Basic Controls

Controls are widgets and gadgets with which the user interacts directly. They form the leaves of the widget tree whose root is the application's top−level shell. In most cases, controls are subclasses of **XmPrimitive** or **XmGadget**, and their parents are subclasses of **XmManager**. (**XmScale** is a manager, but in many ways the application treats it as a primitive.) Motif provides the following basic controls:

- Labels, buttons, and separators
- ScrollBar
- Scale
- List
- Text and TextField

Core, RectObj, XmPrimitive, and XmGadget Classes

Nearly all the basic controls are subclasses of **XmPrimitive** or **XmGadget**. **XmPrimitive**, in turn, is a subclass of the Intrinsics **Core** class, and **XmGadget** is a subclass of the Intrinsics **RectObj** class.

Core

The **Core** class provides basic attributes of all widgets that have associated windows. It has the following groups of resources:

- Specifications of the widget's x and y coordinates, width and height, and border width
- A resource specifying whether or not the widget is sensitive or able to receive input events from the Intrinsics event manager
- Characteristics of the window, including background and border color or pixmap, colormap, depth, and screen
- A resource controlling whether or not the Intrinsics map the window when the widget is managed
- A table associating translations with actions
- A set of accelerators, which is a translation table bound in the context of a particular widget

RectObj

RectObj is the foundation for gadget classes; it is essentially **Core** without the attributes related to having a window. **RectObj** resources control the position and dimensions of the gadget's rectangular area within its parent widget. A **RectObj** resource also determines whether or not the gadget is sensitive.

XmPrimitive

XmPrimitive is the fundamental Motif class for all basic control widgets−−widgets that do not have children. It includes the following resources and behavior:

- Foreground color, top and bottom shadow colors or pixmaps, and shadow thickness
- Thickness and color or pixmap for the highlighting rectangle, which is displayed when the widget has keyboard focus
- Resources to determine whether the user can traverse to the widget and whether or not it is a tab group
- A resource to determine what unit of measurement the widget uses for size and position resources
- Callbacks for the widget to invoke when the user presses `osfHelp`
- A resource for the application to use in associating arbitrary data with the widget
- Translations and actions for keyboard traversal to another widget
- A resource that specifies the direction in which components of the primitive (including strings) are laid out.
- Callbacks that allow the application to control which popup menu will be automatically posted.

XmGadget

XmGadget is the fundamental Motif class for all basic control gadgets. **XmGadget** is equivalent to **XmPrimitive**, with two major exceptions:

- It has no associated window.
- It has no translations or actions. The Manager parent controls traversal between its gadget children, keeps track of gadgets that have input focus, and dispatches events to them.

Labels, Buttons, and Separators

Labels, buttons, and separators are simple widgets built on **XmPrimitive**.

Labels

Labels provide the ability to display static (uneditable) text or a pixmap. Your application can use a Label or LabelGadget to display a message, title, or description. Label and LabelGadget also serve as superclasses for button widgets and gadgets. However, unlike button widgets, Label provides no additional callbacks beyond those that it inherits from Primitive.

The application can specify the following characteristics of Labels, LabelGadgets, and their subclasses:

- The compound string or pixmap to be displayed. When using a pixmap, the application can supply a separate pixmap to be displayed when the widget is insensitive.
- The render table that describes the attributes of the compound string.
- The positioning of the text or pixmap within the widget. One set of resources determines the space allocated for the margins; another determines the distance between the margins and the text or pixmap inside. The **XmNalignment** and **XmNlayoutDirection** resources together determine whether the text or pixmap is centered or is left or right justified within the widget.
- A resource, **XmNrecomputeSize**, that determines whether the widget attempts to remain large enough to contain the text or pixmap. When this resource is True and a resource that affects the size of the text or pixmap, the margins, or the widget itself is changed, the widget tries to resize itself to be just large enough to contain the text or pixmap.

In addition, Label and LabelGadget provide the following facilities for button subclasses in menus:

- A keysym used as a mnemonic to select the button. The user can activate the button by pressing the mnemonic key when the button is visible.
- An accelerator, a `KeyPress` event by which the user can activate the button whether or not it is visible. Accelerators are supported only for PushButtons and ToggleButtons in PulldownMenus and PopupMenus.
- Translations and actions for keyboard traversal within the menu or menu system.

