HP OSF/Motif Programmer's Guide

HP 9000 Series 300/800 Computers

HP Part Number 98794-90005

Hewlett-Packard Company
1000 NE Circle Blvd., Corvallis OR 97330

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Certification of conformance with the OSF/Motif user environment is pending.

OSF/Motif is a trademark of the Open Software Foundation, Inc.

UNIX is a trademark of AT&T.

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

September 1989 ..... Edition 1

Contents

Introduction 1

The *HP OSF/Motif Programmer's Guide* contains information that you will need to create programs with the OSF/Motif widget set. Other manuals in the OSF/Motif documentation set provide information on configuration, standards, and reference data.

1.1 What's in this Guide

This guide contains the following chapters:

TABLE 1-1. Chapter Overview

Chapter	Title	Description
1	Introduction	Provides introductory information on the OSF/Motif system.
2	Widgets, Gadgets, and Convenience Functions	Describes of each widget, gadget, and convenience function in the OSF/Motif system.
3	Using OSF/Motif Widgets in Programs	Describes how to program using the OSF/Motif widget set. Two sample programs are presented.
4	Shell Widgets	Describes the various Shell widgets and how they are used.
5	Dialog Widgets and Functions	Describes Dialog widgets and their associated convenience functions.
6	Menus	Describes each of the various types of menus and presents a sample program for each menu type.
7	Specialized Widgets	Describes in detail three widgets that are considered specialized: List, RowColumn, and Text.

TABLE 1-1. Chapter Overview (Continued)

Chapter	Title	Description
8	Additional Functionality	Describes a number of additional features of the OSF/Motif system.
9	Keyboard Interface	Describes how to use the keyboard for traversal between widgets.
10	Debugging	Describes basic debugging procedures for some typical errors.
11	Glossary	Defines a number of words and phrases commonly used in the OSF/Motif system.

You should also be aware of these conventions:

- All OSF/Motif functions begin with "Xm." For example, `XmCreateForm` is the function to create a Form widget.
- All OSF/Motif widget resource names begin with "XmN." For example, `XmNbottomShadowColor` is a Manager resource that is used to specify the color to use when drawing the bottom and right sides of a window's border shadow. By convention, the first letter after the "N" is lower case but the first letter of succeeding new words in the resource name are capitalized, even though there are no spaces.
- All OSF/Motif widget class names begin with "xm." For example, the class name for the MainWindow widget is `xmMainWindowWidgetClass`. The first letters of succeeding words are capitalized, even though there are no spaces.
- All Xt Intrinsics functions begin with "`Xt`." For example, `XtCreateManagedWidget`" is an Xt Intrinsics function that creates and manages a widget.
- All Resource Manager functions (see *Programming With Xlib*) begin with "`Xrm.`" For example, `XrmInitialize` is a function that initializes the Resource Manager.
- All other X Window System functions begin with "`X.`" For example, `XCreateWindow` is a function that creates an unmapped window and sets its attributes. Most of these functions are described in *Programming With Xlib*.

1.2 The X Window System

The HP OSF/Motif widget set is based on the Xt Intrinsics, a set of functions and procedures that provide quick and easy access to the lower levels of the X Window system.

