned widget.
   *)
  IntegerResource(# resourceName::< XtNMin #);
maxSize:
  (* The absolute maximum size for this pane. This value will never
   * be overridden by the Paned widget. This may case some panes to
   * pushed off the bottom (or right) edge of the paned widget.
   *)
  IntegerResource(# resourceName::< XtNMax #);
preferredPaneSize:
  (* Normally the paned widget asks the child to determine the pre-
   * ferred size of the child's pane. There are times when the
   * application programmer has a better idea of the preferred size of
   * a pane. Setting this attribute causes the value passed to be
   * interpreted as the preferred size, in pixels, of this pane.
   *)
  IntegerResource(# resourceName::< XtNpreferredPaneSize #);
resizeToPreferred:
  (* Determines whether or not to resize each pane to its preferred
   * size when the Paned widget is resized.
   *)
  BooleanResource(# resourceName::< XtNresizeToPreferred#);
showGrip:
  (* If True then a grip will be shown for this pane. The grip
   * associated with a pane is either below or to the right of the
   * pane. No grip is ever shown for the last pane.
   *)
  BooleanResource(# resourceName::< XtNShowGrip #);
skipAdjust:
  (* This attribute is used to determine which pane is forced to be
   * resized. Setting this value to True make this pane less likely to
   * be forced to be resized.
   *)
  BooleanResource(# resourceName::< XtNSkipAdjust #);

(* Extra Constraint Attributes for children of Form. Each child of the
 * Form widget has a set of operations available to control the
 * layout. These attributes allow the Form widget's children to
 * specify individual layout requirements.
 *)
fromHoriz:
  (* Which widget this child should be placed to the right of. If
   * this attribute is not specified then this widget will be
   * positioned relative to the edge of the parent.
   *)
  IntegerResource(# resourceName::< XtNFromHoriz #);

```



```

fromVert:
  (* Which widget this child should be placed underneath.  If this
   * attribute is not specified then this widget will be positioned
   * relative to the edge of the parent.
   *)
  IntegerResource(# resourceName::< XtNFromVert #);
horizDistance:
  (* The amount of space, in pixels, between this child and its left
   * neighbour.
   *)
  IntegerResource(# resourceName::< XtNhorizDistance #);
vertDistance:
  (* The amount of space, in pixels, between this child and its upper
   * neighbour
   *)
  IntegerResource(# resourceName::< XtNvertDistance #);
resizable:
  (* If this attribute is False then the parent widget will ignore
   * all geometry request made by this child. The parent may still
   * resize this child itself, however.
   *)
  BooleanResource(# resourceName::< XtNResizable #);
(* The following four resources specify what to do with the
 * corresponding edge of the child when the parent is resized. The
 * value may be chainBottom, chainLeft, chainRight, chainTop or rubber
 *)
bottom: IntegerResource(# resourceName::< XtNBottom #);
left: IntegerResource(# resourceName::< XtNLeft #);
right: IntegerResource(# resourceName::< XtNRight #);
top: IntegerResource(# resourceName::< XtNTop #);

--- LIB: attributes ---

AwEnv: XtEnv
  (# <<SLOT AwEnvLIB: attributes>>;
   do INNER
   #)

```

---

Awenv Interface

[Mjølner Informatics](#)



# Prompts Interface

```
ORIGIN '../awenv';
BODY 'private/promptsbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

---- xtenvLib: attributes ----

Prompt:
(* A pattern which sets up a dialogbox containing a label and an OK
 * button, which when clicked, dismisses THIS(Prompt). Typing
 * <RETURN> is the same as clicking on the OK button.
 *)
(# message: ^text; (* The text in the label field *)
  icon: @integer
    (* If not 0, this is a bitmap to display next to the prompt *);
  okText: ^text; (* The label of the OK button, default 'OK' *)
  x,y: @integer; (* Where to place the dialogbox on the screen *)
  doubleBorder:< booleanObject
    (* If true, the dialog gets a double border in the Macintosh
     * style
     *);
  validate:< BooleanValue
    (* Called when the OK button is pressed. THIS(Prompt) is only
     * dismissed (and the ok virtual called) if validate returns
     * true.
     *)
    (# do true->value; INNER #);

  show:< (* Make the dialog visible on the screen *)
    (# do INNER; ...; #);
  ok:< object (* Called when the OK button is clicked *);