A Label or LabelGadget can be the source of a drag and drop operation, but cannot be the destination. In other words, you can drag a value from a Label or LabelGadget, but you cannot drop a value into a Label or LabelGadget. The **XmNenableUnselectableDrag** resource of **XmDisplay** holds a Boolean value. If this value is True, then users can drag from a Label or LabelGadget. If this value is False, then users cannot drag from a Label or LabelGadget.

Buttons

A button is a basic control that performs some action when the user activates it. Buttons commonly appear in menus, RadioBoxes, CheckBoxes, SelectionBoxes, and MessageBoxes. This section describes some of the functions of each subclass.

CascadeButtons

A CascadeButton or CascadeButtonGadget is used inside a menu and, when activated, usually causes a PulldownMenu to appear.
CascadeButtons have the following resources and behavior:

- A pixmap displayed at one end of the widget in a PopupMenu or PulldownMenu to indicate that activating the CascadeButton posts another menu.
- A resource, **XmNsubMenuId**, that holds the widget ID of the PulldownMenu posted when the user activates the button.
- **XmNactivateCallback** callbacks, which the widget invokes when the user activates it, and **XmNcascadingCallback** callbacks, which the widget invokes just before posting a PulldownMenu.
- A resource to provide a delay between the time the mouse enters the widget and the time it posts a menu.
- Translations and actions to activate the widget and to post and unpost PulldownMenus. In general, pressing `Btn1` or dragging `Btn1` into the widget posts the PulldownMenu. Releasing `Btn1` in the widget causes the PulldownMenu to remain posted and enables keyboard traversal. When keyboard traversal is enabled, pressing `osfActivate` or `osfSelect` in the widget posts the PulldownMenu and enables keyboard traversal in that menu.

PushButtons

A PushButton or PushButtonGadget can appear either inside or outside a menu. It performs some action determined by the application. When a PushButton is armed, or ready to be activated, it changes its appearance so that it looks as if the user has pressed it in. When it is disarmed it reverts to the appearance of extending out. PushButtons provide the following behavior:

- Callbacks that the widget invokes when it is armed, disarmed, and activated. The application usually provides only an **XmNactivateCallback** procedure to perform the action associated with the button.

- Resources to provide a color or pixmap to be displayed when the button is armed and not inside a menu. When a button in a menu is armed, the top and bottom shadows switch colors.
- A resource to determine whether or not the widget considers multiple mouse clicks distinct from single mouse clicks.
- A resource to determine whether or not the button is marked as the default button when outside a menu. In a BulletinBoard, the default button is the one activated when the user presses `osfActivate` and no other button has keyboard focus. The default button has a distinctive shadow whose thickness is controlled by the **XmNdefaultButtonShadowThickness** resource and whose appearance is controlled by the **XmNdefaultButtonEmphasis** resource of **XmDisplay**.
- Translations to arm, disarm, and activate the button. In general, a button is activated upon pressing `Btn1` while on a button, or dragging `Btn1` or traversing to a button while in menu. Releasing `Btn1` or pressing `osfActivate` (in a menu) or `osfSelect` (in the widget) activates and disarms the button.

ToggleButtons

ToggleButtons and ToggleButtonGadgets are typically in one of two states: they are either "set" or "unset." In addition, ToggleButtons and ToggleButtonGadgets can also be in an indeterminate state; that is, neither set nor unset. They can appear in menus or in nonmenu RowColumn WorkAreas, including RadioBoxes and CheckBoxes. In a RadioBox only one ToggleButton at a time can be on; in a CheckBox more than one ToggleButton can be on. ToggleButtons can have indicators with distinctive shapes to distinguish whether or not more than one button at a time can be set. However, it is the RowColumn parent, not the ToggleButton, that controls this behavior.

ToggleButtons have the following characteristics:

- Callbacks that the widget invokes when it is armed or disarmed and when it changes state. The widget invokes the **XmNvalueChangedCallback** callbacks when the button's state changes from set to unset or from unset to set.
- Resources to control the appearance of the indicator. If **XmNindicatorOn** is False or if **XmNvisibleWhenOff** is False and the button is in the unset state no indicator is displayed. Otherwise, **XmNindicatorType** determines whether the indicator shows that only one or more than one button at a time can be set.
- A color to be displayed when the button is armed and **XmNfillOnSelect** is True and another color to be displayed when the button is not set.
- Pixmaps to be displayed when the button is selected and the Label or LabelGadget superclass's **XmNlabelType** is **XmPIXMAP**.
- Translations to arm and disarm the button and to change its state. In general, a button is activated upon pressing `Btn1` while on a button, or dragging `Btn1` or traversing to a button while in menu. Releasing `Btn1` or pressing `osfActivate` (in a menu) or `osfSelect` (in the widget) activates and disarms the button.