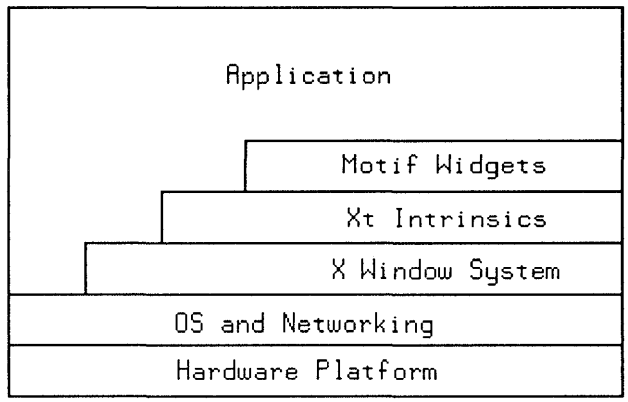


Figure 1-1. User Interface Development Model

You can see from figure 1-1 that the OSF/Motif widget set is layered on top of the Xt Intrinsics, which in turn are layered on top of the X Window System, thus extending the basic abstractions provided by X. The OSF/Motif widget set supports independent development of new or extended widgets. The OSF/Motif widget set consists of a number of different widgets, each of which can be used independently or in combination to aid in creating complex applications. Applications can be written faster and with fewer lines of code using the OSF/Motif widgets; however, they will require more memory than similar applications written without using these widgets.

This manual will explain the individual widgets and show you how to create and use these widgets in your applications.

1.3 Widget Classes and Hierarchy

Every widget is dynamically allocated and contains state information. Every widget belongs to one class, and each class has a structure that is statically allocated and initialized and contains operations for that class. Figure 1-2 shows the basic widget classes.

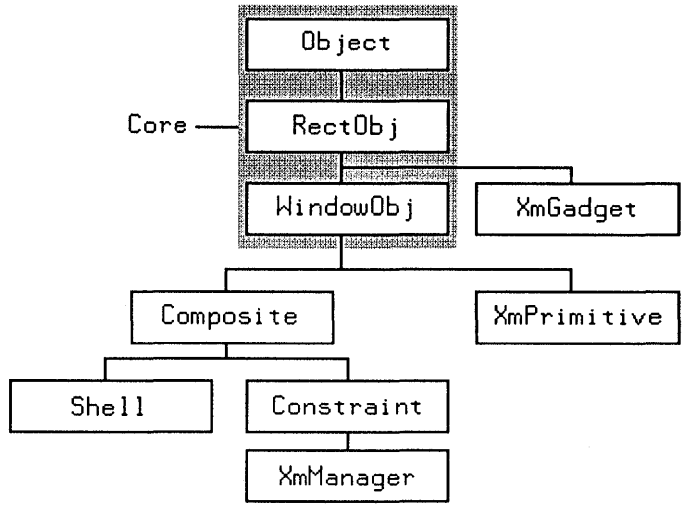


Figure 1-2. Basic widget Class Hierarchy

The basic class is the Core class. It contains resources that are inherited by all other classes. Two classes are layered beneath the Core class, the Composite class and the Primitive class. The Primitive class has no other classes beneath it, but the Composite class has two — the Constraint class and the Shell class. Each lower class can inherit some or all of the resources belonging to a higher class. For example, a Manager class widget can inherit some or all of the resources belonging to the Constraint class, the Composite class, and the Core class. You can find exactly what resources a given widget has by examining its man page in the *HP OSF/Motif Programmer's Reference Manual.*

This section has a number of hierarchy diagrams to help you understand how the widgets relate to each other. Figure 1-2 shows the highest level of widget classes. You can see that the Core class is composed of Object, RectObj, and WindowObj. Core is the base class for all other widget classes.

Figure 1-3 shows the subclasses of the Primitive class.

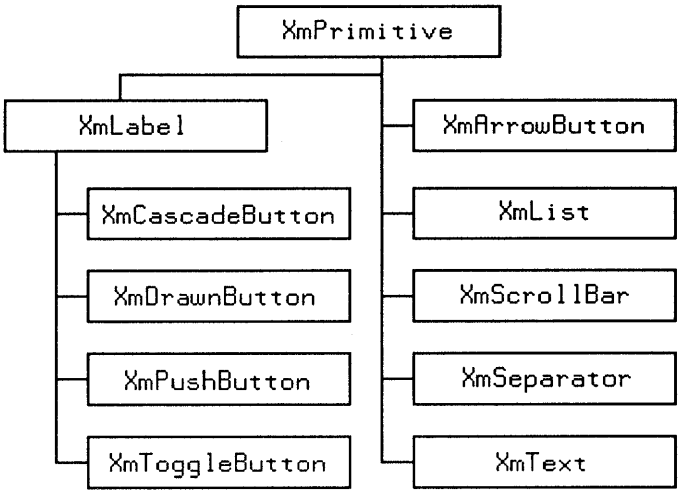


Figure 1-3. Primitive Class Widgets

Figure 1-4 shows the subclasses of the Shell class.

