Widget Reference (G–P) 8

Gauge Widget

Class

Class Name: `Gauge`
Class Pointer: `gaugeWidgetClass`

Ancestry

Core-Primitive-Gauge

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/Gauge.h>
```

Description

The Gauge widget displays a numeric value graphically. It is similar to a Slider except that it is read-only.

Components

The Gauge widget consists of the following components:

- Bar with shaded region indicating the current value (oriented either horizontally or vertically)
- Ticks (optional)
- Minimum value (optional)
- Maximum value (optional)
- Current value (optional—must be created and managed by the application)

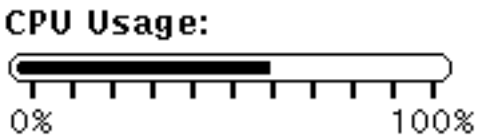

Figure 8-1 Gauge Horizontal Orientation

Coloration

For 3D and 2D, the area surrounding the Gauge and its labels is drawn with the parent's `XtNbackground`. `XtNfontColor` is used to draw the minimum and maximum value labels.

For 3D, the gauge component and tickmark coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1. `XtNforeground` is used to draw the current-value indicator bar inside the Gauge.

For 2D, `XtNbackground` and `XtNforeground` are used to render the gauge component and the tickmarks, as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

Application Notification

The application is responsible for setting the current value of the Gauge as well as creating a widget (such as a StaticText or TextLine) to display the current value numerically.

Resources

Table 8-1 Gauge Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-2 Gauge Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	SGI
XtNreferenceName	String	NULL	GI

Table 8-2 Gauge Primitive Resources (Continued)

Name	Type	Default	Access
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 8-3 Gauge Resources

Name	Type	Default	Access
XtNleftMargin	Dimension	OL_IGNORE	SGI
XtNmaxLabel	OlStr	NULL	SGI
XtNminLabel	OlStr	NULL	SGI
XtNorientation	OlDefine	OL_VERTICAL	GI
XtNrecomputeSize	Boolean	FALSE	SGI
XtNrightMargin	Dimension	OL_IGNORE	SGI
XtNsliderMax	int	100	SGI
XtNsliderMin	int	0	SGI
XtNsliderValue	int	0	SGI
XtNspan	Dimension	OL_IGNORE	SGI
XtNticks	int	0	SGI
XtNtickUnit	OlDefine	OL_NONE	SGI

XtNleftMargin

Class	Type	Default	Access
XtCMargin	Dimension	OL_IGNORE	SGI

Synopsis: The number of pixels in the margin to the left of the gauge.
Values: `OL_IGNORE/"ignore"` or any valid `Dimension`.

XtNmaxLabel

Class	Type	Default	Access
XtCLabel	OlStr	NULL	SGI

Synopsis: The label to be placed next to the maximum value position.

Values: Any `OlStr` value valid in the current locale.

For a vertical gauge, the label is placed to the right of the maximum value position. If there is not enough space for the entire label and `XtNrecomputeSize` is TRUE, then the widget will request more space to show the entire label.

For a horizontal gauge, the label is placed centered and below the maximum value position. If there is not enough room to center the label and `XtNrecomputeSize` is set to FALSE, the end of the label will be aligned with the right end of the outline of the gauge. If this label collides with the minimum label, some part of the labels will overlap. If there is not enough room to center the label and `XtNrecomputeSize` is set to TRUE, then the widget will request more space to center the label below the maximum value position.

XtNminLabel

Class	Type	Default	Access
XtCLabel	OlStr	NULL	SGI

Synopsis: `The` label to be placed next to the minimum value position.
Values: Any `OlStr` value valid in the current locale.

For a vertical gauge, the label is placed to the right of the minimum value position. If there is not enough space for the entire label and `XtNrecomputeSize` is TRUE, then the widget will request more space to show the entire label.

For a horizontal gauge, the label is placed centered and below the minimum value position. If there is not enough room to center the label and `XtNrecomputeSize` is set to FALSE, the beginning of the label will be aligned with the left end of the outline of the gauge and is drawn to the right. If this label collides with the maximum label, some part of the labels will overlap. If there is not enough room to center the label and `XtNrecomputeSize` is set to TRUE, the widget will request for more space to center the label below the minimum value position.

XtNorientation

Class	Type	Default	Access
XtCOrientation	OlDefine	OL_VERTICAL	GI

Synopsis: The direction for the visual presentation of the widget.

Values: `OL_HORIZONTAL/"horizontal"` – Define a horizontal gauge.
`OL_VERTICAL/"vertical"` – Define a vertical gauge.

XtNrecomputeSize

Class	Type	Default	Access
XtCRecomputeSize	Boolean	FALSE	SGI

Synopsis: The widget's resize policy.
Values: `TRUE/"true"` – The widget will resize itself whenever needed, to compensate for the space needed to show the tick marks and the labels.
`FALSE/"false"` – The widget will not resize itself.

The gauge widget uses `XtNspan`, the sizes of the labels, and `XtNtickUnit` to determine the preferred size.

XtNrightMargin

Class	Type	Default	Access
XtCMargin	Dimension	OL_IGNORE	SGI

Synopsis: The number of pixels in the margin to the right of the gauge.
Values: `OL_IGNORE/"ignore"` or any valid `Dimension`.

XtNsliderMax/ XtNsliderMin

Class	Type	Default	Access
XtCSliderMax	int	100	SGI
XtCSliderMin	int	0	SGI

Synopsis: The range of values tracked by the Gauge widget.
Values: `XtNsliderMin` < `XtNsliderMax`

XtNsliderValue

Class	Type	Default	Access
XtCSliderValue	int	0	SGI

Synopsis: The values represented by the current position of the end of the shaded portion of the gauge.
Values: `XtNsliderMin` ≤ `XtNSliderValue` ≤ `XtNsliderMax`

XtNspan

Class	Type	Default	Access
XtCSpan	Dimension	OL_IGNORE	SGI

Synopsis: If `XtNrecomputeSize` is set to TRUE, the preferred length of the gauge, not counting the space needed for the labels.
Values: `OL_IGNORE/"ignore"` or any valid `Dimension`.

The gauge widget uses the span value, the sizes of the labels, and `XtNtickUnit` to determine the preferred size.

XtNticks

Class	Type	Default	Access
XtCTicks	int	0	SGI

Synopsis: The interval between tick marks.
Values: The unit of the interval value is determined by `XtNtickUnit`.

XtNtickUnit

Class	Type	Default	Access
XtCTickUnit	OlDefine	OL_NONE	SGI

Synopsis: The interpretation of the `XtNticks` resource.
Values: `OL_NONE/"none"` – Display no tick marks and ignore `XtNticks`.
`OL_PERCENT/"percent"` – Interpret `XtNticks` as the percentage of the gauge value range.
`OL_SLIDERVALUE/"slidervalue"` – Interpret `XtNticks` in gauge value units.

To be consistent with the Scrollbar widget, the effective spacing between tick marks, designated in `XtNticks` and `XtNtickUnit`, should be less than or equal to the spacing in `XtNgranularity`.

Activation Types

The following table lists the activation types used by the Gauge.

Table 8-4 Gauge Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey

Table 8-4 Gauge Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

The Gauge widget is a read-only control that has no activation types besides the ones in "Common Activation Types" on page 68.

See Also

"Slider Widget" on page 586.

Gauge Function

The following convenience function is used to set gauge values.

OlSetGaugeValue

```
#include <Xol/Gauge.h>

extern void OlSetGaugeValue(
    Widget      w,
    int         value);
```

This function is an alternative and faster method of setting the current value of a Gauge widget. The effect is equivalent to doing `XtSetValues()` on the `XtNsliderValue` resource of the widget.

MenuButton Widget

Class

Class Name: `MenuButton`
Class Pointer: `menuButtonWidgetClass,menuButtonGadgetClass`

Ancestry

Core-Primitive-Button-MenuButton

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/MenuButton.h>
```

Description

The MenuButton is used to create a popup menu. It appears similar to the OblongButton widget (seeOblongButton Widget on page 464), except that it also has a mark, called a menumark, near the right end of the button. When the user invokes the MENU command on the MenuButton, a menu pops up. The MenuButton provides the features of menu default selection and menu previewing as well as the features of the MenuShell widget.

Components

The MenuButton consists of an oblong border containing a label and a menumark. A popup menu is attached. Each MenuButton also has the components of the MenuShell widget.

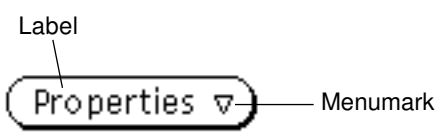

Figure 8-2 MenuButton Appearance

Subwidgets

The MenuButton widget automatically creates and attaches a MenuShell widget. An application can add menu items to this menu by obtaining the value of the `XtNmenuPane` resource and adding children to this widget.

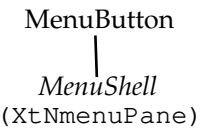

Figure 8-3 MenuButton Subwidget

Popping Up the MenuShell Subwidget

When the MenuButton widget is not in a menu, pressing or clicking MENU on the MenuButton pops up the MenuButton's menu in the direction of the menumark. In this case the menumark points down; therefore, the menu will pop up below the MenuButton.

When the MenuButton widget is in a *stay-up* menu (implementing a cascade menu), pressing or clicking MENU when the pointer is within or on the button's border pops up the button's menu in the direction of the menumark.

When the MenuButton widget is in a *popup* menu (implementing a cascade menu), moving the pointer into the menumark region pops up the menu in the direction of the menumark. The position is computed when the movement into the menumark region is first detected, but rapid pointer motion and internal delays in popping up the menu may let the pointer wander.

Moving the pointer out of the MenuButton widget, but not directly into the newly popped up menu, causes that menu to be popped down. This occurs even if the pointer is moved into and out of the newly popped up menu in the interim.

Menu Previewing

The MenuButton widget supports previewing of the default menu item if the toolkit resource `XtNselectDoesPreview` is set to TRUE. In this case, if SELECT is activated on the MenuButton, the widget will display the label of the default item within the border of the MenuButton while the SELECT mouse

button is down. If the user releases the SELECT mouse button inside the MenuButton border, the default menu item will be selected.

Menu Placement—Not Enough Space

If the right or bottom edge of the screen is too close to allow the menu placement described above, the menu pops up aligned with the edge of the screen and the pointer is shifted horizontally to keep it 4 points from the left edge of the menu items. If the left edge of the screen is too close, the menu pops up 4 points from the edge and the pointer is shifted to lie on the edge. The pointer does not jump back after the menu is dismissed.

Selecting the Default Item

Each MenuButton widget has a default item belonging to the MenuShell subwidget. If SELECT is activated on the MenuButton (and the `XtNselectDoesPreview` toolkit resource is set to TRUE) and the default item is inactive (the `XtNsensitive` resource is FALSE) or busy (the `XtNbusy` resource is TRUE), the system beeps.

If SELECT is activated on the MenuButton (and the `XtNselectDoesPreview` toolkit resource is set to TRUE) and the default item is another MenuButton (for a cascading menu), its default will be selected; this recurses through the menu tree until a non-MenuButton default widget is found.

Coloration

For 3D, MenuButton coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1. `XtNfontColor` is used to draw the label.

For 2D, `XtNbackground` and `XtNfontColor` are used to render the MenuButton as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

If the toolkit resource `XtNmouseless` is set to TRUE and the toolkit resource `XtNinputFocusFeedback` is set to OL_INPUT_FOCUS_COLOR, then the background of the MenuButton will be drawn with the value of

XtNinputFocusColor when the widget receives input focus. However, if XtNinputFocusColor is the same as XtNbackground, then the MenuButton inverts XtNfontColor and XtNbackground. Once the input focus leaves the widget, the original coloration is restored.

Label Appearance

The XtNwidth, XtNheight, XtNrecomputeSize, and XtNlabelJustify resources interact to produce a truncated, clipped, centered, or left-justified label as shown in the following table.

Table 8-5 MenuButton Label Appearance

XtNwidth	XtNrecomputeSize	XtNlabelJustify	Result
any value	TRUE	any	Just Fits
> needed for label	FALSE	OL_LEFT	Left Justified
> needed for label	FALSE	OL_CENTER	Centered
< needed for label	FALSE	any	Trunc
XtNheight	**XtNrecomputeSize**	**XtNlabelJustify**	**Result**
any value	TRUE	any	Just Fits
> needed for label	FALSE	any	Centered
< needed for label	FALSE	any	Clipped

When the label is centered or left-justified, the extra space is filled with the background color of the MenuButton widget, as determined by the XtNbackground and XtNbackgroundPixmap resources. When a text label is truncated, the truncation occurs at a character boundary and a “more arrow” is inserted to show that part of the label is missing. The arrow requires that more of the label be truncated than would otherwise be necessary. If the width of the button is too small to show even one character with the triangle, only the triangle is shown. If the width is so small that the entire triangle cannot be shown, the arrow is clipped on the right.

Resources

Table 8-6 MenuButton Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	(calculated)	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	(calculated)	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-7 MenuButton Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	n/a
XtNacceleratorText	String	NULL	n/a
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	n/a
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI

Table 8-7 MenuButton Primitive Resources (Continued)

Name	Type	Default	Access
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 8-8 MenuButton Resources

Name	Type	Default	Access
XtNdefault	Boolean	FALSE	SGI
XtNlabel	OlStr	(instance name)	SGI
XtNlabelImage	XImage *	NULL	SGI
XtNlabelJustify	OlDefine	OL_LEFT	SGI
XtNlabelType	OlDefine	OL_STRING	SGI
XtNmenuMark	OlDefine	(calculated)	SGI
XtNrecomputeSize	Boolean	TRUE	SGI

The following table lists the MenuButton resources that are propagated to the MenuShell subwidget.

Table 8-9 MenuShell Subwidget Resources[1]

Name	Type	Default	Access
XtNcenter	Boolean	TRUE	I
XtNhPad	Dimension	6	I
XtNhSpace	Dimension	6	I
XtNlayoutType	OlDefine	OL_FIXEDCOLS	I
XtNmeasure	int	1	I
XtNmenuPane	Widget	(special)	G
XtNpushpin	OlDefine	OL_NONE	I
XtNpushpinDefault	Boolean	FALSE	I
XtNsameSize	OlDefine	OL_COLUMNS	I
XtNshellTitle	OlStr	(widget name)	SGI
XtNvPad	Dimension	3	I
XtNvSpace	Dimension	1	I

1. These subwidget resources are described in the sections "ControlArea Widget" on page 249 and "MenuShell Widget" on page 414.