DrawnButtons

A DrawnButton is an empty button surrounded by a shadow border. It is intended to be used as a PushButton but with graphics drawn by the application. Like a PushButton, it has translations and actions to arm, disarm, and activate the button and invoke the corresponding callbacks. If **XmNpushButtonEnabled** is True, it draws the shadow so that the button appears pressed in when armed and popped out when disarmed.

Other than this, the application must manage the button's visual appearance. It has **XmNexposeCallback** and **XmNresizeCallback** callbacks to notify the application that the button has been exposed or resized and therefore needs to be redrawn. The application must be careful not to draw within the button's shadows or highlight areas. The application can use a clipping rectangle in the widget's graphics context that takes account of the button's **XmNhighlightThickness** and **XmNshadowThickness**.

ArrowButtons

An ArrowButton or ArrowButtonGadget is a button with an arrow graphic and a shadow. A resource controls the direction of the arrow. Unlike other buttons, it is not a subclass of **XmLabel** or **XmLabelGadget**, but it has some of the same behavior as other buttons. It has callbacks that the widget invokes when armed, disarmed, or activated. It has translations and actions similar to those of other buttons to arm, disarm, or activate the button.

Separators

A Separator or SeparatorGadget separates controls or groups of controls. It usually appears as a horizontal or vertical line and supports several styles of line drawing. Resources control its orientation and the type of line it draws. One line style consists of no line at all. This allows the application to control the appearance of the separator by setting its **XmNbackgroundColor** or **XmNbackgroundPixmap**.

ScrollBar

A widget can act as a viewport onto a virtual scroll. The ScrollBar is the control that moves the viewport horizontally or vertically relative to the underlying scroll. A ScrollBar consists of a rectangle, called the scroll region, representing the full size of the scroll. It has a smaller rectangle, called the slider, within the scroll region, representing the position and size of the viewport relative to the full scroll. The ScrollBar usually has arrow graphics at both ends of the larger rectangle.

A ScrollBar has translations and actions that allow the user to move the slider. By clicking on an arrow, the user moves the slider one small increment in the direction of the arrow. By clicking in the scroll region between an arrow and the slider, the user moves the slider a larger increment (the page increment) in the direction of the arrow. When the ScrollBar has keyboard focus, the user can use the keyboard to move the slider in this way. The user can also drag the slider using the mouse.

By itself, the ScrollBar does not have an association with a widget acting as a viewport onto a scroll. Most applications use a ScrolledWindow, a Manager widget with a child to be scrolled and possibly with one or two ScrollBars to control the scrolling. ScrolledWindow can automatically control the interaction between the scrolled child and the ScrollBars, or it can allow the application to control the interaction. For more information see Chapter 8.

ScrollBar has a number of resources that allow the application to use it to control scrolling:

- A minimum value (**XmNminimum**), representing the position of the slider at one end of the scroll region, and a maximum value (**XmNmaximum**), representing the position of the slider at the other end of the scroll region. These values can be in any integral units the application chooses, so long as the maximum is greater than the minimum.
- The length of the slider (**XmNsliderSize**) between 1 and (**XmNmaximum** - **XmNminimum**).
- A value (**XmNvalue**), ranging between **XmNminimum** and (**XmNmaximum** - **XmNsliderSize**), representing the current position of the slider between the maximum and minimum values.
- Values for the increment (**XmNincrement**) and page increment (**XmNpageIncrement**) by which the user can move the slider.
- A resource (**XmNprocessingDirection**) that determines whether the minimum value is on the left or right for horizontal ScrollBars or is on the bottom or top for vertical ScrollBars.
- Distinct callbacks that the widget invokes when the user moves the slider by one increment in either direction, by one page increment in either direction, or all the way to either end of the scroll region. The widget invokes other callbacks as the user drags the slider and when the user stops dragging the slider. The application does not have to provide routines for all these callback lists; if it provides only an **XmNvalueChangedCallback** procedure, the widget invokes that procedure whenever the ScrollBar value changes (except during interactive dragging of the slider).
- Resources to control the color of the scroll region, whether the ScrollBar is horizontal or vertical, and whether or not the ScrollBar has arrows.
- Resources to control the delays before the widget moves the slider continuously as the user presses and holds `Btn1` on an arrow or the scroll region.
- A resource (**XmNsnapBackMultiple** to specify the distance over which the ScrollBar slider snaps back to its original position when the user drags the mouse outside the ScrollBar edge.
- Resources to specify the appearance of the slider (**XmNsliderMark** and **XmNsliderVisual**).
- A resource (**XmNslidingMode**) to specify the mode the slider works in. For example, this resource could make the ScrollBar look like an old −fashioned mercury thermometer (**XmTHERMOMETER** mode) or like a carpenter's level (**XmSLIDER** mode).