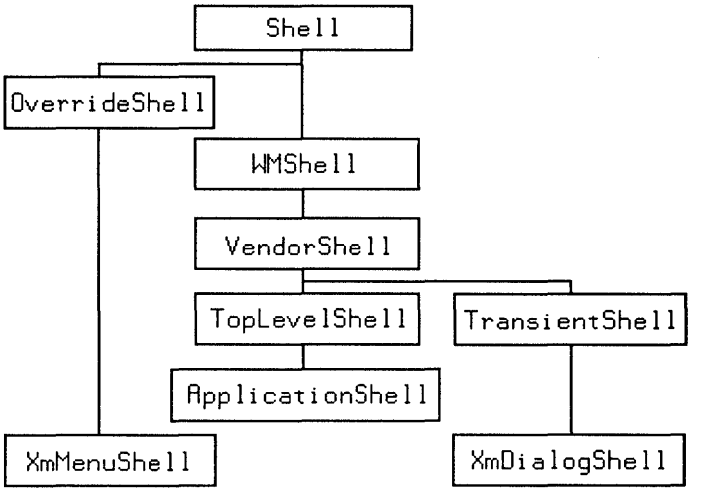


Figure 1-4. Shell Widgets

Figure 1-5 shows the Manager class widgets. Note from figure 1-2 that Manager is a subclass of Composite and Constraint.

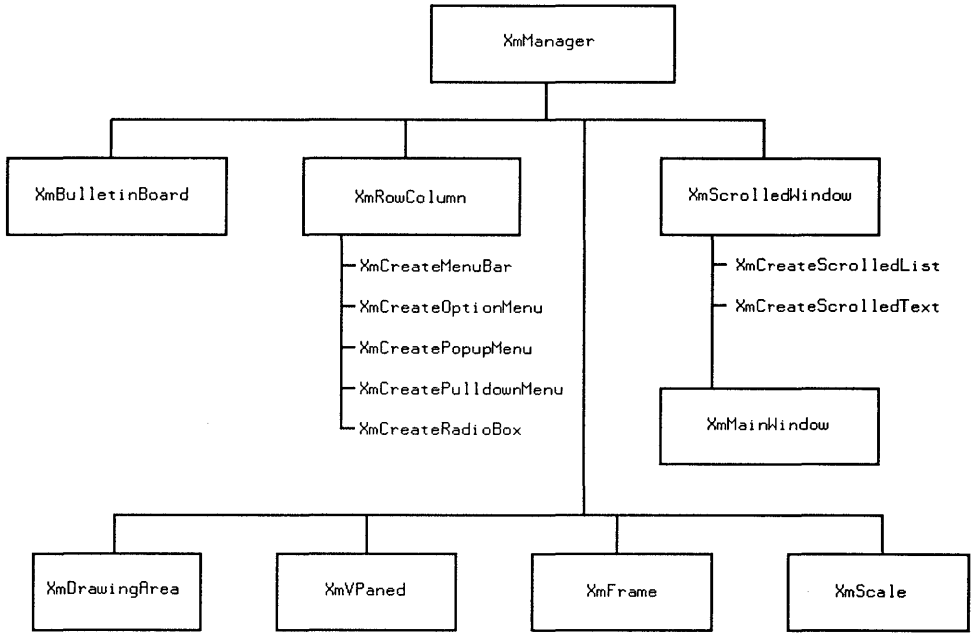


Figure 1-5. Manager Widgets

Figure 1-6 shows the Dialog widgets that are a subclass of Manager. Note that all of the Dialog widgets are subclasses of BulletinBoard. Also, note the convenience functions that are present. These are explained in detail in chapter 5, "Dialog Widgets and Functions."