XtNdefault

Class	Type	Default	Access
XtCDefault	Boolean	FALSE	SGI

Synopsis: Whether the MenuButton is the default choice in its immediate shell.

Values: `TRUE/"true"` – If the button is in a menu, an oval ring is drawn around the button to show that the button is the default choice of one or more buttons.
`FALSE/"false"` – This button is not the default control of the shell.

Setting `XtNdefault` to TRUE has the effect of setting `XtNdefault` of the previous default control for the shell to be FALSE. The OL_MENUDEFAULT and OL_MENUDEFAULTKEY activation types modify this resource.

XtNlabel

Class	Type	Default	Access
XtCLabel	OlStr	(instance name)	SGI

Synopsis: The text for the Label.

Values: Any `OlStr` value valid in the current locale.

This resource is ignored if the `XtNlabelType` resource has the value OL_IMAGE.

The MenuButton label is colored using the `XtNfontColor` resource.

XtNlabelImage

Class	Type	Default	Access
XtCLabelImage	XImage *	NULL	SGI

Synopsis: The image for the Label.

This resource is ignored unless the `XtNlabelType` resource has the value OL_IMAGE. If the image is of type `XYBitmap`, the image is highlighted when appropriate by reversing the 0 and 1 values of each pixel (that is, by XORing the image data). If the image is of type `XYPixmap` or `ZPixmap`, the image is not highlighted, although the space around the image inside the border is highlighted.

If the image is smaller than the space available for it inside the border and `XtNlabelTile` is FALSE, the image is centered vertically and either centered or left-justified horizontally, depending on the value of the `XtNlabelJustify` resource. If the image is larger than the space available for it, it is clipped so that it does not display outside the border. If the `XtNdefault` resource is TRUE so that the border is doubled, the space available is that inside the inner line of the border.

XtNlabelJustify

Class	Type	Default	Access
XtCLabelJustify	OlDefine	OL_LEFT	SGI

Synopsis: The justification of a label within the widget.
Values: `OL_LEFT/"left"` – Left-justify the label.
`OL_CENTER/"center"` – Center the label.

XtNlabelType

Class	Type	Default	Access
XtCLabelType	OlDefine	OL_STRING	SGI

Synopsis: The form that the label takes.
Values: `OL_STRING/"string"` – The label is text.
`OL_IMAGE/"image"` – The label is an image.

XtNmenuMark

Class	Type	Default	Access
XtCMenuMark	OlDefine	(calculated)	SGI

Synopsis: The direction of the menu arrow.
Values: `OL_DOWN/"down"` – The menu arrow points down.
`OL_RIGHT/"right"` – The menu arrow points to the right.

The default is OL_RIGHT if the immediate shell ancestor is a MenuShell or a subclass thereof; otherwise, it is OL_DOWN.

XtNmenuPane

Class	Type	Default	Access
XtCMenuPane	Widget	NULL	G

Synopsis: The widget where menu items can be added.
Values: ID of the menupane widget contained in the MenuButton's MenuShell.

This resource is available once the MenuButton widget has been created.

XtNrecomputeSize

Class	Type	Default	Access
XtCRecomputeSize	Boolean	TRUE	SGI

Synopsis: The resize policy of the widget.
Values: `TRUE/"true"` – The MenuButton widget will do normal size calculations that may cause its geometry to change and automatically set the `XtNheight` and `XtNwidth` resources. `FALSE/"false"` – The MenuButton widget will leave its size alone; this may cause truncation of the visible image being shown by the MenuButton widget if the fixed size is too small, or may cause padding if the fixed size is too large. The location of the padding is determined by the `XtNlabelJustify` resource.

Activation Types

The following table lists the activation types used by the MenuButton.

Table 8-10 MenuButton Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MENU	MENU	XtNmenuBtn
OL_MENUDEFAULT	MENUDEFAULT	XtNmenuDefaultBtn
OL_MENUDEFAULTKEY	MENUDEFAULT	XtNmenuDefaultKey
OL_MENUKEY	MENU	XtNmenuKey
OL_MOVEDOWN	DOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey

Table 8-10 MenuButton Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	UP	XtNupKey
OL_MULTIDOWN	JUMP DOWN	XtNmultiDownKey
OL_MULTILEFT	JUMP LEFT	XtNmultiLeftKey
OL_MULTIRIGHT	JUMP RIGHT	XtNmultiRightKey
OL_MULTIUP	JUMP UP	XtNmultiUpKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_NEXTWINDOW	NEXTWINDOW	XtNnextWinKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_PREVWINDOW	PREVWINDOW	XtNprevWinKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

Activation types not described in the following list are described in "Common Activation Types" on page 68.

OL_MENU

The OL_MENU activation type can be used to pop up the menu in two different modes: press-drag-release and click-move-click. These modes are described in the *OPEN LOOK GUI Functional Specification* section "Using Menu Buttons" in Chapter 15. The position of the menu depends on the space available on the screen and is described in the *OPEN LOOK GUI Functional Specification* section "Menu Placement" in Chapter 15.

OL_MENUDEFAULT/ OL_MENUDEFAULTKEY

The OL_MENUDEFAULT and OL_MENUDEFAULTKEY activation types apply only to MenuButtons that are descendants of a Menu. These activation types will set the MenuButton `XtNdefault` resource to TRUE, and change the display of the widget according to the *OPEN LOOK GUI Functional Specification* section "Changing Menu Defaults" in Chapter 15.

OL_MENUKEY

The OL_MENUKEY activation type can be used to pop up the menu according to the *OPEN LOOK Mouseless Specification* section 4.2.

OL_SELECT

The activation of the MenuButton widget with the SELECT button depends on the value of the toolkit resource `XtNselectDoesPreview`. When the resource `XtNselectDoesPreview` is FALSE, this activation type will behave exactly as the OL_MENU activation type. When `XtNselectDoesPreview` is TRUE, SELECT can be used as a shortcut to display and activate the menu default as described in the *OPEN LOOK GUI Functional Specification* Chapter 15 and Chapter 5. If SELECT is released within the MenuButton, the default menu item will be activated with the OL_SELECTKEY activation type.

OL_SELECTKEY

The OL_SELECTKEY activation type can be used to pop up the menu according to the *OPEN LOOK Mouseless Specification* section 4.2.

See Also

"AbbrevMenuButton Widget" on page 217,
"ControlArea Widget" on page 249
"MenuShell Widget" on page 414.

8

MenuShell Widget

MenuShell Widget

Class

Class Name: `MenuShell`
Class Pointer: `menuShellWidgetClass`

Ancestry

Core-Composite-Shell-WMShell-VendorShell-TransientShell-MenuShell

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/Menu.h>
```

Description

The MenuShell widget is used to create a menu not associated with either a MenuButton or an AbbrevMenuButton. For example, a MenuShell widget can be attached to a button, such as an OblongButton widget, but this does not make the button into a menu button. However, all the features of the MenuShell widget (except those related to menu creation) also pertain to the MenuButton menu.

Components

A menu contains a set of items that are presented to the user for selection. These are specified by the application as widgets attached to the menu. One of these items is a default item. (A menu always has exactly one default item.) The items are laid out in a control area. A menu also has a title, a title separator, a border or window border, a drop shadow, and an optional pushpin. The application chooses the label for the Title and whether a menu has a pushpin.

A popup or stay-up menu shows the title, border, pushpin (if available), items, and drop shadow. The title is left out if the menu is from either a menu button or an abbreviated menu button. A pinned menu shows the window border, title, pushpin, items, but no drop shadow.

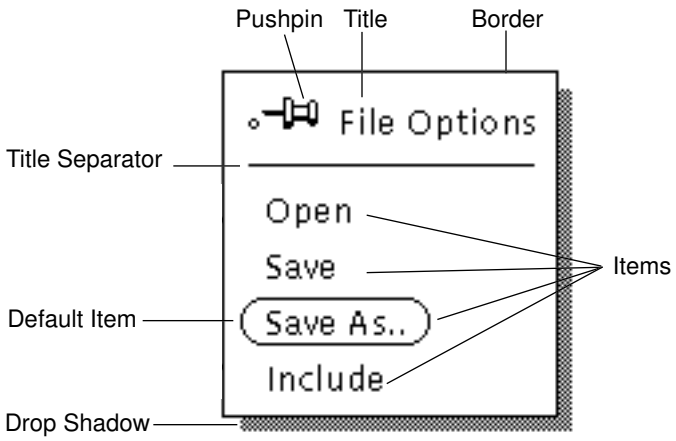

Figure 8-4 MenuShell Menu Components

Subwidgets

The MenuShell contains one subwidget, a “menupane,” which is a container for the components of the menu. This widget is provided automatically and is accessible through the `XtNmenuPane` resource.

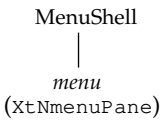

Figure 8-5 MenuShell Subwidgets

Subclass of Shell Widget

The MenuShell widget is a subclass of Shell widget, so the `XtCreatePopupShell()` routine is used to create it instead of the more common `XtCreateWidget()`.

Menupane

The menupane is not described as a separate widget here; the only interface to it for the application programmer is as a parent widget to which the widgets comprising the menu items are attached. The menu items are not attached directly to the MenuShell widget, since a shell widget can take only one child.

Associating a Menu with a Widget

A menu can be associated with any widget, including Primitive widgets. The connection is made by creating the menu widget as a child of the other widget. Being a shell widget, the MenuShell widget is not a normal widget-child of its parent, but a popup child. If the application allows it, the menu augments the parent's event list so that the popping up of the menu is handled automatically.

Popup Control

Pressing MENU when the pointer is over the parent of the MenuShell widget causes the menu to be popped up. The menu is presented as a popup menu, where the items are available for a *press-drag-release* type of selection (see below).

Clicking MENU when the pointer is over the parent of the MenuShell widget also causes the menu to be popped up, but the menu is presented as a stay-up menu, where the items are available for a *click-move-click* type of selection, instead (see below).

A "slow click" (a press with a delay before the release) may show the menu as a popup on the press, then as a stay-up on the release.

Selection Control

The MenuShell arranges for its children to respond to either the press-drag-release or the click-move-click type of selection.

With the press-drag-release type of control, the user can keep MENU pressed and move the pointer to the item of choice; releasing MENU selects the item and pops the menu down. If the pointer is not over an item when MENU is released, the menu simply pops down.

With the click-move-click type of control, the user can move the pointer to the item of choice (MENU has already been released to end a click); clicking SELECT or MENU selects the item and pops the menu down. If the pointer is not over an item when SELECT or MENU is clicked, the menu simply pops down.

These selection methods apply to all menu items *except* menu buttons. For example, in Figure 8-4 on page 415, `Locate Owner` can be selected using the methods described here. For the other items in the figure (which are menu

buttons), see "MenuButton Widget" on page 403 for the explanation of menu button selection behavior.

Converting Stay-up to Popup Menu

Pressing MENU in a stay-up menu converts it to a popup menu. Thus, the click-move-click selection control becomes a press-drag-release selection control.

Highlighting Items

In the press-drag-release type of selection control, each menu item highlights while the pointer is over it. The form of the highlighting depends on the type of widget making up the item. Again, the MenuShell widget arranges for its children to respond in this way. No highlighting occurs when the click-move-click type of selection control is used.

Use of the Pushpin

The pushpin is presented to the user like any of the items to be selected from the menu, except that it is always the topmost item, and it is presented visually as an "adornment" of the header, next to the title (if present). The user can select the pushpin, pushing it in to cause the menu to remain on the display as a popup window, or a *pinned menu*, and pulling it out to make the menu a stay-up menu. To the user, a pinned menu behaves indistinguishably from a command window.

Default Item

If none of the menu items is explicitly set as the default item, the menu picks the first menu item to be the default item. If the menu contains a pushpin and no other menu item is explicitly set as the default item, the pushpin is chosen as the default item.

Popup Position

If the menu is not from a menu button, the menu pops up so that the default item is vertically centered 4 points to the right of the pointer. If the right or bottom edge of the screen is too close to allow this placement, the menu pops up with its edge aligned to the edge of the screen, and the mouse pointer is shifted horizontally to keep it 4 points from the left edge of the default item.

For the popup position when the menu is from a menu button, see "MenuButton Widget" on page 403.

Coloration

The following diagram illustrates the resources that affect MenuShell coloration in 2D.

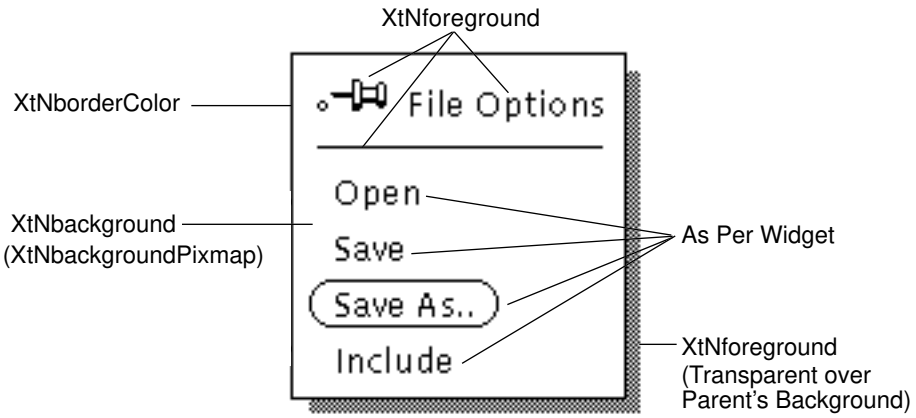

Figure 8-6 MenuShell Menu Coloration

Coloration is identical for 3D, except that the coloration of the pushpin is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1 of the pushpin, and the BG2 (shaded), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1.

Programmatic Menu Popup and Popdown

Four convenience routines are provided to programmatically control the mapping and unmapping of menus.

OlMenuPopup