  <<SLOT PromptLIB: attributes>>;
  additems:< object (* private *);
  private: @...;
enter (x, y, icon, message[], okText[])
do show
#) (* Prompt *);

PromptForString: Prompt
(* A pattern which sets up a dialogbox that asks the user to enter
 * a text. It contains a label, a text entry field an OK and
 * optionally a Cancel button. The pointer need not be inside the
 * text entry field when typing. Typing <RETURN> is the same as
 * clicking on OK, and typing <ESCAPE> is the same as clicking on
 * Cancel.
 *)
(# initialValue: ^text; (* The initial text in the text entry field. *)
  cancelText: ^text
    (* The label of the Cancel button. If NONE, no cancel button
     * will be present
     *)
  cancel:< object(* Called when the Cancel button is clicked *);
  validate:<
    (# theText: ^text
      (* The value of the text entry field when validate is
       * invoked
       *)
    );
```



```

...
#);
ok::<
  (# value: ^text
    (* The value of the text entry field when OK was clicked *));
  do INNER
  #);

show::< (# do ...; #);
additems::<(# do ... #);
<<SLOT promptForStringLIB: attributes>>
enter (cancelText[], initialValue[])
#) (* promptForString *);

```

**PromptForBoolean:** Prompt

```

(* A pattern which sets up a dialogbox that asks the user to enter
* a boolean. It contains a label, two buttons for the answer and
* optionally a Cancel button. Typing <RETURN> is the same as
* clicking on the OK button, and typing <ESCAPE> is the same as
* clicking on Cancel. Also the first letter (in either case) of the
* labels of the OK and No buttons, respectively, may be typed to
* choose that button.
*)
(# noText: ^text; (* The label of the No button, default No' *)
  cancelText: ^text
    (* The label of the Cancel button. If NONE, no cancel button
    * will be present
    *);
  no:< object (* Called when the No button is clicked *);
  cancel:< object (* Called when the Cancel button is clicked *);

  additems::<(# do ... #);
  <<SLOT promptForBooleanLIB: attributes>>
enter (noText[], cancelText[])
#)

```

---



# Heapview Interface

```
ORIGIN '../awenv';
BODY 'private/heapviewbody';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- xtenvlib: attributes --

HeapView: ToplevelShell
  (* Use an instance of HeapView if you want to display the
   * heap usage of your application.
   *)
  (# private: @...;
   init: < (# do ... #);
   #);

openHeapView:
  (# h: ^HeapView
   do &HeapView[] -> h[];
   ('Beta Heap View',toplevel) -> h.init
   #)
```

---

Heapview Interface

[Mjolner Informatics](#)



# Textactions Interface

```
ORIGIN '../awenv';

(*
 * COPYRIGHT
 *      Copyright Mjolner Informatics, 1992-97
 *      All rights reserved.
 *)

-- AsciiTextLib: attributes --

(* motion bindings *)

forward_character:
(* Move the insert point forward one character in the buffer.  If
 * the insert point is at the end or beginning of a line this action
 * will move the insert point to the next line.
 *)
(# do 'forward-character' -> CallAction #);

backward_character:
(* Move the insert point backward one character in the buffer.  If
 * the insert point is at the end or beginning of a line this action
 * will move the insert point to the previous line.
 *)
(# do 'backward-character' -> CallAction #);

forward_word:
(* Move the insert point to the next word boundary.  A word
 * boundary is defined as a Space, Tab or Carriage Return.
 *)
(# do 'forward-word' -> CallAction #);

backward_word:
(* Move the insert point to the previous word boundary.  A word
 * boundary is defined as a Space, Tab or Carriage Return.
 *)
(# do 'backward-word' -> CallAction #);

forward_paragraph:
(* Move the insert point to the next paragraph boundary.  A
 * paragraph boundary is defined as two Carriage Returns in a row
 * with only Spaces or Tabs between them.
 *)
(# do 'forward-paragraph' -> CallAction #);