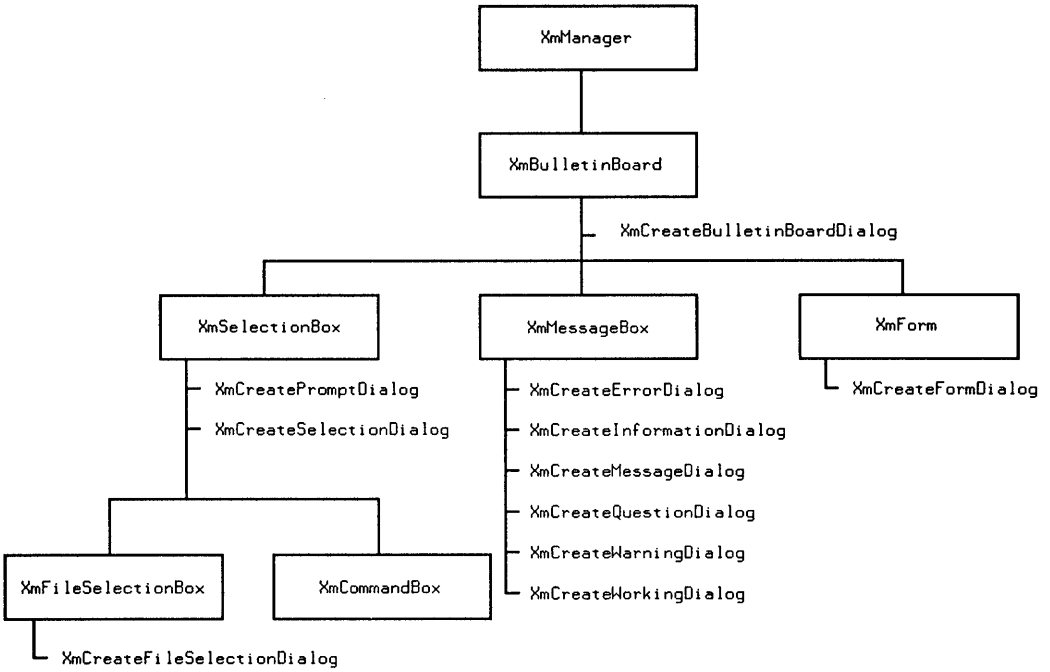


Figure 1-6. Dialog Widgets

Figure 1-7 shows the Gadgets that are an integral part of the OSF/Motif toolkit.

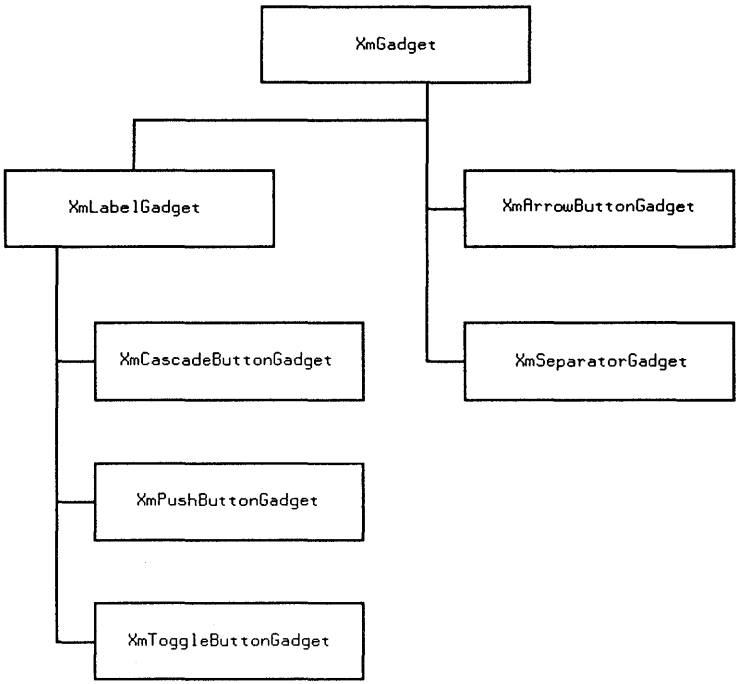


Figure 1-7. Gadgets

1.4 Compiling Sample Programs

There are a number of sample programs discussed throughout this manual. The source code for most of these programs can be found in the directory `/usr/contrib/Xm`. There is also a Makefile in this directory that you can use to compile and link the programs. Follow this procedure to compile and link a program.