```
void OlMenuPopup(
    Widget                menu,
    Widget                emanate,
    Cardinal              item_index,
    OlDefine              state,
    Boolean               set_position,
    Position              x,
    Position              y,
    OlMenuPositionProc    position_proc);
```

menu A `menuShellWidget` ID obtained by creating a menu explicitly.

emanate The object that the menu is currently associated with; it is supplied to the *position_proc* when the menu positioning is done. If this field is NULL, the menu's parent object is used as the emanate object for later positioning calculations.

item_index If *emanate* is a flattened widget, this parameter specifies the particular item.

state The state the menu should be in when it is visible on the screen: one of: OL_PINNED_MENU, OL_PRESS_DRAG_MENU, or OL_STAYUP_MENU.

set_position A flag indicating whether the following two arguments (*x* and *y*) are used to help position the menu. If the flag is FALSE, the current Pointer Location is used to initialize *x* and *y*.

x, y These coordinates are used by the positioning routine. Typically, these values represent the pointer position with respect to the RootWindow (e.g., `xevent-> xbutton.x_root` and `xevent->xbutton.y_root`). However, if the menu's state is OL_PINNED_MENU, these coordinates represent the desired upper-left hand corner of the pinned menu.

position_proc The procedure called to determine the menu's position if the menu's state is either OL_PINNED_MENU or OL_STAYUP_MENU. If the menu's state is OL_PINNED_MENU, the *position_proc* value is ignored. If this procedure is NULL, the default positioning routine (i.e., the one associated with the emanate widget or the menu's parent) is used. The type of this procedure is:

```
typedef void (*OlMenuPositionProc)(
      Widget        menu,
      Widget        emanate,
      Cardinal      item_index,
      OlDefine      state,
      Position      *mx,
      Position      *my,
      Position      *px,
      Position      *px);
```

menu A `menuShellWidget` ID obtained by creating a menu explicitly.

emanate Menu's emanate widget.

item_index Emanate *item_index* or OL_NO_ITEM

state The state of the menu; either OL_PRESS_DRAG_MENU or OL_STAYUP_MENU

mx, my	Pointers containing the menu's current x and y locations. If the position routine wants to move the menu, it should change these values. The position routine should not move the menu explicitly.
px, py	The x and y locations supplied to the `OlMenuPopup()` routine. If the position routine changes these values, the pointer is warped to the new location.

OlMenuPost

```
void OlMenuPost(
    Widget      menu);
```

This convenience routine is equivalent to:

```
OlMenuPopup(menu, NULL, OL_NO_ITEM, OL_PRESS_DRAG_MENU, FALSE,
            0, 0, (OlMenuPositionProc) NULL);
```

OlMenuPopdown

```
void OlMenuPopdown(
    Widget      menu,
    Boolean     dismiss_pinned);
```

This routine pops down a menu. If a menu is pinned, a value of TRUE for *dismiss_pinned* is required to dismiss it. If a menu does not have a pushpin or the menu is not pinned, the *dismiss_pinned* field is ignored.

OlMenuUnpost

```
void OlMenuUnpost(
    Widget      menu);
```

This convenience routine is equivalent to:

```
OlMenuPopdown(menu, FALSE);
```

Avoiding Permanent Toolkit Grabs

To avoid permanent toolkit grabs, the application should add the `OlCallbackPopdownMenu` callback procedure to the `XtNpopupCallback` callback list of any shell (except NoticeShell and MenuShell) that may be popped up with an exclusive toolkit grab by the application.

```
void OlCallbackPopdownMenu(
    Widget          w,
    XtPointer       client_data,
    XtPointer       call_data)
```

The arguments are:

w	Specifies the widget
client_data	Not used by this procedure
call_data	Not used by this procedure

Keyboard Traversal

By default, all menus allow traversal among the traversable controls added to the widget.

Popping up a Menu via the keyboard is done by traversing to a MenuButton, using NEXTFIELD, PREVFIELD, MOVEUP, MOVEDOWN, MOVERIGHT, or LEFT, and pressing the MENUKEY key. If a menu is attached to a control other than a MenuButton, it can be popped up by traversing to that control and pressing the MENUKEY.

Keyboard traversal within a menu is done using the PREVFIELD, NEXTFIELD, MOVEUP, MOVEDOWN, MOVELEFT and MOVERIGHT keys. The PREVFIELD, MOVEUP, and MOVELEFT keys move the input focus to the previous Menu item with keyboard traversal enabled. If the input focus is on the first item of the Menu, then pressing one of these keys will wrap to the last item of the Menu with keyboard traversal enabled. The NEXTFIELD, MOVEDOWN, and MOVERIGHT keys move the input focus to the next Menu item with keyboard traversal enabled. If the input focus is on the last item of the Menu, then pressing one of these keys will wrap to the first item of the Menu with keyboard traversal enabled.

To traverse out of the menu, the following keys can be used:

- CANCEL dismisses the menu and returns focus to the originating control
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application

NEXTWINDOW, PREVWINDOW, NEXTAPP, and PREVAPP are provided by the OLWM window manager and are set via different resources.

8

MenuShell Widget Resources

Table 8-11 MenuShell Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	GI
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-12 MenuShell Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 8-13 MenuShell Shell Resources

Name	Type	Default	Access
XtNallowShellResize	Boolean	TRUE	SGI
XtNcreatePopupChildProc	XtCreatePopupChildProc	NULL	SGI

Table 8-13 MenuShell Shell Resources (Continued)

Name	Type	Default	Access
XtNgeometry	String	NULL	GI
XtNoverrideRedirect	Boolean	FALSE	SGI
XtNpopdownCallback	XtCallbackList	NULL	SGIO
XtNpopupCallback	XtCallbackList	NULL	SGIO
XtNsaveUnder	Boolean	TRUE	SGI
XtNvisual	Visual *	(parent's)	GIO

Table 8-14 MenuShell WMShell Resources

Name	Type	Default	Access
XtNbaseHeight	int	XtUnspecifiedShellInt	SGI
XtNbaseWidth	int	XtUnspecifiedShellInt	SGI
XtNheightInc	int	XtUnspecifiedShellInt	SGI
XtNiconMask	Pixmap	NULL	SGI
XtNiconPixmap	Pixmap	NULL	SGI
XtNiconWindow	Window	NULL	SGI
XtNiconX	int	XtUnspecifiedShellInt	SGI
XtNiconY	int	XtUnspecifiedShellInt	SGI
XtNinitialState	InitialState	NormalState	SGI
XtNinput	Bool	FALSE	G
XtNmaxAspectX	int	XtUnspecifiedShellInt	SGI
XtNmaxAspectY	int	XtUnspecifiedShellInt	SGI
XtNmaxHeight	int	OL_IGNORE	SGI
XtNmaxWidth	int	OL_IGNORE	SGI
XtNminAspectX	int	XtUnspecifiedShellInt	SGI
XtNminAspectY	int	XtUnspecifiedShellInt	SGI
XtNminHeight	int	OL_IGNORE	SGI
XtNminWidth	int	OL_IGNORE	SGI
XtNtitle	String	NULL	SGI
XtNtitleEncoding	Atom	XA_STRING	SGI
XtNtransient	Boolean	TRUE	SGI
XtNwaitForWm	Boolean	TRUE	SGI
XtNwidthInc	int	XtUnspecifiedShellInt	SGI
XtNwindowGroup	Window	XtUnspecifiedWindow	SGI

Table 8-14 MenuShell WMShell Resources (Continued)

Name	Type	Default	Access
XtNwinGravity	int	XtUnspecifiedShellInt	SGI
XtNwmTimeout	int	5000 (msec)	SGI

Table 8-15 MenuShell VendorShell Resources

Name	Type	Default	Access
XtNbusy	Boolean	FALSE	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNdefaultImName	String	NULL	SGI
XtNfooterPresent	Boolean	FALSE	SGI
XtNfocusWidget	Widget	(see description)	SGI
XtNimFontSet	OlFont	XtDefaultFontSet	SGI
XtNimStatusStyle	OlImStatusStyle	OL_NO_STATUS	GI
XtNleftFooterString	OlStr	NULL	SGI
XtNleftFooterVisible	Boolean	TRUE	SGI
XtNmenuButton	Boolean	(see description)	GI
XtNmenuType	OlDefine	(see description)	SGI
XtNpushpin	OlDefine	(see description)	SGI
XtNresizeCorners	Boolean	(see description)	SGI
XtNrightFooterString	OlStr	NULL	SGI
XtNrightFooterVisible	Boolean	TRUE	SGI
XtNshellTitle	OlStr	NULL	SGI
XtNuserData	XtPointer	NULL	SGI
XtNwindowHeader	Boolean	(see description)	GI
XtNwmProtocol	XtCallbackList	NULL	SGIO
XtNwmProtocolInterested	int	OL_WM_DELETE_WINDOW | OL_WM_TAKE_FOCUS	I

Table 8-16 MenuShell TransientShell Resources

Name	Type	Default	Access
XtNtransientFor	Widget	NULL	SGI

Table 8-17 MenuShell Resources

Name	Type	Default	Access
XtNmenuAugment	Boolean	TRUE	GI
XtNmenuPane	Widget	NULL	G
XtNpushpin	OlDefine	OL_NONE	GI
XtNpushpinDefault	Boolean	FALSE	GI

The MenuPane subwidget attached to the MenuShell widget is constrained by the following resources. These resources can be set and read just like any other resources for the MenuShell.

Table 8-18 MenuPane Subwidget Resources[1]

Name	Type	Default	Access
XtNcenter	Boolean	TRUE	I
XtNhPad	Dimension	6	I
XtNhSpace	Dimension	6	I
XtNlayoutType	OlDefine	OL_FIXEDCOLS	I
XtNmeasure	int	1	I
XtNsameSize	OlDefine	OL_COLUMNS	I
XtNvPad	Dimension	3	I
XtNvSpace	Dimension	1	I

1. These subwidget resources are described in the section "ControlArea Widget" on page 249.

XtNmenuAugment

Class	Type	Default	Access
XtCMenuAugment	Boolean	TRUE	GI

Synopsis: The one responsible for popping up the menu.

Values: `TRUE/"true"` – The MenuShell widget augments its parent's event handling so that the pressing or clicking of MENU automatically pops up the menu.
`FALSE/"false"` – The application is responsible for detecting when the menu should be popped up, and for calling the routine `OlMenuPost(menu_widget)` to pop up the menu. See page 420.

XtNmenuPane

Class	Type	Default	Access
XtCMenuPane	Widget	NULL	G

Synopsis: The container widget that will be the parent of the menu items.

The value of this resource is available after the MenuShell widget has been created.

XtNpushpin

Class	Type	Default	Access
XtCPushpin	OlDefine	OL_NONE	GI

Synopsis: Whether the MenuShell widget has a pushpin.
Values: `OL_NONE/"none"` – No pushpin will be included in the list of menu items.
`OL_OUT/"out"` – A pushpin will be included as an item the user can select; if the user selects the pushpin, the menu will be made into an OPEN LOOK window. Note that the pushpin item is always at the top of the menu list.

Unlike other widgets, the value OL_IN is not allowed for the MenuShell widget.

XtNpushpinDefault

Class	Type	Default	Access
XtCPushpinDefault	Boolean	FALSE	SGI

Synopsis: Whether the pushpin is the default item.
Values: `TRUE/"true"` – The pushpin is the default item.
`FALSE/"false"` – The pushpin is not the default item.

If a menu has a pushpin and none of the menupane items has been designated as the default, the pushpin automatically becomes the menu's default item and the value of the resource will be updated to reflect this.

Activation Types

The following table lists the activation types used by the MenuShell.

Table 8-19 MenuShell Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVERIGHT	RIGHT	XtNrightKey
OL_MOVEUP	UP	XtNupKey
OL_MULTIDOWN	JUMP DOWN	XtNmultiDownKey
OL_MULTILEFT	JUMP LEFT	XtNmultiLeftKey
OL_MULTIRIGHT	JUMP RIGHT	XtNmultiRightKey
OL_MULTIUP	JUMP UP	XtNmultiUpKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_NEXTWINDOW	NEXTWINDOW	XtNnextWinKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_PREVWINDOW	PREVWINDOW	XtNprevWinKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

The MenuShell widget has no activation types besides the ones in “Common Activation Types” on page 68.

See Also

“ControlArea Widget” on page 249,
“MenuButton Widget” on page 403.

8

Nonexclusives Widget

Nonexclusives Widget

Class

Class Name: `Nonexclusives`
Class Pointer: `nonexclusivesWidgetClass`

Ancestry

Core-Composite-Constraint-Manager-Nonexclusives

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/Nonexclusi.h>
```

Description

The Nonexclusives widget provides layout management and selection control for a set of rectangular buttons or check boxes. It provides a simple way to build a several-of-many button selection object.

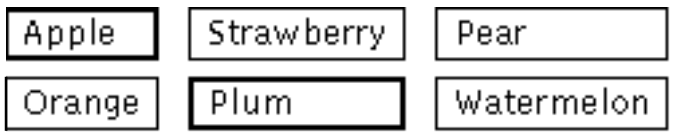

Figure 8-7 Nonexclusive Buttons Example

Layout and Labels

The Nonexclusives widget lays out the rectangular buttons or check boxes in a grid in the order they are added as child widgets by the application. The number of rows or columns in this grid can be controlled by the application. If the grid has more than one row, the Nonexclusives widget forces the rectangular buttons or check boxes in each column to be the same size as the widest in the column.

If the grid is a single row, each button will be only as wide as necessary to display the label.

The rectangular buttons or check boxes are separated by a distance that is 50% of the prevailing point size for the containing window.

Use in Menu

The Nonexclusives widget can be added as a single child to a menupane to implement a several-of-many menu choice. Only RectButton widgets can be used in a Nonexclusives widget in a menu.

Restrictions on Children

Child widgets of a Nonexclusives widget are required to be all from the same class--from either `rectButtonWidgetClass` or `checkBoxWidgetClass`.

Coloration

Any space not colored by the Nonexclusives children will be colored with the value of the `XtNbackground` or `XtNbackgroundPixmap` resource.