backward_paragraph:
(* Move the insert point to the previous paragraph boundary.  A
 * paragraph boundary is defined as two Carriage Returns in a row
 * with only Spaces or Tabs between them.
 *)
(# do 'backward-paragraph' -> CallAction #);

beginning_of_line:
(* Move to the beginning of the current line.  If the insert point
 * is already at the beginning of the line then no action is taken.
 *)
(# do 'beginning-of-line' -> CallAction #);

end_of_line:
(* Move to the beginning of the current line.  If the insert point
 * is already at the beginning of the line then no action is taken.
 *)
(# do 'end-of-line' -> CallAction #);

next_line:
(* Move the insert point down one line.  If the insert point is
 * currently N characters from the beginning of the line then it
 * will be N characters from the beginning of the next line.  If N
 * is past the end of the line, the insert point is placed at the
 * end of the line.
```



```

*)
(# do 'next-line' -> CallAction #);
previous_line:
(* Move the insert point up one line.  If the insert point is
 * currently N characters from the beginning of the line then it
 * will be N characters from the beginning of the previous line.  If
 * N is past the end of the line, the insert point is placed at the
 * end of the line.
 *)
(# do 'previous-line' -> CallAction #);
next_page:
(* Move the insert point down one page in the file.  One page is
 * defined as the current height of the text widget.  The insert
 * point is always placed at the first character of the top line by
 * this action.
 *)
(# do 'next-page' -> CallAction #);
previous_page:
(* Move the insert point up one page in the file.  One page is
 * defined as the current height of the text widget.  The insert
 * point is always placed at the first character of the top line by
 * this action.
 *)
(# do 'previous-page' -> CallAction #);
beginning_of_file:
(* Place the insert point at the beginning of the current text
 * buffer.  The text widget is then scrolled the minimum amount
 * necessary to make the new insert point location visible.
 *)
(# do 'beginning-of-file' -> CallAction #);
end_of_file:
(* Place the insert point at the end of the current text buffer.
 * The text widget is then scrolled the minimum amount necessary to
 * make the new insert point location visible.
 *)
(# do 'end-of-file' -> CallAction #);
scroll_one_line_up:
(* Scroll the current text field up by one line.  This does not
 * move the insert point.  Other than the scrollbars this is the
 * only way that the insert point may be moved off of the visible
 * text area.  The widget will be scrolled so that the insert point
 * is back on the screen as soon as some other action is executed.
 *)
(# do 'scroll-one-line-up' -> CallAction #);
scroll_one_line_down:
(* Scroll the current text field down by one line.  This does not
 * move the insert point.  Other than the scrollbars this is the
 * only way that the insert point may be moved off of the visible
 * text area.  The widget will be scrolled so that the insert point
 * is back on the screen as soon as some other action is executed.
 *)
(# do 'scroll-one-line-down' -> CallAction #);

(* delete bindings *)

delete_next_character:
(* Remove the character immediately after the insert point.  If a
 * Carriage Return is removed then the next line is appended to the
 * end of the current line.
 *)
(# do 'delete-next-character' -> CallAction #);
delete_previous_character:
(* Remove the character immediately before the insert point.  If a
 * Carriage Return is removed then the next line is appended to the
 * end of the current line.
 *)

```



```

    (# do 'delete-previous-character' -> CallAction #);
delete_next_word:
    (* Remove all characters between the insert point location and the
    * next word boundary. A word boundary is defined as a Space, Tab
    * or Carriage Return.
    *)
    (# do 'delete-next-word' -> CallAction #);
delete_previous_word:
    (* Remove all characters between the insert point location and the
    * previous word boundary. A word boundary is defined as a Space,
    * Tab or Carriage Return.
    *)
    (# do 'delete-previous-word' -> CallAction #);
delete_selection:
    (* This action removes all characters in the current selection.
    * The selection can be set with the selection actions.
    *)
    (# do 'delete-selection' -> CallAction #);