Two convenience routines, **XmScrollBarGetValues** and **XmScrollBarSetValues**, allow the application to get and set the value, slider size, increment, and page increment in one call.

Scale

A Scale graphically displays a value between two end points. Its appearance and behavior are much like those of a ScrollBar without arrows. However, unlike a ScrollBar, a Scale can optionally display a title. In addition, a Scale can optionally display a textual version of its value to supplement the graphic display of its value.

Like a ScrollBar, a Scale has a minimum, a maximum, and a current value. The current value must be between between the minimum and maximum, inclusive.

Scales can either be editable (the default) or noneditable. The **XmNeditable** resource controls this attribute. Once set, a user cannot change the value in a noneditable Scale. An editable Scale, on the other hand, allows the user to supply a new value.

The increment by which the arrow keys move the slider is always 1, but the application can supply a multiple increment (**XmNscaleMultiple**) analogous to ScrollBar's **XmNpageIncrement**. Scale has two callback lists: **XmNvalueChangedCallback** is invoked when the user changes the value but is not in the process of dragging the slider, and **XmNdragCallback** is invoked when the user changes the value while dragging the slider.

Your application cannot control the size of the slider.

Scale has resources controlling whether the orientation is vertical or horizontal and which end of the Scale represents the minimum value. Other resources control aspects of the Scale's appearance, including the width and height, the title string, whether or not the Scale displays the current value next to the slider, the number of decimal places in the displayed value, and a render table for the title and value.

Drag and Drop in Scale

A Scale can be the source of a drag and drop operation, but it cannot be the destination. In other words, you can drag a value from a Scale, but you cannot drop a value into a Scale. The **XmNenableUnselectableDrag** resource of **XmDisplay** holds a Boolean value. If this value is True, then users can drag from a Scale. If this value is False, then users cannot drag from a Scale.

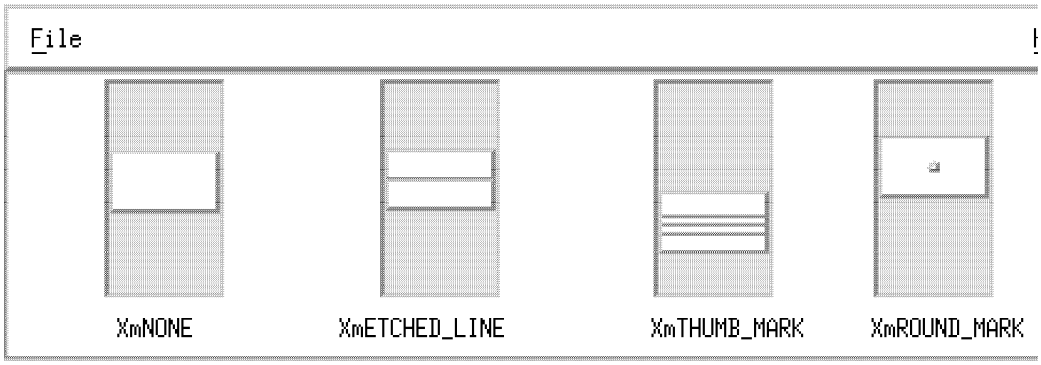
The only part of the Scale that a user can drag is the Scale's current value. More precisely, a user can drag a textual version of the Scale's current value, not the graphic version. The textual version can only be dragged if **XmNshowValue** is True.