Keyboard Traversal

The Nonexclusives widget manages the traversal between a set of RectButtons. When the user traverses to a Nonexclusives widget, the first RectButton in the set will receive input focus. The MOVEUP, MOVEDOWN, MOVERIGHT, and MOVELEFT keys move the input focus between the RectButtons. To traverse out of the Nonexclusives widget, the following keys can be used:

- NEXTFIELD moves to the next traversable widget in the window
- PREVFIELD moves to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application

These controls have two states: “set” and “not set.” Pressing the SELECTKEY on a nonexclusive control will toggle the current state. If the control is in a menu, then the MENUKEY will also toggle the current state. If the control is “set,” then toggling the control will call the `XtNunselect` callback list. If the control is “not set,” then toggling the control will call the `XtNselect` callback list.

8

Nonexclusives Widget Resources

Table 8-20 Nonexclusives Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-21 Nonexclusives Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 8-22 Nonexclusives Manager Resources

Name	Type	Default	Access
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNinputFocusColor	Pixel	Red	SGID

Table 8-22 Nonexclusives Manager Resources (Continued)

Name	Type	Default	Access
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNunrealizeCallback	XtCallbackList	NULL	SGIO
XtNuserData	XtPointer	NULL	SGI

Table 8-23 Nonexclusives Resources

Name	Type	Default	Access
XtNlayoutType	OlDefine	OL_FIXEDROWS	SGI
XtNmeasure	int	1	SGI
XtNrecomputeSize	Boolean	TRUE	SGI

XtNlayoutType

Class	Type	Default	Access
XtCLayoutType	OlDefine	OL_FIXEDROWS	SGI

Synopsis: The type of layout of the child widgets.
Values: `OL_FIXEDROWS/"fixedrows"` - Set a fixed number of rows.
`OL_FIXEDCOLS/"fixedcols"` - Set a fixed number of columns.

The choices are to specify the number of rows or the number of columns. Only one of these dimensions can be specified directly; the other is determined by the number of child widgets added, and will always be enough to show all the child widgets.

XtNmeasure

Class	Type	Default	Access
XtCMeasure	int	1	SGI

Synopsis: The number of rows or columns in the layout of the child widgets.
Values: 0 < `XtNmeasure`

If there are not enough child widgets to fill a row or column, the remaining space is left empty.

XtNrecomputeSize

Class	Type	Default	Access
XtCRecomputeSize	Boolean	TRUE	SGI

Synopsis: The resize policy of the widget.
Values: `TRUE/"true"` – The widget resizes itself to accommodate changes in its children's sizes due to changes in resources such as fonts or labels.
`FALSE/"false"` – The widget does not resize itself.

Activation Types

The following table lists the activation types used by the Nonexclusives.

Table 8-24 Nonexclusives Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	LEFT	XtNleftKey
OL_MOVERIGHT	RIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_NEXTWINDOW	NEXTWINDOW	XtNnextWinKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

The Nonexclusives widget has no activation types besides the ones in "Common Activation Types" on page 68.

See Also

"Exclusives Widget" on page 277,
"FlatNonexclusives Widget" on page 347.

NoticeShell Widget

Class

Class Name: `NoticeShell`
Class Pointer: `noticeShellWidgetClass`

Ancestry

Core-Composite-Shell-WMShell-VendorShell-TransientShell-NoticeShell

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/Notice.h>
```

Description

The NoticeShell widget creates a popup window to notify the user of some condition. It is typically used to allow the user to confirm a choice, or to warn the user of a problem. When a NoticeShell is popped up, all other interaction with the application is precluded until the NoticeShell is popped down.

Components

The NoticeShell widget has four main components:

- Text Area – Where the message to the user is displayed
- Control Area – A container for one or more widgets that the user uses to control how to continue with an application
- Default Button – A button to acknowledge notice or to confirm action
- Emanate Widget – Typically the control requiring immediate attention that was activated by the user. (Emanate is not a named widget class. It is named this because it is the widget from which the NoticeShell "emanates," or pops up.) The application identifies the emanate widget to the NoticeShell widget.

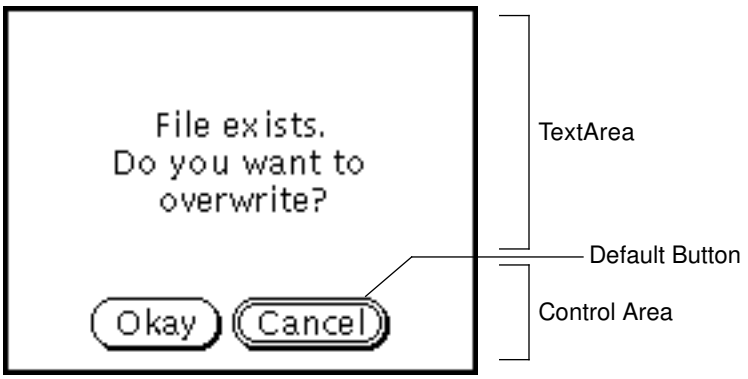

Figure 8-8 NoticeShell Widget

Subwidgets

The NoticeShell contains two subwidgets, a StaticText and a ControlArea, created automatically, and accessible through the `XtNtextArea` and `XtNcontrolArea` resources, respectively.

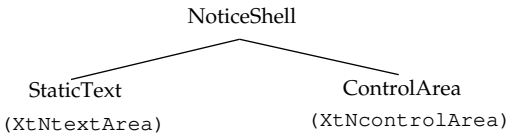

Figure 8-9 MenuShell Subwidgets

Subclass of the Shell Widget

The NoticeShell widget is a subclass of Shell; therefore, the `XtCreatePopupShell()` routine is used to create a notice instead of the normal `XtCreateWidget()`.

Popping the Notice Up and Down

The application controls when the NoticeShell widget is to be displayed or popped up, via the `XtPopup()` routine. However, the application does not need to control when the NoticeShell widget is popped down. If the control area subwidget of the NoticeShell contains any OblongButton widgets, the

NoticeShell widget will pop down when the user clicks SELECT on one of these buttons. The application can also pop down the NoticeShell itself using `XtPopDown()`.

Busy Button, Busy Application

When the NoticeShell pops up, the application may ignore further input from the user if the *`grab_kind`* argument to `XtPopup` is set to `XtGrabExclusive` (see the description of `XtPopup` in the *X Toolkit Intrinsics Reference Manual*). This prevents the user from interacting with anything other than the notice. As feedback of this to the user, the NoticeShell causes the headers of all the base windows and popup windows to be stippled in the *busy* pattern, and causes a stipple pattern in the emanate widget. The latter stipple pattern is caused by setting the `XtNbusy` resource to TRUE in the emanate widget. If the emanate widget does not recognize this resource, nothing will happen.

On popping down, the NoticeShell widget clears all stipple patterns and "unfreezes" the application, assuming the *`grab_kind`* argument to `XtPopup` had been set to `XtGrabExclusive`.

Text and ControlAreas

The Text and Control Areas of the NoticeShell are handled by separate widget interfaces. The widget IDs of these two areas can be obtained from the NoticeShell widget after it has been created using the `XtNtextArea` and `XtNcontrolArea` resources, respectively.

By default, the Text and ControlAreas abut with no space between them. The application can control the distance between them by setting margins on the ControlArea widget.

Keyboard Traversal

The NoticeShell widget limits keyboard traversal of the application to the buttons within the ControlArea.

The user can traverse between the controls in the ControlArea using the NEXTFIELD, PREVFIELD, MOVEUP, MOVEDOWN, MOVERIGHT, and MOVELEFT keys. The NEXTAPP key will traverse to the next application, and the PREVAPP key will traverse to the previous application, but the NEXTWINDOW and

PREVWINDOW keys are disabled. When keyboard traversal is used to move back to the NoticeShell's application, focus goes to the NoticeShell.

Coloration

The following diagram illustrates the resources that affect NoticeShell coloration.

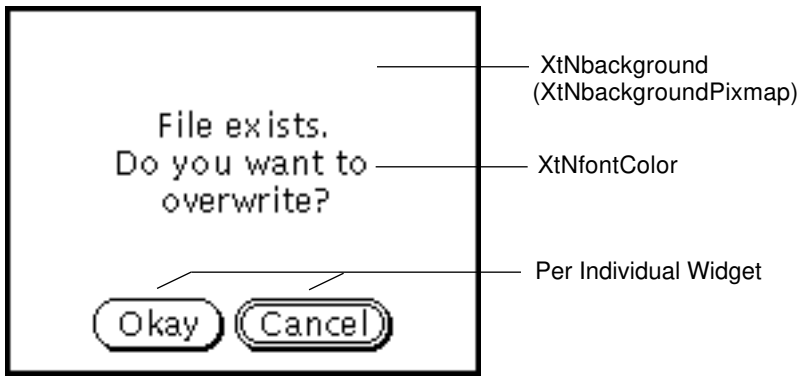

Figure 8-10 NoticeShell Coloration

Resources

Table 8-25 NoticeShell Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI

Table 8-25 NoticeShell Core Resources (Continued)

Name	Type	Default	Access
XtNscreen	Screen *	(parent's)	GI
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-26 NoticeShell Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 8-27 NoticeShell Shell Resources

Name	Type	Default	Access
XtNallowShellResize	Boolean	TRUE	SGI
XtNcreatePopupChildProc	XtCreatePopupChildProc	NULL	SGI
XtNgeometry	String	NULL	GI
XtNoverrideRedirect	Boolean	FALSE	SGI
XtNpopdownCallback	XtCallbackList	NULL	SGIO
XtNpopupCallback	XtCallbackList	NULL	SGIO
XtNsaveUnder	Boolean	TRUE	SGI
XtNvisual	Visual *	(parent's)	GIO

Table 8-28 NoticeShell WMShell Resources

Name	Type	Default	Access
XtNbaseHeight	int	XtUnspecifiedShellInt	SGI
XtNbaseWidth	int	XtUnspecifiedShellInt	SGI
XtNheightInc	int	XtUnspecifiedShellInt	SGI
XtNiconMask	Pixmap	NULL	SGI

Table 8-28 NoticeShell WMShell Resources (Continued)

Name	Type	Default	Access
XtNiconPixmap	Pixmap	NULL	SGI
XtNiconWindow	Window	NULL	SGI
XtNiconX	int	XtUnspecifiedShellInt	SGI
XtNiconY	int	XtUnspecifiedShellInt	SGI
XtNinitialState	InitialState	NormalState	SGI
XtNinput	Bool	FALSE	G
XtNmaxAspectX	int	XtUnspecifiedShellInt	SGI
XtNmaxAspectY	int	XtUnspecifiedShellInt	SGI
XtNmaxHeight	int	OL_IGNORE	SGI
XtNmaxWidth	int	OL_IGNORE	SGI
XtNminAspectX	int	XtUnspecifiedShellInt	SGI
XtNminAspectY	int	XtUnspecifiedShellInt	SGI
XtNminHeight	int	OL_IGNORE	SGI
XtNminWidth	int	OL_IGNORE	SGI
XtNtitle	String	NULL	SGI
XtNtitleEncoding	ATom	XA_STRING	SGI
XtNtransient	Boolean	TRUE	SGI
XtNwaitForWm	Boolean	TRUE	SGI
XtNwidthInc	int	XtUnspecifiedShellInt	SGI
XtNwindowGroup	Window	XtUnspecifiedWindow	SGI
XtNwinGravity	int	XtUnspecifiedShellInt	SGI
XtNwmTimeout	int	5000 (msec)	SGI

Table 8-29 NoticeShell VendorShell Resources

Name	Type	Default	Access
XtNbusy	Boolean	FALSE	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNdefaultImName	String	NULL	SGI
XtNfooterPresent	Boolean	FALSE	SGI
XtNfocusWidget	Widget	(see description)	SGI
XtNimFontSet	OlFont	XtDefaultFontSet	SGI
XtNimStatusStyle	XtOlImStatusStyle	OL_NO_STATUS	GI
XtNleftFooterString	OlStr	NULL	SGI

Table 8-29 NoticeShell VendorShell Resources (Continued)

Name	Type	Default	Access
XtNleftFooterVisible	Boolean	TRUE	SGI
XtNmenuButton	Boolean	(see description)	GI
XtNmenuType	OlDefine	(see description)	SGI
XtNpushpin	OlDefine	(see description)	SGI
XtNresizeCorners	Boolean	(see description)	SGI
XtNrightFooterString	OlStr	NULL	SGI
XtNrightFooterVisible	Boolean	TRUE	SGI
XtNshellTitle	OlStr	NULL	SGI
XtNuserData	XtPointer	NULL	SGI
XtNwindowHeader	Boolean	(see description)	GI
XtNwmProtocol	XtCallbackList	NULL	SGIO
XtNwmProtocolInterested	int	OL_WM_DELETE_WINDOW | OL_WM_TAKE_FOCUS	I

Table 8-30 NoticeShell TransientShell Resources

Name	Type	Default	Access
XtNtransientFor	Widget	NULL	SGI

Table 8-31 NoticeShell Resources

Name	Type	Default	Access
XtNcontrolArea	Widget	NULL	G
XtNemanateWidget	Widget	(parent's)	SGI
XtNpointerWarping	Boolean	TRUE	SGI
XtNtextArea	Widget	NULL	G
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI

The following table lists resources passed to the StaticText subwidget maintained by the NoticeShell; they become resources of the StaticText subwidget and can be set and read just like any other NoticeShell resources.

Table 8-32 NoticeShell StaticText Subwidget Resources[1]

Name	Type	Default	Access
XtNalignment	OlDefine	OL_LEFT	I
XtNfont	OlFont	XtDefaultFont	I
XtNfontColor	Pixel	XtDefaultForeground	I
XtNlineSpace	int	0	I
XtNstring	OlStr	NULL	I
XtNstrip	Boolean	TRUE	I
XtNwrap	Boolean	TRUE	I

1. These subwidget resource are defined in the section "StaticText Widget" on page 600.

The following table lists resources passed to the ControlArea subwidget maintained by the NoticeShell; they become resources of the ControlArea subwidget and can be set and read just like any other NoticeShell resources.

Table 8-33 NoticeShell ControlArea Subwidget Resources[1]

Name	Type	Default	Access
XtNhPad	Dimension	4	I
XtNhSpace	Dimension	4	I
XtNlayoutType	OlDefine	OL_FIXEDROWS	I
XtNmeasure	int	1	I
XtNsameSize	OlDefine	OL_COLUMNS	I
XtNvPad	Dimension	4	I
XtNvSpace	Dimension	4	I

1. These subwidget resource are defined in the section "ControlArea Widget" on page 249.

XtNcontrolArea

Class	Type	Default	Access
XtCControlArea	Widget	NULL	G

Synopsis: The widget ID of the ControlArea widget.

This value is available once the NoticeShell widget has been created. Any widgets of the class OblongButton added to the Control Area are assumed to be window disposition controls; that is, when the user activates one of them, the NoticeShell widget pops itself down.

XtNemanateWidget

Class	Type	Default	Access
XtCEmanateWidget	Widget	(parent's)	SGI

Synopsis: The widget ID of the emanate widget.

On popping up, the NoticeShell widget attempts to set this widget to be busy, by making its `XtNbusy` resource TRUE; if the widget doesn't recognize the resource, nothing happens.

On popping down, the NoticeShell widget clears the `XtNbusy` resource. When the NoticeShell widget pops up, it tries not to cover this widget; this may fail depending on its location and the size of the NoticeShell widget. The default for this resource is the parent. The parent, however, cannot be a gadget (an OblongButtonGadget, for instance). To emanate a NoticeShell from a gadget, specify another widget as the parent and set `XtNemanateWidget` to the gadget.

XtNpointerWarping

Class	Type	Default	Access
XtCPointerWarping	Boolean	TRUE	SGI

Synopsis: Whether the pointer will jump to the default button when the Notice is displayed.

Values: `TRUE/"true"` – The pointer jumps to the default button.
`FALSE/"false"` – The pointer doesn't jump.

XtNtextArea

Class	Type	Default	Access
XtCTextArea	Widget	NULL	G

Synopsis: The widget ID of the StaticText widget that controls the text area.

This value is available once the NoticeShell widget has been created.

XtNtextFormat

Class	Type	Default	Access
XtCTextFormat	OlStrRep	OL_SB_STR_REP	GI

Synopsis: The expected data format of all the textual resources of a widget.
Values: `OL_SB_STR_REP` – Single-byte character representation.
`OL_WC_STR_REP` – Wide character representation.
`OL_MB_STR_REP` – Multibyte character representation.

See "XtNtextFormat" on page 29 for details of initialization and the default value.

Activation Types

The following table lists the activation types used by the NoticeShell.

Table 8-34 NoticeShell Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

The NoticeShell widget has no activation types besides the ones in "Common Activation Types" on page 68.

See Also

"ControlArea Widget" on page 249,
"StaticText Widget" on page 600.

NumericField Widget

Class

Class Name: `NumericField`
Class Pointer: `numericFieldWidgetClass`

Ancestry

Core-Primitive-TextLine-NumericField

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/AbbrevMenu.h>
```