(* kill bindings *)

kill_word:
    (* Act exactly like the delete-next-word action, but stuffs the
    * word that was killed into the kill buffer (CUT_BUFFER_1).
    *)
    (# do 'kill-word' -> CallAction #);
backward_kill_word:
    (* Act exactly like the delete-previous-word action, but stuffs the
    * word that was killed into the kill buffer (CUT_BUFFER_1).
    *)
    (# do 'backward-kill-word' -> CallAction #);
kill_selection:
    (* This action deletes the current selection and stuffs the deleted
    * text into the kill buffer (CUT_BUFFER_1).
    *)
    (# do 'kill-selection' -> CallAction #);
kill_to_end_of_line:
    (* This action deletes the entire line to the right of the insert
    * point position, and stuffs the deleted text into the kill buffer
    * (CUT_BUFFER_1).
    *)
    (# do 'kill-to-end-of-line' -> CallAction #);
kill_paragraph:
    (* This action deletes the current paragraph, if between paragraphs
    * it deletes the paragraph above the insert point, and stuffs the
    * deleted text into the kill buffer (CUT_BUFFER_1).
    *)
    (# do 'kill-paragraph' -> CallAction #);
kill_to_end_of_paragraph:
    (* This action deletes everything between the current insert point
    * location and the next paragraph boundary, and stuffs the deleted
    * text into the kill buffer (CUT_BUFFER_1).
    *)
    (# do 'kill-to-end-of-paragraph' -> CallAction #);

(* unkill bindings *)

unkill:
    (* Undo previous kill, i.e., paste the contents of the kill buffer
    * (CUT_BUFFER_1) at the insertion point
    *)
    (# do 'unkill' -> CallAction #);
stuff:
    (* Paste the contents of CUT_BUFFER_0 at the insertion point *)
    (# do 'stuff' -> CallAction #);

```



```

(* new line stuff *)

newline_and_indent:
(* This action inserts a newline into the text and adds spaces to
 * that line to indent it to match the previous line [ This action
 * is still a bit buggy ].
 *)
(# do 'newline-and-indent' -> CallAction #);

newline_and_backup:
(* This action inserts a newline into the text after the insert
 * point.
 *)
(# do 'newline-and-backup' -> CallAction #);

newline:
(* This action inserts a newline into the text before the insert
 * point.
 *)
(# do 'newline' -> CallAction #);

(* Selection stuff *)

select_word:
(* This action selects the word in which the insert point is
 * currently located. If the insert point is between words then it
 * will select the previous word.
 *)
(# do 'select-word' -> CallAction #);

select_all:
(* This action selects the entire text buffer. *)
(# do 'select-all' -> CallAction #);

select_start:
(* This action sets the insert point to the current pointer
 * location. It will then begin a selection at this location. If
 * many of these selection actions occur quickly in succession then
 * the selection count mechanism will be invoked.
 *)
(# do 'select-start' -> CallAction #);

select_adjust:
(* This action allows a selection started with the select-start
 * action to be modified, as described above.
 *)
(# do 'select-adjust' -> CallAction #);

select_end:
(* This action ends a text selection that began with the
 * select-start action, and asserts ownership of the selection or
 * selections specified. A name can be a selection (e.g. PRIMARY)
 * or a cut buffer (e.g. CUT_BUFFER0). Note that case is important.
 * If no names are specified, PRIMARY is asserted.
 *)
(# name: ^text;
enter name[]
do (if name[]=NONE then
    'select-end' -> CallAction
  else
    ('select-end', name[]) -> CallArgAction
  if);
#);

extend_start:
(* This action finds the nearest end of the current selection, and
 * moves it to the current pointer location.
 *)
(# do 'extend-start' -> CallAction #);

extend_adjust:
(* This action allows a selection started with an extend-start
 * action to be modified.
 *)

```



```

    (# do 'extend-adjust' -> CallAction #);
extend_end:
    (* This action ends a text selection that began with the
    * extend-start action, and asserts ownership of the selection or
    * selections specified. A name can be a selection (e.g. PRIMARY)
    * or a cut buffer (e.g. CUT_BUFFER0). Note that case is important.
    * If no names are given, PRIMARY is asserted.
    *)
    (# name: ^text;
    enter name[]
    do (if name[]=NONE then
        'extend-end' -> CallAction
      else
        ('extend-end', name[]) -> CallArgAction
      if);
    #);
insert_selection:
    (* This action retrieves the value of the first (left-most) named
    * selection that exists or the cut buffer that is not empty and
    * inserts it into the Text widget at the current insert point
    * location. A name can be a selection (e.g. PRIMARY) or a cut
    * buffer (e.g. CUT_BUFFER0). Note that case is important.
    *)
    (# name: ^text;
    enter name[]
    do (if name[]=NONE then
        'insert-selection' -> CallAction
      else
        ('insert-selection', name[]) -> CallArgAction
      if);
    #);