Visuals Inside a Scale

The **XmNslidingMode** resource controls the anchoring of the slider. There are two possibilities. One possibility (**XmSLIDER**) is that the slider floats freely inside the Scale. Another possibility (**XmTHERMOMETER**) is that the slider appears tethered to one side of the Scale. The aptly named **XmTHERMOMETER** mode could allow your Scale widget to emulate an old−fashioned rising column of Mercury thermometer or barometer.

The **XmNsliderMark** resource controls the graphic that appears inside the slider. Figure 5−1 illustrates all the available slider marks.

Figure 1 Different Slider Marks Inside a Scale



Two convenience routines, **XmScaleGetValue** and **XmScaleSetValue**, allow the application to get and set the slider value.

Tic Marks

By default, a Scale has no labels or tic marks along the rectangle in which the slider moves. The application can add these by calling **XmScaleSetTicks**.

List

A List is an array of textual items from which the user selects one or more entries. Each item is a compound string. List has four modes, controlled by the **XmNselectionPolicy** resource, for selecting items:

Single Select

> At most one item is selected. Performing the selection action on an item toggles the selection state of the item and deselects any other selected item.

Browse Select

> At most one item is selected. Performing the selection action on an item selects the item and deselects any other selected item. Dragging `Btn1` through the list moves the selection along with the cursor.

Multiple Select

> Any number of items can be selected. Performing the selection action on an item toggles the selection state of the item but does not deselect any other selected item.

Extended Select

> Any number of items can be selected. The user can select either continuous or discontinuous ranges of items, depending on the mouse buttons used or, when using the keyboard, on whether the List is in Normal Mode or Add Mode.

When the user makes a selection, the List invokes one of four callback lists, depending on the selection policy:

Selection Policy	**Callback List**
Single Select	**XmNsingleSelectionCallback**
Browse Select	**XmNbrowseSelectionCallback**
Multiple Select	**XmNmultipleSelectionCallback**
Extended Select	**XmNextendedSelectionCallback**

By default, the List does not invoke a callback list when the List is in Browse Select or Extended Select mode and the user drags the mouse cursor over a new item. It does invoke the callbacks when the user releases the mouse button. If **XmNautomaticSelection** is True, the List invokes the callbacks while the user is dragging the mouse.

The widget data passed to selection callback routines contains both the selected items−−the compound strings−−and integers representing the positions within the list of the selected items. The first item in the list is at position 1, the second item at position 2, and so on.

List has another callback list, **XmNdefaultActionCallback**, which it invokes when the user double clicks or presses `osfActivate` on an item. The widget data passed to these callback routines contains only the item at the location cursor and its position, not the selected items. When the user performs the default action via a double click, the List calls the appropriate selection callbacks on the first click and the **XmNdefaultActionCallback** callbacks on the second click.

List includes several other sets of resources:

- Arrays and counts of the List items and selected items
- The number of items, **XmNvisibleItemCount**, that the list can display at one time, and the position in the List of the first visible item
- Several resources that affect the appearance of the list items: render table, justification (**XmNstringDirection**), spacing between items, and margins between the items and the List border
- The maximum time interval between clicks for a double click
- A resource (**XmNlistSizePolicy**) that determines what the List does when an item is too wide to fit into the List: it can keep its size and, if it is a ScrolledList, add a horizontal ScrollBar; grow to accommodate the item; or try to grow and, if it fails to accommodate the item but is a ScrolledList, add a ScrollBar
- A resource that determines whether the ScrollBars in a ScrolledList are displayed at all times or only when needed

A ScrolledList is a List inside a ScrolledWindow. The application can use **XmCreateScrolledList** to create one.

In addition to its resources, List has a variety of convenience routines that allow the application to add, remove, select, and deselect items; specify

the first or last visible item; find the position of an item or the positions of the selected items; set Add Mode; and scroll the List horizontally.

Text and TextField

The Text widget displays and, optionally, edits text. When the widget is editable and the user presses a key that represents a text character, that character is inserted into the text. Other translations and actions allow the user to navigate or to select, cut, copy, paste, or scroll the text.

The Text widget is used to edit simple text, which can only use a single font or fontset. The TextField widget is a text editor optimized for editing a single line of text. It also only uses simple text. This widget is largely used for entering commands and filenames.

For more information on Text and TextField, see Chapter 10.