Description

The NumericField widget is a one-line input field for alphanumeric text. It implements the Numeric TextField object defined in the *OPEN LOOK GUI Functional Specification*. The widget supports the basic datatypes—integers and floats—and provides hooks for more complicated user-defined datatypes. Providing support for any datatype involves:

- Implementing Converters between the datatype and `String`
- Implementing per-key and per-field validation
- Implementing increment and decrement operations

Once the input focus is moved into the widget, keyboard entry is allowed. If the input value exceeds the length of the input field, the ScrollButtons appear. Hidden text can then be scrolled into view by pressing the ScrollButtons. Pressing the buttons continuously scrolls the text repeatedly with a user-adjustable delay. The Increment or Decrement buttons can be used to increment or decrement the input value in an appropriate manner.

Components

The NumericField contains the following graphical elements:

- Right-justified bold label at the left of the input field.

- Input field
- Input caret (not present in ReadOnly mode)
- 1-point (for Mono) or chiseled underline (not present in ReadOnly mode)
- Optional ScrollButtons
- Increment and Decrement buttons, referred to as *DeltaButtons* hereafter

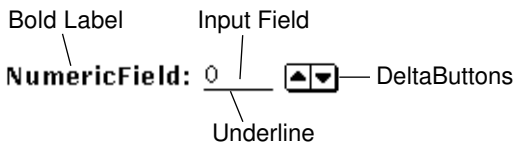

Figure 8-11 NumericField Components

Keyboard Traversal

The NumericField allows keyboard entry if it is sensitive and it has the keyboard focus. However, in ReadOnly mode the widget is not traversable and it does not receive keyboard focus. Selection is also turned off during ReadOnly mode.

The NumericField responds to the following keyboard navigation keys:

- NEXTFIELD moves to the next traversable widget in the window
- PREVFIELD moves to the previous traversable widget in the window
- CHARFWD moves the caret forward one character
- CHARBAK moves the caret backward one character
- WORDFWD moves the caret forward one word
- WORDBAK moves the caret back one word
- LINESTART moves the caret to the beginning of the display
- LINEEND moves the caret to the end of the display
- MENUKEY posts the menu associated with the NumericField

The NumericField responds to the following edit keys:

- DELCHARFWD deletes the character to the right of the caret
- DELCHARBAK deletes the character to the left of the caret
- DELWORDFWD deletes the word to the right of the caret
- DELWORDBAK deletes the word to the left of the caret
- DELLINEFWD deletes to the end of the line from the caret
- DELLINEBAK deletes from the beginning of the line to the caret
- DELLINE deletes the line containing the caret
- UNDO undoes the last edit

Keyboard Mnemonic Display

The NumericField does not display any mnemonic. If the NumericField is the child of a Caption widget, the Caption can be used to display any mnemonic.

Keyboard Accelerator Display

The NumericField does not respond to any keyboard accelerators.

Display of Text

The NumericField displays its contents in the font specified by the `XtNfont` resource. If the length of the text exceeds the length of the input field, the widget sets up the left or right scrollbuttons (or both) to indicate this. The text is then visually truncated at the boundaries to show only as many characters as can fit in the input field. The truncation is always at a character boundary. A scrollbutton is present only if characters are hidden in the direction indicated by that buttons. The user can scroll to show the hidden parts of the text by clicking or pressing the scrollbuttons. Clicking SELECT on any scrollbutton will scroll the text one character in the direction indicated by that button. Pressing SELECT on any scrollbutton will repeat the scrolling with a user-adjustable delay between each scroll.

Caret Position

As characters are entered from the keyboard, the caret moves to the right until it reaches the right end of the input field. As additional characters are typed, the text jump-scrolls to the left by a specific amount. Note that the NumericField always keeps the cursor visible. Thus, the presence or absence of either of the scrollbuttons is controlled by the current cursor position.

Selection Of Text

The user can perform text selection by using the mouse or the keyboard. The widget also provides a set of convenience functions to manipulate the selection programmatically; see “TextLine Functions” on page 708.

DeltaButtons

The DeltaButtons can be used to increment or decrement the value in the NumericField by a specified amount. The widget provides default logic for the supported datatypes (`int`s and `float`s) as well as hooks for user-defined datatypes. The DeltaButtons become insensitive when the value equals the maximum or minimum range.

Coloration

For 3D and 2D, `XtNfontColor` is used to draw the NumericField's text and `XtNinputFocusColor` is used to draw the active caret.

For 3D, the NumericField's underline and increment/decrement button coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1.

For 2D, `XtNbackground` and `XtNforeground` are used to render the NumericField's underline and increment/decrement buttons as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

Known Deficiencies

The TextLine widget, on which the NumericField is built, currently does not support the *implicit commit* feature; see "Input Method" on page 80. This could be a deficiency in Asian locales.

Resources

Table 8-35 NumericField Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI

Table 8-35 NumericField Core Resources (Continued)

Name	Type	Default	Access
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-36 NumericField Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGI
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	SGI
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

8

NumericField Widget

Table 8-37 NumericField TextLine Resources[1]

Name	Type	Default	Access
XtNblinkRate	int	1000	SGI
XtNcaptionAlignment	OlDefine	OL_CENTER	G
XtNcaptionFont[2]	OlFont	OlDefaultBoldFont	SI
XtNcaptionLabel[2]	OlStr	NULL	SGI
XtNcaptionPosition[2]	OlDefine	OL_LEFT	G
XtNcaptionSpace[2]	Dimension	4	G
XtNcaptionWidth[2]	Dimension	0	G
XtNcharsVisible	int	0	GI
XtNcommitCallback	XtCallbackList	(special)	SGIO
XtNcursorPosition	int	0	SGI
XtNeditType	OlDefine	OL_TEXT_EDIT	SGI
XtNimPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI
XtNinitialDelay	int	500	SGI
XtNinsertTab	Boolean	FALSE	SGI
XtNmaximumChars	int	0	GI
XtNmenu	Widget	NULL	GI
XtNmotionCallback	XtCallbackList	NULL	SGIO
XtNpostModifyCallback	XtCallbackList	NULL	SGIO
XtNpreModifyCallback	XtCallbackList	(special)	SGIO
XtNrepeatRate	int	100	SGI
XtNstring	OlStr	NULL	SGI
XtNupdateDisplay	Boolean	TRUE	SGI

1. These resources are defined in the section "TextLine Widget" on page 688.

2. These resources are provided by the TextLine to support captions and are subject to change in a future OLIT release.

Table 8-38 NumericField Resources

Name	Type	Default	Access
XtNconvertProc	OlNFConvertProc	(special)	I
XtNdelta	XtPointer	1 for ints; 0.1 for floats; else NULL	SGI
XtNdeltaCallback	XtCallbackList	NULL	SGIO

Table 8-38 NumericField Resources (Continued)

Name	Type	Default	Access
XtNdeltaState	OlDefine	OL_ACTIVE	SGI
XtNmaxValue	XtPointer	NULL	SGI
XtNminValue	XtPointer	NULL	SGI
XtNsizeOf	Cardinal	(calculated)	GI
XtNtype	String	XtRInt	GI
XtNvalidateCallback	XtCallbackList	NULL	SGIO
XtNvalue	XtPointer	NULL	SGI

The NumericField supports the basic datatypes `int` and `float`. The widget also provides a mechanism for applications to implement their own datatypes. Since the widget can support multiple datatypes, all resources that refer to any data are uniformly represented as pointers to the actual data, which could be any of the above widget-supported types or the application's specific type. Implementing additional datatypes is discussed in "Implementing New Datatypes" on page 455.

XtNconvertProc

Class	Type	Default	Access
XtCConvertProc	XtROlNFConvertProc	(special)	I

Synopsis: The widget-defined `int/float` to `String` converter.

This resource specifies the logic to convert between `String` and the user-defined datatype. This needs to be set only when the application has defined a custom datatype. See "Implementing New Datatypes" on page 455.

XtNdelta

Class	Type	Default	Access
XtCDelta	XtPointer	1 for ints; 0.1 for floats; else NULL	SGI

Synopsis: The Increment or Decrement delta to be applied to the data field, when either of the DeltaButtons is pressed. The widget applies the delta internally for `ints` and `floats`.

By default, the widget makes internal copies of the data. Setting the `XtNsizeOf` resource to zero prevents the widget from doing so. See the section

on "Implementing New Datatypes" on page 455 for reasons and techniques of doing this.

XtNdeltaState

Class	Type	Default	Access
XtCDeltaState	OlDefine	OL_ACTIVE	SGI

Synopsis: The state of the DeltaButtons.

Values: `OL_ACTIVE/"active"` – Both buttons are active.
`OL_INCR_INACTIVE/"incr_inactive"` – The increment button is inactive, the decrement button is active.
`OL_DECR_INACTIVE/"decr_inactive"` – The decrement button is inactive, the increment button is active.
`OL_ABSENT/"absent"` – DeltaButtons are not present.

XtNdeltaCallback

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when either of the DeltaButtons is pressed. If the buttons are pressed continuously, the callback will be invoked multiple times. The callback is not invoked if the DeltaButtons are not present.

The *call_data* structure is:

```
typedef struct {
    int             reason;
    XEvent          *event;
    XtPointer       delta;
    XtPointer       current_value;
    OlDefine        current_delta_state;
    XtPointer       new_value;
    OlDefine        new_delta_state;
    Boolean         update;
} OlNFDeltaCallbackStruct;
```

The fields signify:

reason	OL_REASON_INCREMENT or OL_REASON_DECREMENT
event	A pointer to the corresponding `XEvent` structure
delta	The value of the `XtNdelta` resource
current_value	The current value of the `XtNvalue` resource

current_delta_state	The current value of `XtNdeltaState`
new_value	The new value of the `XtNvalue` resource if the widget's default delta logic (if any) is carried out. The application can override the default behavior by changing the contents of this field to any new value.
new_delta_state	The new value of the `XtNdeltaState` if the widget's default delta logic (if any) is carried out. The application can override the default behavior by changing this field to any new value.
update	This resource should be set to TRUE to indicate that the *new_value* has changed.

If `XtNtype` is user-defined, then modifying *new_value* to the new value and setting update to TRUE will cause the widget to update `XtNvalue` with the new value, erase the *old_value*, and display the *new_value*. If the value of the *new_delta_state* field changes, the widget will update the `XtNdeltaState` resource and redisplay the DeltaButtons appropriately.

If `XtNtype` is `XtRFloat/XtRInt`, the *new_value* field is computed by the widget by incrementing/decrementing `XtNvalue` by `XtNdelta` and validating it against `XtNmaxValue/XtNminValue`. `XtNdeltaState` is also set up appropriately by the widget. The application can override this default logic by modifying *new_value* and setting update to TRUE. This would cause the widget to update `XtNvalue` with the new value, erase the *old_value*, and display the *new_value*. Note that no increment/decrement or validation will be done by the widget. If the value of the *new_delta_state* field is changed, the widget would update the `XtNdeltaState` resource and redisplay the DeltaButtons appropriately. This is also the default behavior (i.e., if no callbacks are registered).

XtNmaxValue/ XtNminValue

Class	Type	Default	Access
XtCMaxValue	XtPointer	NULL	SGI
XtCMinValue	XtPointer	NULL	SGI

Synopsis: The maximum (minimum) value that can be set for the data field.