    (* Miscellaneous *)

redraw_display:
    (* This action recomputes the location of all the text lines on the
    * display, scrolls the text to vertically center the line
    * containing the insert point on the screen, clears the entire
    * screen, and redisplay it.
    *)
    (# do 'redraw-display' -> CallAction #);
insert_file:
    (* This action activates the insert file popup. The filename
    * option specifies the default filename to put in the filename
    * buffer of the popup. If no filename is specified the the buffer
    * is empty at startup.
    *)
    (# filename: ^text;
    enter filename[]
    do (if filename[]=NONE then
        'insert-file' -> CallAction
      else
        ('insert-file', filename[]) -> CallArgAction
      if);
    #);
search:
    (* This action activates the search popup. The direction must be
    * specified as either forward or backward. The string is optional
    * and is used as an initial value for the Search for: string.
    *)
    (# string: ^text;
    enter string[]
    do (if string[]=NONE then
        'search' -> CallAction
      else
        ('search', string[]) -> CallArgAction
    );

```



```

        if);
    #);
insert_char:
    (* This action may only be attached to a key event. It calls
    * XLookupString to translate the event into a (rebindable) Latin-1
    * character (sequence) and inserts that sequence into the text at
    * the insert point position.
    *)
    (# event: @XKeyEvent;
    enter event
    do (thewidget, 'insert-char', event, 0, 0) -> XtCallActionProc
    #);
insert_string:
    (* This action inserts each string into the text at the insert
    * point location. Any string beginning with the characters "0x"
    * and containing only valid hexadecimal digits in the remainder is
    * interpreted as a hexadecimal constant and the corresponding
    * single character is inserted instead.
    *)
    (# string: ^text;
    enter string[]
    do (if string[]=NONE then
        'insert-string called with no string' -> screen.putline;
        else
        ('insert-string', string[]) -> CallArgAction
        if);
    #);
focus_in:
    (* Currently does nothing. *)
    (# do 'focus-in' -> CallAction #);
focus_out:
    (* Currently does nothing. *)
    (# do 'focus-out' -> CallAction #);
display_caret:
    (* This action allows the insert point to be turned on and off.
    * The state argument specifies the desired state of the insert
    * point. This value may be any of the string values accepted for
    * Boolean resources (e.g. on, True, off, False, etc.). If no
    * arguments are specified, the default value is True. The when
    * argument specifies, for EnterNotify or LeaveNotify events whether
    * or not the focus field in the event is to be examined. If the
    * second argument is not specified, or specified as something other
    * than always then if the action is bound to an EnterNotify or
    * LeaveNotify event, the action will be taken only if the focus
    * field is True. An augmented binding that might be useful is:
    *
    * Text.Translations: #override \ <FocusIn>: display-caret(on)
    * \n\ <FocusOut>: display-caret(off)
    *)
    (# state: ^text;
    enter state[]
    do (if state[]=NONE then
        'display-caret called with no state' -> screen.putline;
        else
        ('display-caret', state[]) -> CallArgAction
        if);
    #);
multiply:
    (* The multiply action allows the user to multiply the effects of
    * many of the text actions. Thus the following action sequence
    * multiply(10) delete-next-word() will delete 10 words. It does
    * not matter whether these actions take place in one event or many
    * events. Using the default translations the key sequence
    * Control-u, Control-d will delete 4 characters. Multiply actions
    * can be chained, thus multiply(5) multiply(5) is the same as
    * multiply(25). If the string reset is passed to the multiply

```