For datatypes `XtRInt` and `XtRFloat`, the widget does the following if this resource is not NULL:

- While incrementing (decrementing) the data field, if the value is greater than or equal to `XtNmaxValue` (less than or equal to `XtNminValue`), the increment (decrement) button is desensitized.
- When the value is committed by the user, the widget verifies that the value does not exceed `XtNmaxValue` (is not less than `XtNminValue`).

By default, the widget makes internal copies of the data. Setting the `XtNsizeOf` resource to zero prevents the widget from doing so. See the section on "Implementing New Datatypes" on page 455 for reasons and techniques of doing this.

XtNsizeOf

Class	Type	Default	Access
XtCSizeOf	Cardinal	(calculated)	GI

Synopsis: The size of the datatype that is used by the application.
Values: Size of the datatype or zero.

This value needs to be set only when the application has defined a custom datatype. The special value of zero can be used to indicate that the widget need not make copies of any resources. See the section on "Implementing New Datatypes" on page 455 for reasons and techniques of doing this.

XtNtype

Class	Type	Default	Access
XtCCType	String	XtRInt	GI

Synopsis: The type of the data.
Values: `XtRInt` – Integer
`XtRFloat` – Float

The application can also define a custom datatype, as described in "Implementing New Datatypes" on page 455.

XtNvalidateCallback

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The validation callback.

The *call_data* structure is:

```
typedef struct {
    int             reason;
    XEvent          *event;
    XtPointer       value;
    OlDefine        delta_state;
    Boolean         update;
    Boolean         valid;
} OlNFValidateCallbackStruct;
```

This callback is invoked when the user presses the RETURN, NEXTFIELD, or PREVFIELD keys. The fields signify:

reason	OL_REASON_VALIDATE
event	A pointer to the corresponding `XEvent` structure
value	The current value of the `XtNvalue` resource. The application can make appropriate changes to this field and set the update flag to update `XtNvalue`.
delta_state	The current value of `XtNdeltaState`. This can be set to a new value to update this resource.
update	This resource should be set to TRUE to indicate that the value has changed.
valid	A field that indicates the validity of the contents.

If `XtNtype` is `XtRInt` or `XtRFloat`, then the widget will internally validate the *value* against `XtNmaxValue` and `XtNminValue`. If the value is invalid, it sets *valid* to FALSE. The callback can take appropriate action, like beeping, popping up a notice, etc. It can also reset the *value* and *delta_state* resources to suitable default values by modifying those fields and setting *update* to TRUE. On return from the callback, if *valid* is FALSE, the widget maintains focus within the widget. If *valid* is TRUE, the widget relinquishes focus. If `XtNtype` is user-defined, the widget invokes this callback immediately. Depending on the *valid* flag, the widget will maintain focus or relinquish focus.

XtNvalue

Class	Type	Default	Access
XtCValue	XtPointer	NULL	SGI

Synopsis: The pointer to an `int`, `float`, or the user-defined datatype.

The contents of the pointer is the data to be displayed. The data is converted into a string (depending on the datatype) and used as the `XtNstring` resource of the superclass, the TextLine widget.

By default, the NumericField widget makes internal copies of the data. Setting the `XtNsizeOf` resource to zero prevents the widget from doing so. See the section on "Implementing New Datatypes" on page 455 for reasons and techniques of doing this.

Examples for `XtVaSetValues()`:

```
int i = 10;
XtVaSetValues(w, XtNvalue, &i, NULL);
...
int *ip;
XtVaGetValues(w, XtNvalue, &ip, NULL);
```

In the second example, *ip* now points to data in the widget's space. Therefore, the application must make copies if it needs to manipulate this data.

Per-Key and Per-Field Validation

The NumericField widget uses the callbacks of its superclass to implement per-key and per-field validation, as described below.

The NumericField widget itself does per-key validation for the datatypes `XtRInt` and `XtRFloat`. The widget registers `XtNpreModifyCallbacks` on its superclass (TextLine) for `int`s and `float`s. The callback for `int` verifies that each keystroke is a valid digit; the callback for `float` ensures that each keystroke is either a digit or a decimal separator and that only one decimal separator occurs in the field. Applications can override this behavior by removing the widget-registered callback and installing their own callback. For user-defined datatypes, the application can register a customized per-key validation callback.

The NumericField widget also does per-field validation for the datatypes `XtRInt` and `XtRFloat`, which verifies that the field-value is within the minimum and maximum ranges. The widget registers a callback on the `XtNcommitCallback` of the TextLine superclass. This callback verifies the field-value against the minimum and maximum ranges, if they exist. On success, the widget sets the *valid* field of the `OlNFValidateCallbackStruct` to TRUE and invokes any `XtNvalidateCallback` callbacks registered. On

failure, the *valid* flag is set to FALSE and any `XtNvalidateCallback` callbacks present are invoked.

Implementing New Datatypes

The NumericField widget currently supports `int` and `float` as the base datatypes; i.e., the widget internally provides all logic and code necessary to handle these datatypes. The widget also allows applications to define and use their own datatypes. Examples of such user-defined datatypes would include Date, Time, Currency, etc. Implementing a new datatype involves the following steps:

1. **Register the new datatype with the widget.**
2. **Implement Converters between the datatype and String.**
3. **Optionally implement Per-Key and Per-Field Validation routines.**
4. **Optionally implement Increment and Decrement operations.**

Each step is elaborated below:

1. The application specifies a name for the new datatype. The name should be a printable ASCII `String`. This name is registered with the widget by setting it up as the `XtNtype` resource. The application should also set the `XtNsizeOf` resource to the size in bytes of the new datatype. The widget internally uses this value while making copies of the contents of `XtNvalue`, `XtNdelta`, `XtNminValue`, and `XtNmaxValue`. `XtNsizeOf` can be set to the special value of '0' to indicate that the widget need not make copies of the above resources. This will result in the widget accessing these values directly through pointers to the application's dataspace.

   Typically, a programmer would want to do this if the datatype is large in size and the application stores all the above resource values in global space or space allocated by `malloc`(3), so that copying between the application's dataspace and the widget's dataspace is redundant or inefficient. Caution should be used, however, since modifying any of these values also directly affects the widget's state.

2. The NumericField displays the contents of `XtNvalue` by converting the contents to a string and setting it up as the `XtNstring` resource of its superclass—the TextLine. Similarly, the widget updates `XtNvalue` by retrieving the `XtNstring` resource value of the TextLine and converting it

back to the specified datatype. The widget does these conversions internally for the base datatypes. However, for user-defined datatypes, the application needs to provide this conversion logic. This is done by setting up the `XtNconvertProc` function. The prototype of this function is as follows:

```
Boolean Converter(
    Widget          w,
    OlStrRep        format,
    XrmQuark        from_type,
    XrmQuark        to_type,
    XtPointer       *value,
    XtPointer       *string,
    Cardinal        *string_length);
```

w	The widget.
format	The *text_format* of the widget.
from_type	The source datatype; i.e, the datatype to be converted from. This is a quarkified representation of the datatype name.
to_type	The destination datatype; i.e., the datatype to be converted to. This is a quarkified representation of the datatype name.
value	A pointer to the data stored.
string	A pointer to the data stored as a string.
string_length	The size of the memory where the string is stored.

The Converter should do the following:

- Attempt the type conversion from the source datatype to the destination datatype.
- Copy the converted data to the appropriate location; i.e., in *string* if the destination datatype is `String`, in *value* if the destination datatype is the user's datatype. Before copying, ensure that the location is a valid memory pointer and that the storage size is sufficient. If not, allocate sufficient memory and then copy the data. Update **string_length* to the length of the string if the destination datatype is `String`.
- If the conversion failed, issue a warning and return FALSE; otherwise, return TRUE. The *string* datatype is identified as `XtRString`.

3. Per-Key validation ensures that each key is valid in the current context. The application can verify this by registering an appropriate `XtNpreModifyCallback` on the widget.

Per-Field validation ensures that the contents of the widget are valid, when the user commits a value (i.e., presses <return> or <Tab>). The application can achieve this by registering an appropriate `XtNvalidateCallback`. The callback should set the *valid* field of the *call_data* to indicate the validity of the data. The callback can also reset the widget contents by updating *value* and setting *update* to TRUE.

4. The widget should update its contents and the state of its DeltaButtons whenever the buttons are pressed. The widget does this internally for the base datatypes. For user-defined datatypes, the application can implement the necessary logic by registering an appropriate `XtNdeltaCallback`. The callback should update the *new_value* and *new_delta_state* fields in the *call_data* and set *update* to TRUE, if the contents of the widget need to be changed.

Activation Types

The following table lists the activation types used by the NumericField.

Table 8-39 NumericField Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_CHARBAK	LEFT	XtNleftKey
OL_CHARFWD	RIGHT	XtNrightKey
OL_COPY	COPY	XtNcopyBtn
OL_CUT	CUT	XtNcutBtn
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_DELCHARBAK	DELETE BACKWARD	XtNdelCharBakFwd
OL_DELCHARFWD	DELETE FORWARD	XtNdelCharFwdKey
OL_DELLINE	DELETE LINE	XtNdelLineKey
OL_DELLINEBAK	DELLINEBAK	XtNdelLineBakKey
OL_DELLINEFWD	DELLINEFWD	XtNdelLineFwdKey
OL_DELWORDBAK	DELWORDBAK	XtNdelWordBakKey
OL_DELWORDFWD	DELWORDFWD	XtNdelWordFwdKey
OL_HELP	HELP	XtNhelpKey
OL_LINEEND	ROW END	XtNlineEndKey
OL_LINESTART	ROW START	XtNlineStartKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey

Table 8-39 NumericField Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PASTE	PASTE	XtNpasteBtn
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey
OL_UNDO	UNDO	XtNundoKey
OL_WORDBAK	JUMP LEFT	XtNwordBakKey
OL_WORDFWD	JUMP RIGHT	XtNwordFwdKey

Activation types not described in the following table are described in "Common Activation Types" on page 68.

OL_CHARBAK

The cursor is moved backward by one character. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_CHARFWD

The cursor is moved forward by one character. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_COPY

This activation type copies the current selection from the widget to the CLIPBOARD.

OL_CUT

This activation type copies the current selection from the widget to the CLIPBOARD and also deletes the selected text from the widget. The following callbacks are invoked:

`XtNpreModifyCallback` Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.

`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELCHARBAK

If there exists a selection in the widget, it is deleted. If there is no selection, the character before the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELCHARFWD

If there exists a selection in the widget, it is deleted. If there is no selection, the character after the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELLINE

The whole line is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELLINEBAK

If there exists a selection in the widget, it is deleted. If there is no selection, the segment of the line before the insert point The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELLINEFWD

If there exists a selection in the widget, it is deleted. If there is no selection, the segment of the line after the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELWORDBAK

If there exists a selection in the widget, it is deleted. If there is no selection, the word before the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELWORDFWD

If there exists a selection in the widget, it is deleted. If there is no selection, the word after the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_LINESTART

The cursor is moved to the start of the line. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_LINEEND

The cursor is moved to the end of the line. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_NEXTFIELD

This activation type invokes the `XtNcommitCallback`. If the callback sets *valid* to TRUE and if `XtNmouseless` is TRUE, the widget transfers focus to the next traversable widget. If the callback sets *valid* to TRUE and if `XtNmouseless` is FALSE, the widget transfers focus and sets the insert point to the next TextLine or TextEdit widget. If the callback sets *valid* to FALSE, the widget maintains focus and insert point within itself.

OL_PASTE

This activation type inserts the contents of the CLIPBOARD into the widget at the current insert point. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the insertion occurs. The callback can prevent the insertion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the insertion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the insertion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_PREVFIELD

This activation type invokes the `XtNcommitCallback`. If the callback sets *valid* to TRUE and if `XtNmouseless` is TRUE, the widget transfers focus to the previous traversable widget. If the callback sets *valid* to TRUE and if `XtNmouseless` is FALSE, the widget transfers focus and sets the insert point to the previous TextLine or TextEdit widget. If the callback sets *valid* to FALSE, the widget maintains focus and insert point within itself.

OL_UNDO

This activation type undoes the last modification to the widget's text buffer. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before any modification occurs. The callback can prevent the modification by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after any modification occurs.

`XtNmotionCallback`	Invoked before the cursor position is changed due to the modification. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_WORDBAK

The cursor is moved backward by one word. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_WORDFWD

The cursor is moved forward by one word. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

See Also

"TextLine Widget" on page 688.

8

OblongButton Widget

Class

Class Name: `OblongButton`
Class Pointer: `oblongButtonWidgetClass,oblongButtonGadgetClass`

Ancestry

Core-Primitive-Button-OblongButton

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/OblongButt.h>
```

Description

The OblongButton is an action widget that the user can "push" by pressing SELECT on it. When the button is pushed, its border inverts, making it appear as if the button has actually been pressed. It is typically used to initiate one or several application-defined actions.

Components

The OblongButton consists of a label surrounded by a rounded, or oblong, border.

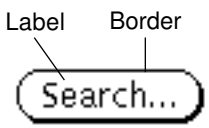

Figure 8-12 OblongButton Components

Busy Indication During Callbacks

Each OblongButton is associated with an application-defined action implemented as a list of callbacks. To let the user know that an action is still

taking place, the OblongButton stipples the area inside the border before issuing the callbacks. When the last callback returns, the OblongButton restores its original appearance. If the application's action continues to be "busy" after the callbacks return, the application should set the `XtNbusy` resource to TRUE before returning from the callbacks, then reset it to FALSE when the action is no longer taking place.

The "busy" stipple pattern is designed to show enough dots to gray the button noticeably, while still leaving a text label legible.

OblongButtons In Popup Menus

Entering an oblong button while MENU is depressed highlights the button's interior. Releasing MENU then restores the original appearance and invokes the action for the button as described above. Leaving the button before releasing MENU restores the appearance but does not invoke the action.

OblongButtons Not In Popup Menus

Clicking SELECT on an OblongButton starts the action associated with the button. Pressing SELECT, or moving the pointer into the button while SELECT is pressed, highlights the button's interior. Releasing SELECT restores the appearance and invokes the action for the button as described above. Moving the pointer off the button before releasing SELECT also restores the appearance, but does not invoke the action.

If the OblongButton is in a stay-up menu, clicking or pressing MENU works the same as SELECT. If the OblongButton is not in a stay-up (or popup) menu, clicking or pressing MENU does not do anything; the event is passed up to an ancestor widget.

OblongButton Gadgets

Correct button behavior is not guaranteed if gadgets are positioned so that they overlap.

Gadgets share some Core fields but, since they are not subclasses of Core, do not have all Core fields. In particular, they do not have a name field or a translation field, so translations cannot be specified or overridden.

Event Handlers cannot be added to gadgets using `XtAddEventHandler()`.

Coloration

For 3D, OblongButton coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1. `XtNfontColor` is used to draw the label.

For 2D, `XtNbackground` and `XtNfontColor` are used to render the OblongButton as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

If the toolkit resource `XtNmouseless` is set to TRUE and the toolkit resource `XtNinputFocusFeedback` is set to OL_INPUT_FOCUS_COLOR, then the background of the OblongButton will be drawn with the value of `XtNinputFocusColor` when the widget receives input focus. However, if `XtNinputFocusColor` is the same as `XtNbackground`, then the OblongButton inverts `XtNfontColor` and `XtNbackground`. Once the input focus leaves the widget, the original coloration is restored.

Keyboard Traversal

The default value of the `XtNtraversalOn` resource is TRUE.

The OblongButton widget responds to the following keyboard navigation keys:

- NEXTFIELD, MOVEDOWN, and MOVERIGHT move to the next traversable widget in the window
- PREVFIELD, MOVEUP, and MOVELEFT move to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application

The OblongButton will respond to the SELECTKEY by acting as if the SELECT buttons had been clicked.

Keyboard Mnemonic Display

The OblongButton widget displays its keyboard mnemonic as part of its label. If the mnemonic character is in the label, then that character is highlighted according to the value of the `XtNshowMnemonics` toolkit resource. If the mnemonic character is not in the label, it is displayed to the right of the label in parentheses and highlighted according to the value of the `XtNshowMnemonics` resource.

If label truncation is necessary, the mnemonic displayed in parentheses is truncated as a unit.

Keyboard Accelerator Display

The display of keyboard accelerators is controlled by the toolkit resource `XtNshowAccelerators`. When the value of `XtNshowAccelerators` is OL_DISPLAY, the OblongButton widget displays the keyboard accelerator as part of its label. The string in the `XtNacceleratorText` resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

If label truncation is necessary, the accelerator is truncated as a unit. The accelerator is truncated before the mnemonic or the label.

Label Appearance

The `XtNwidth`, `XtNheight`, `XtNrecomputeSize`, and `XtNlabelJustify` resources interact to produce a truncated, clipped, centered, or left-justified label as shown in the following table.

Table 8-40 OblongButton Label Appearance

XtNwidth	XtNrecomputeSize	XtNlabelJustify	Result
any value	TRUE	any	Just Fits
> needed for label	FALSE	OL_LEFT	Left Justified
> needed for label	FALSE	OL_CENTER	Centered
< needed for label	FALSE	any	Trunc
XtNheight	**XtNrecomputeSize**	**XtNlabelJustify**	**Result**
any value	TRUE	any	Just Fits
> needed for label	FALSE	any	Centered
< needed for label	FALSE	any	Clipped

When the label is centered or left-justified, the extra space is filled with the background color of the OblongButton widget, as determined by the `XtNbackground` and `XtNbackgroundPixmap` resources. When a text label is truncated, the truncation occurs at a character boundary and a solid triangle is inserted to show that part of the label is missing. The triangle requires that more of the label be truncated than would otherwise be necessary. If the width of the button is too small to show even one character with the triangle, only the triangle is shown. If the width is so small that the entire triangle cannot be shown, the triangle is clipped on the right. An image label is simply truncated; no triangle is shown. See also the `XtNlabelTile` resource for how it affects the appearance of a label image.

Resources

Table 8-41 OblongButton Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	(calculated)	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	(calculated)	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-42 OblongButton Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	SGI
XtNreferenceName	String	NULL	GI

Table 8-42 OblongButton Primitive Resources (Continued)

Name	Type	Default	Access
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 8-43 OblongButton Resources

Name	Type	Default	Access
XtNbusy	Boolean	FALSE	SGI
XtNdefault	Boolean	FALSE	SGI
XtNlabel	OlStr	(instance name)	SGI
XtNlabelImage	XImage *	NULL	SGI
XtNlabelJustify	OlDefine	OL_LEFT	SGI
XtNlabelTile	Boolean	FALSE	SGI
XtNlabelType	OlDefine	OL_STRING	SGI
XtNrecomputeSize	Boolean	TRUE	SGI
XtNselect	XtCallbackList	NULL	SGIO

XtNbusy

Class	Type	Default	Access
XtCBusy	Boolean	FALSE	SGI

Synopsis: The "busy" state of the button.

Values: `TRUE/"true"` – The button is stippled as "busy.". The system will beep if the user attempts to select the button; the attempt is refused and no callbacks are invoked.

`FALSE/"false"` – The button is not stippled and works normally.

XtNdefault

Class	Type	Default	Access
XtCDefault	Boolean	FALSE	SGI

Synopsis: Whether the OblongButton is the default choice in its immediate shell.

Values: `TRUE/"true"` – If the button is in a menu, an oval ring is drawn around the button to show that the button is the default choice of one or more buttons.
`FALSE/"false"` – The button is not the default choice.

XtNlabel

Class	Type	Default	Access
XtCLabel	OlStr	(instance name)	SGI

Synopsis: The text for the Label.

Values: Any `OlStr` value valid in the current locale.

This resource is ignored if the `XtNlabelType` resource has the value OL_IMAGE.

The OblongButton label is colored using the `XtNfontColor` resource.

XtNlabelImage

Class	Type	Default	Access
XtCLabelImage	XImage *	NULL	SGI

Synopsis: The image for the Label.

This resource is ignored unless the `XtNlabelType` resource has the value OL_IMAGE. If the image is of type `XYBitmap`, the image is highlighted when appropriate by reversing the 0 and 1 values of each pixel (that is, by XORing the image data). If the image is of type `XYPixmap` or `ZPixmap`, the image is not highlighted, although the space around the image inside the border is.

If the image is smaller than the space available for it inside the border and `XtNlabelTile` is FALSE, the image is centered vertically and either centered or left-justified horizontally, depending on the value of the `XtNlabelJustify` resource. If the image is larger than the space available for it, it is clipped so that it does not display outside the border. If the `XtNdefault` resource is TRUE so that the border is doubled, the space available is that inside the inner line of the border.

XtNlabelJustify

Class	Type	Default	Access
XtCLabelJustify	OlDefine	OL_LEFT	SGI

Synopsis: The justification of the Label within the widget width.
Values: `OL_LEFT/"left"` – The Label is left-justified.
`OL_CENTER/"center"` – The Label is centered.

XtNlabelTile

Class	Type	Default	Access
XtCLabelTile	Boolean	FALSE	SGI

Synopsis: The tiling of the Label's background.
Values: `TRUE/"true"` – For an image that is smaller than the subobject's background, the label area is tiled with the image to fill the subobject's background.
`FALSE/"false"` – The label is placed as described by the `XtNlabelJustify` resource.

This resource augments the `XtNlabelImage` resource to allow tiling the subobject's background. The `XtNlabelTile` resource is ignored for text labels.

XtNlabelType

Class	Type	Default	Access
XtCLabelType	OlDefine	OL_STRING	SGI

Synopsis: The form that the Label takes.
Values: `OL_STRING/"string"` – The label is text.
`OL_IMAGE/"image"` – The label is an image.
`OL_POPUP/"popup"` – The label is text followed by an ellipsis.

XtNrecomputeSize

Class	Type	Default	Access
XtCRecomputeSize	Boolean	TRUE	SGI

Synopsis: Whether the OblongButton widget should calculate its size.
Values: `TRUE/"true"` – The OblongButton widget will do normal size calculations that may cause its geometry to change, and automatically set the `XtNheight` and `XtNwidth` resources.
`FALSE/"false"` – The OblongButton widget will leave its size alone; this may cause truncation of the visible image being shown

by the OblongButton widget if the fixed size is too small, or may cause padding if the fixed size is too large. The location of the padding is determined by the `XtNlabelJustify` resource.

XtNselect

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the widget is selected.

Activation Types

The following table lists the activation types used by the OblongButton.

Table 8-44 OblongButton Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MENU	MENU	XtNmenuBtn
OL_MENUDEFAULT	MENUDEFAULT	XtNmenuDefaultBtn
OL_MENUDEFAULTKEY	MENUDEFAULT	XtNmenuDefaultKey
OL_MENUKEY	MENU	XtNmenuKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

Activation types not described in the following list are described in “Common Activation Types” on page 68.

OL_MENU/
OL_MENUKEY

The OblongButton only will respond to the OL_MENU and OL_MENUKEY activation types if it is a descendant of a Menu widget. When this is the case, the OL_MENU and OL_MENUKEY will behave as the OL_SELECT and OL_SELECTKEY, respectively.

OL_MENUDEFAULT/
OL_MENUDEFAULTKEY

The OL_MENUDEFAULT and OL_MENUDEFAULTKEY activation types apply only to OblongButtons that are descendants of a Menu. These activation types set the OblongButton `XtNdefault` resource to TRUE, and change the display of the widget according to the *OPEN LOOK GUI Functional Specification* section "Changing Menu Defaults" in Chapter 15.

OL_SELECT/
OL_SELECTKEY

The activation of the OblongButton widget with the SELECT button or key will cause the `XtNselect` callback to be called.

See Also

"Exclusives Widget" on page 277
"FlatExclusives Widget" on page 337
"FlatNonexclusives Widget" on page 347
"MenuButton Widget" on page 403
"Nonexclusives Widget" on page 428
"RectButton Widget" on page 489.

PopupWindowShell Widget

Class

Class Name: `PopupWindowShell`
Class Pointer: `popupWindowShellWidgetClass`

Ancestry

Core-Composite-Shell-WMShell-VendorShell-TransientShell-PopupWindowShell

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/PopupWindo.h>
```

Description

The PopupWindowShell widget is used to implement the OPEN LOOK property window, managing its creation, and providing a simple interface for populating the window with controls. However, the PopupWindowShell has no innate semantics to relate the controls to a selected object; this must be handled by the application. For example, the application must dim all the controls if an object selected by the user is incompatible with a displayed property window.

The decoration of the PopupWindowShell by the Window Manager is controlled by the resources listed in the following table.

Table 8-45 PopupWindowShell Default Window Decorations

Resource	Type	Default
XtNmenuButton	Boolean	FALSE
XtNpushpin	OlDefine	OL_OUT
XtNresizeCorners	Boolean	FALSE
XtNwindowHeader	Boolean	TRUE

Components

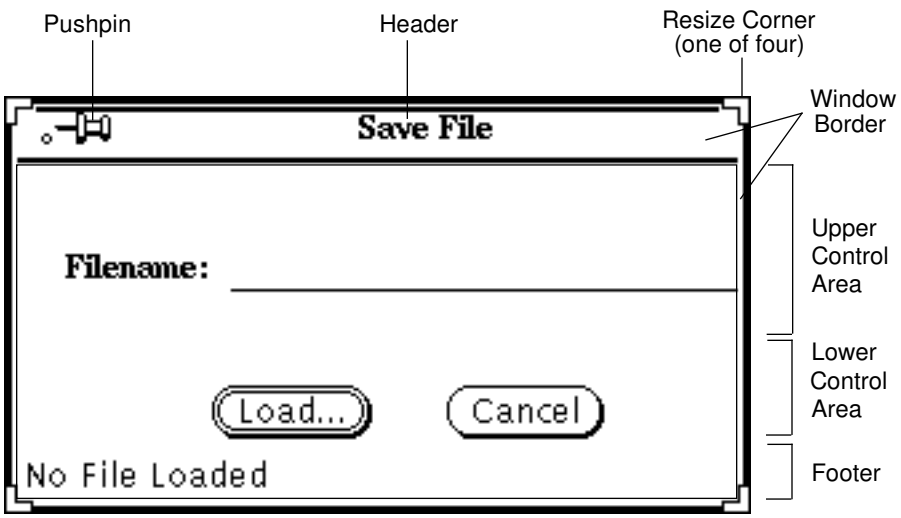

Figure 8-13 PopupWindowShell Components

The PopupWindowShell has the following parts, as shown in the figure.

- Upper control area
- Lower control area
- Window border
- Popup window menu
- Settings menu (conditional)
- Apply button (conditional)
- Reset button (conditional)
- Reset to Factory button (conditional)
- Set Defaults button (conditional)
- Header
- Window mark
- Pushpin (optional)
- Resize corners (optional)
- Footer (optional)

The window border, popup window menu, header, window mark, and pushpin provide the user with window management controls over the PopupWindowShell widget. The Apply, Reset, Reset to Factory, and Set

Defaults Buttons are automatically created by adding an appropriate callback to help create a standard layout of a property window.

Subwidgets

The PopupWindowShell contains three subwidgets, a FooterPanel and two ControlArea widgets, provided automatically, and accessible through the following resources:

- `XtNfooterPanel`
- `XtNlowerControlArea`
- `XtNupperControlArea`

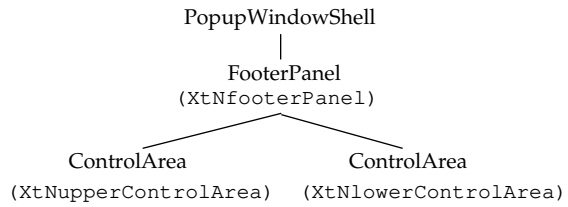

Figure 8-14 PopupWindowShell Subwidgets

Automatic Addition of Buttons, Settings Menu

To aid in the creation of a property window, the PopupWindowShell has several callbacks typically used in such a popup window, for applying, resetting, etc. For each of these callbacks that the application sets in the argument list used for creation of the PopupWindow, the PopupWindowShell widget automatically creates a button in the lower ControlArea, and the same button in the Settings Menu. For example, if the application specifies a callback for `XtNapply`, the popup displays an Apply button. If none of the callbacks is defined, no buttons are automatically created and no Settings Menu is created.

If the application is building a command window, it has to create whatever buttons and menus are needed.