```

    * action the effects of all previous multiplies are removed and a
    * beep is sent to the display.
    *)
    (# value: @integer;
     valuetext: @text;
    enter value
    do value -> valuetext.putint;
     ('multiply', valuetext[]) -> CallArgAction
    #);

form_paragraph:
    (* This action removes all the Carriage Returns from the current
    * paragraph and reinserts them so that each line is as long as
    * possible, while still fitting on the current screen. Lines are
    * broken at word boundaries if at all possible. This action
    * currently works only on Text widgets that use ASCII text.
    *)
    (# do 'form-paragraph' -> CallAction #);

transpose_characters:
    (* This action will swap the position of the character to the left
    * of the insert point with the character to the right of the insert
    * point. The insert point will then be advanced one character.
    *)
    (# do 'transpose-characters' -> CallAction #);

no_op:
    (* The no-op action makes no change to the text widget, and is
    * mainly used to override translations. This action takes one
    * optional argument. If this argument is true then a beep is sent
    * to the display.
    *)
    (# ringbell: @boolean;
    enter ringbell
    do (if not ringbell then
        'no-op' -> CallAction
      else
        ('no-op', 'RingBell') -> CallArgAction
      if);
    #);

-- XtObjectLib: attributes --

(* Gennerally applicable interface for calling actions directly *)

CallAction:
    (* Call action with no params *)
    (# action: ^text
    enter action[]
    do (thewidget, action, 0, 0, 0) -> XtCallActionProc
    #);

CallArgAction:
    (* Call action with one argument *)
    (# action: ^text;
     arg: ^text;
     params: @StringArray(# initialsize::< (# do 1->value #)#);
    enter (action[], arg[])
    do params.init;
     arg[] -> params.addText;
     (thewidget, action, 0, params, params.number) -> XtCallActionProc;
     params.clear;
    #);

CallArgsAction:
    (* Call action with several params *)
    (# action: ^text;
     args: ^StringArray;
    enter (action[], args[])

```



```
do (thewidget, action, 0, args, args.number) -> XtCallActionProc;  
#)
```

---

Textactions Interface

[Mjølner Informatics](#)



# Editres Interface

```
ORIGIN '../xtenv';
BODY 'private/editresbody';
(* enableEditres invoked on any shell widget (e.g. TopLevel) will allow
 * the shell and children to respond to the editres protocol,
 * which in turn allows for the use of the editres program.
 * From the editres man page:
 *
 * Editres is a tool that allows users and application developers
 * to view the full widget hierarchy of any X Toolkit client that
 * speaks the Editres protocol. In addition editres will help the
 * user construct resource specifications, allow the user to apply
 * the resource to the application and view the results dynamically.
 * Once the user is happy with a resource specification editres will
 * append the resource string to the user's X Resources file.
 *
 *
 * BETA interface contributed by
 *   Holger Tuerk <htuerk00@marvin.informatik.uni-dortmund.de>
 *)
-- shellLib: Attributes --
enableEditres: (# ... #)
```

---

Editres Interface

[Mjølner Informatics](#)



# Xterm Interface

```
ORIGIN '~beta/basiclib/systemenv';
BODY 'private/xtermbody';
--systemlib: attributes --
```

## **startCommandInXterm:**

```
(* Starts a command in an xterm window. Input to the command
* may be given via the window, and all output from the command
* is shown in the window.
* When the command is finished, the text 'Type Control-c to
* close window' is printed in the xterm window.
* The xterm program is searched for in standard places, and if
* not found, the XTERM environment variable is assumed to contain
* the full path of the xterm program.
*)
(# command: ^text (* full path of executable *);
args: ^text (* arguments to executable *);
title: ^text
    (* What to write in the xterm title bar; if NONE, the name and
    * arguments of the command is used
    *);
verbose: @boolean (* Print trace on screen *);
enter (command[], args[], title[], verbose)
do ...
#);
```

## **setXtermTitle:**

```
(* Print out escape sequence that sets the xterm title bar
* to titletext.
* Assumes that the program is run from an xterm shell.
*)
(# titletext: ^text
enter titletext[]
...
#);
```

## **setXtermIconName:**

```
(* Print out escape sequence that sets the xterm icon name
* to icontext.
* Assumes that the program is run from an xterm shell.
*)
(# icontext: ^text
enter icontext[]
...
#)
```

---