Subclass Shell Widget

The PopupWindowShell widget is a subclass of `Shell`, so the `XtCreatePopupShell()` routine is used to create a popup window instead of the normal `XtCreateWidget()`.

Popping the Window Up and Down

The application controls when the PopupWindowShell widget is to be displayed or popped up with the `XtPopup()` routine.

The application also has the responsibility for raising a mapped popup window to the front if the user attempts to pop it up and it is already up. This can be accomplished using the `XRaiseWindow()` function.

However, the application cannot control when the PopupWindowShell widget is to be popped down, since the user may have pinned it up with the intent that it stays up until he or she dismisses it. The widget itself detects when to pop down: the user clicks SELECT on an OblongButton widget in the lower ControlArea, or the user dismisses the popup window using the Popup Window Menu or pushpin.

Control Areas

The upper and lower ControlAreas are handled by separate widget interfaces. The application needs to obtain the individual widget IDs for the control areas (*upper_controlarea* and *lower_controlarea*) and footer container (*footerarea*) from the PopupWindowShell widget.

The two ControlAreas and the Footer abut so that there is no space between them. An application can control the distance between the controls in the Control Areas by setting margins in each area.

If the PopupWindowShell widget automatically creates the Apply, Reset, Reset to Factory, or Set Defaults Buttons, it puts them in that order in the lower ControlArea. No space is left for a missing button. These buttons will also appear before any buttons added to the lower ControlArea by the application.

Coloration

The following diagram illustrates the resources that affect PopupWindowShell coloration.

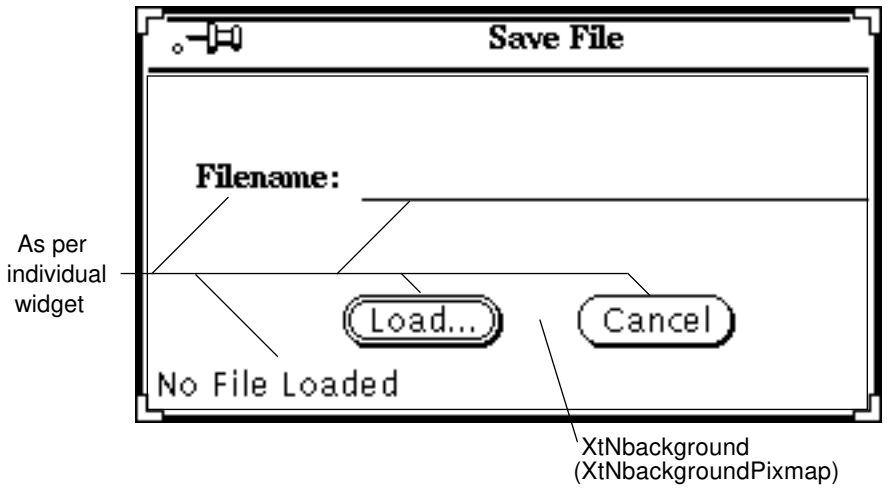

Figure 8-15 PopupWindowShell Coloration

Keyboard Traversal

The PopupWindowShell widget has a number of components that the user can traverse between. The buttons in the lower ControlArea and in the Settings menu have the following mnemonics:

Table 8-46 PopupWindowShell Traversable Components

Resource	Button Name	Mnemonic
XtNapply	Apply	A
XtNreset	Reset	R
XtNresetFactory	Reset to Factory	F
XtNsetDefaults	Set Defaults	S

These mnemonics will be displayed in the button labels according to the value of the toolkit resource `XtNshowMnemonics` (see page 16). The buttons are created with `XtNtraversalOn` set to TRUE.

The TOGGLEPUSHPIN key changes the state of the pushpin in the window header. If the pushpin is in, TOGGLEPUSHPIN will pull the pin out and pop down the window. If the pushpin is out, TOGGLEPUSHPIN will stick the pin in.

Resources

Table 8-47 PopupWindowShell Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	GI
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 8-48 PopupWindowShell Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 8-49 PopupWindowShell Shell Resources

Name	Type	Default	Access
XtNallowShellResize	Boolean	TRUE	SGI
XtNcreatePopupChildProc	XtCreatePopupChildProc	NULL	SGI
XtNgeometry	String	NULL	GI
XtNoverrideRedirect	Boolean	FALSE	SGI
XtNpopdownCallback	XtCallbackList	NULL	SGIO
XtNpopupCallback	XtCallbackList	NULL	SGIO
XtNsaveUnder	Boolean	FALSE	SGI
XtNvisual	Visual *	(parent's)	GIO

Table 8-50 PopupWindowShell WMShell Resources

Name	Type	Default	Access
XtNbaseHeight	int	XtUnspecifiedShellInt	SGI
XtNbaseWidth	int	XtUnspecifiedShellInt	SGI
XtNheightInc	int	XtUnspecifiedShellInt	SGI
XtNiconMask	Pixmap	NULL	SGI
XtNiconPixmap	Pixmap	NULL	SGI
XtNiconWindow	Window	NULL	SGI
XtNiconX	int	XtUnspecifiedShellInt	SGI
XtNiconY	int	XtUnspecifiedShellInt	SGI
XtNinitialState	InitialState	NormalState	SGI
XtNinput	Bool	FALSE	G
XtNmaxAspectX	int	XtUnspecifiedShellInt	SGI
XtNmaxAspectY	int	XtUnspecifiedShellInt	SGI
XtNmaxHeight	int	OL_IGNORE	SGI
XtNmaxWidth	int	OL_IGNORE	SGI
XtNminAspectX	int	XtUnspecifiedShellInt	SGI
XtNminAspectY	int	XtUnspecifiedShellInt	SGI
XtNminHeight	int	OL_IGNORE	SGI
XtNminWidth	int	OL_IGNORE	SGI

Table 8-50 PopupWindowShell WMShell Resources (Continued)

Name	Type	Default	Access
XtNtitle	String	NULL	SGI
XtNtitleEncoding	Atom	XA_STRING	SGI
XtNtransient	Boolean	TRUE	SGI
XtNwaitForWm	Boolean	TRUE	SGI
XtNwidthInc	int	XtUnspecifiedShellInt	SGI
XtNwindowGroup	Window	XtUnspecifiedWindow	SGI
XtNwinGravity	int	XtUnspecifiedShellInt	SGI
XtNwmTimeout	int	5000 (msec)	SGI

Table 8-51 PopupWindowShell VendorShell Resources

Name	Type	Default	Access
XtNbusy	Boolean	FALSE	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNdefaultImName	String	NULL	SGI
XtNfooterPresent	Boolean	FALSE	SGI
XtNfocusWidget	Widget	(see description)	SGI
XtNimFontSet	OlFont	XtDefaultFontSet	SGI
XtNimStatusStyle	OlImStatusStyle	OL_NO_STATUS	GI
XtNleftFooterString	OlStr	NULL	SGI
XtNleftFooterVisible	Boolean	TRUE	SGI
XtNmenuButton	Boolean	(see description)	GI
XtNmenuType	OlDefine	(see description)	SGI
XtNpushpin	OlDefine	(see description)	SGI
XtNresizeCorners	Boolean	(see description)	SGI
XtNrightFooterString	OlStr	NULL	SGI
XtNrightFooterVisible	Boolean	TRUE	SGI
XtNshellTitle	OlStr	NULL	SGI
XtNuserData	XtPointer	NULL	SGI
XtNwindowHeader	Boolean	(see description)	GI
XtNwmProtocol	XtCallbackList	NULL	SGIO
XtNwmProtocolInterested	int	OL_WM_DELETE_WINDOW | OL_WM_TAKE_FOCUS	I

Table 8-52 PopupWindowShell TransientShell Resources

Name	Type	Default	Access
XtNtransientFor	Widget	NULL	SGI

Table 8-53 PopupWindowShell Resources

Name	Type	Default	Access
XtNapply	XtCallbackList	NULL	I
XtNapplyLabel	String	“Apply”	GI
XtNapplyMnemonic	unsigned char	‘A’	GI
XtNfooterPanel	Widget	(none)	G
XtNlowerControlArea	Widget	(none)	G
XtNmenuTitle	String	“Settings”	GI
XtNpointerWarping	Boolean	TRUE	SGI
XtNreset	XtCallbackList	NULL	I
XtNresetFactory	XtCallbackList	NULL	I
XtNresetFactoryLabel	String	“Reset To Factory”	GI
XtNresetFactoryMnemonic	unsigned char	‘F’	GI
XtNresetLabel	String	“Reset”	GI
XtNresetMnemonic	unsigned char	‘R’	GI
XtNsetDefaults	XtCallbackList	NULL	I
XtNsetDefaultsLabel	String	“Set Defaults”	GI
XtNsetDefaultsMnemonic	unsigned char	‘S’	GI
XtNupperControlArea	Widget	(none)	G
XtNverify	XtCallbackList	NULL	I

The following table lists resources passed to the ControlArea subwidget maintained by the PopupWindowShell; they become resources of the ControlArea subwidget and can be set and read just like any other PopupWindowShell resources.

Table 8-54 PopupWindowShell ControlArea Subwidget Resources[1]

Name	Type	Default	Access
XtNalignCaptions	Boolean	TRUE for upper Control Area; FALSE for lower	I
XtNcenter	Boolean	FALSE	I
XtNhPad	Dimension	0	I
XtNhSpace	Dimension	0	I
XtNlayoutType	OlDefine	OL_FIXEDCOLS for upper Control Area; OL_FIXEDROWS for lower	I
XtNmeasure	int	1	I
XtNsameSize	OlDefine	OL_COLUMNS	I
XtNvPad	Dimension	0	I
XtNvSpace	Dimension	0	I

1. These subwidget resource are defined in the section "ControlArea Widget" on page 249.

XtNapply/ XtNreset/ XtNresetFactory/ XtNsetDefaults

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	I

Synopsis: The callback lists invoked for buttons in the lower ControlArea and in the Settings Menu—the Apply, Reset, Reset to Factory, and Set Defaults buttons, respectively.

Typically, an application defines one of these resources only when it is using the PopupWindowShell widget for a property window. For each resource with a defined callback, a unique button is added in the lower ControlArea; conversely, where a resource has no callback defined by the application, no button is shown. The callbacks must be set at initialization time in order for the buttons to be created. The labels for these buttons are listed below, in the order they appear in the lower ControlArea.

No space is left for a missing button. In general, the callback list for one of these resources is issued when the user activates the button associated with the resource. After the callbacks are issued, the PopupWindowShell widget will attempt to pop itself down, first checking with the application that this may be done by issuing the `XtNverify` callbacks, then checking the state of the pushpin.

XtNapplyLabel/ XtNresetLabel/ XtNresetFactoryLabel/ XtNsetDefaultsLabel

Class	Type	Default	Access
XtCApplyLabel	String	"Apply"	GI
XtCResetLabel	String	"Reset"	GI
XtCResetFactoryLabel	String	"Reset to Factory"	GI
XtCSetDefaultsLabel	String	"Set Defaults"	GI

Synopsis: The labels for the Apply, Reset, Reset to Factory, and Set Defaults buttons, respectively.

XtNapplyMnemonic/ XtNresetMnemonic/ XtNresetFactoryMnemonic/ XtNsetDefaultsMnemonic

Class	Type	Default	Access
XtCApplyMnemonic	unsigned char	'A'	GI
XtCResetMnemonic	unsigned char	'R'	GI
XtCResetFactoryMnemonic	unsigned char	'F'	GI
XtCSetDefaultsMnemonic	unsigned char	'S'	GI

Synopsis: The mnemonics for the Apply, Reset, Reset to Factory, and Set Defaults buttons, respectively.
Values: Any ASCII character.

XtNfooterPanel

Class	Type	Default	Access
XtCFooterPanel	Widget	(none)	G

Synopsis: The widget ID of the FooterPanel composite child widget that handles the footer.

The value of this resource is available once the PopupWindowShell widget has been created. If the application wants a footer, it can add one to the composite identified by this resource.

XtNlowerControlArea/ XtNupperControlArea

Class	Type	Default	Access
XtCLowerControlArea	Widget	(none)	G
XtCUpperControlArea	Widget	(none)	G

Synopsis: The widget IDs of the ControlArea composite child widgets that handle the lower and upper ControlAreas, respectively.

The application can use each widget ID to populate the PopupWindowShell with controls. These widget IDs are available once the PopupWindowShell widget has been created. Any widgets of the class OblongButton added to the lower ControlArea are assumed to be window disposition controls; that is, when the user activates one of them the PopupWindowShell widget should pop itself down, if allowed by the application and the state of the pushpin.

XtNmenuTitle

Class	Type	Default	Access
XtCMenuTitle	String	"Settings"	GI

Synopsis: The title for the conditional Settings Menu.

XtNpointerWarping

Class	Type	Default	Access
XtCPointerWarping	Boolean	TRUE	SGI

Synopsis: Whether the pointer will jump to the focus widget when the PopupWindowShell is popped up.

Values: `TRUE/"true"` – The pointer jumps to the focus widget (see "XtNfocusWidget" on page 44) when the widget is popped up.
`FALSE/"false"` – The pointer doesn't jump.

XtNverify

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	I

Synopsis: The callback list invoked when popping down.

The *call_data* parameter to the callback is a pointer to a variable of type `Boolean`. It is initially set to TRUE, and the application should set a value that reflects whether the pop-down is allowed. Typically, the application will use this to prevent a pop-down so that an error message can be displayed. Since more than one callback routine may be registered for this resource, each callback routine can first check the value pointed to by the *call_data* parameter to see if a previous callback in the list has already rejected the pop-down attempt. If one has, the subsequent callback need not continue evaluating whether a pop-down is allowed. If the value is still TRUE after the last callback returns, the pop-down continues. Since these callbacks are issued before the PopupWindowShell checks the state of the pushpin, the application should not assume that the pop-down will occur even though it has allowed it.

Activation Types

The following table lists the activation types used by the PopupWindowShell.

Table 8-55 PopupWindowShell Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

The PopupWindowShell widget has no activation types besides the ones in "Common Activation Types" on page 68.

See Also

"ControlArea Widget" on page 249,
"FooterPanel Widget" on page 381.

8

PopupWindowShell Widget