1. Copy the program source code file and the Makefile found in `/usr/contrib/Xm` to your work directory. *Do not* attempt to compile the program in the `/usr/contrib/Xm` directory.
2. Compile the program by executing the following command:

   ```
   make <programname>
   ```

3. There is a defaults file called `XMdemos` in the directory `/usr/contrib/Xm` that you will need to move (or copy) to the directory `/usr/lib/X11/app-defaults` before you run the program. This defaults file contains defaults for most of the sample programs presented in this guide.

1.5 Available Documentation

Use the following table to determine the document or documents you need to accomplish specific tasks.

TABLE 1-2. Documentation Map

Task	Document Title
Configure the X Window System	*A Beginner's Guide to the X Window System* and *Using the X Window System*
Learn how to start the X Window System	*A Beginner's Guide to the X Window System* and *Using the X Window System*
Customize the X Window Environment	*Using the X Window System*
Find resource definitions and values	*HP OSF/Motif Programmer's Reference Manual*
Incorporate widgets into applications	*HP OSF/Motif Programmer's Guide*

Other sources of information are listed below:

- *Xlib Programming Manual For Version 11 Release of the X Window System*, by Adrian Nye, published by O'Reilly and Associates, Newton, MA (1-800-338-NUTS).
- *Xlib Reference Manual For Version 11 Release of the X Window System*, by Adrian Nye, published by O'Reilly and Associates, Newton, MA (1-800-338-NUTS).
- *X Window System User's Guide*, by Tim O'Reilly, Valerie Quercia, and Linda Lamb, published by O'Reilly and Associates, Newton, MA (1-800-338-NUTS).
- *Introduction to the X Window System*, by Oliver Jones, published by Prentice-Hall, Englewood Cliffs, NJ 07632.
- *X Window System, Version 11 Inter-Client Communications Conventions Manual*, by David S. H. Rosenthal, Sun MicroSystems, Mountain View, CA 94043

Widgets, Gadgets, and Convenience Functions 2

The HP OSF/Motif system has a variety of widgets and gadgets, each designed to accomplish a specific set of tasks, either individually or in combination with others. There are convenience functions that create certain widgets or sets of widgets for a specific purpose. This chapter explains widgets, gadgets, and convenience functions.

2.1 Widgets

Widgets are used either individually or in combination to make the creation of complex applications easier and faster. Some widgets display information, others are merely containers for other widgets. Some widgets are restricted to displaying information and do not react to keyboard or mouse input. Others change their display in response to input and can invoke functions when instructed to do so. You can customize some aspects of a widget, such as fonts, foreground and background colors, border widths and colors, and sizes.

An **instance** of a widget class is composed of a data structure containing values and procedures for that particular widget instance. There is also a class structure that contains values and procedures applicable to *all* widgets of that class.

Widgets are grouped into several classes, depending on the function of the widget. Logically, a widget class consists of the procedures and data associated with all widgets belonging to that class. These procedures and data can be inherited by subclasses. Physically, a widget class is a pointer to a structure. The contents of this structure are constant for all widgets of the widget class. A widget instance is allocated and initialized by `XmCreate<`*widget name*`>`, `XmCreateWidget`, or `XmCreateManagedWidget`. See chapter 3, "Using OSF/Motif Widgets in Programs," for specific examples of creating widgets.

This section provides an overview of the available widgets. The man pages in the *HP OSF/Motif Programmer's Reference Manual* contain detailed information for each of the widgets. Figure 2-1 shows how widgets might be combined in an application.

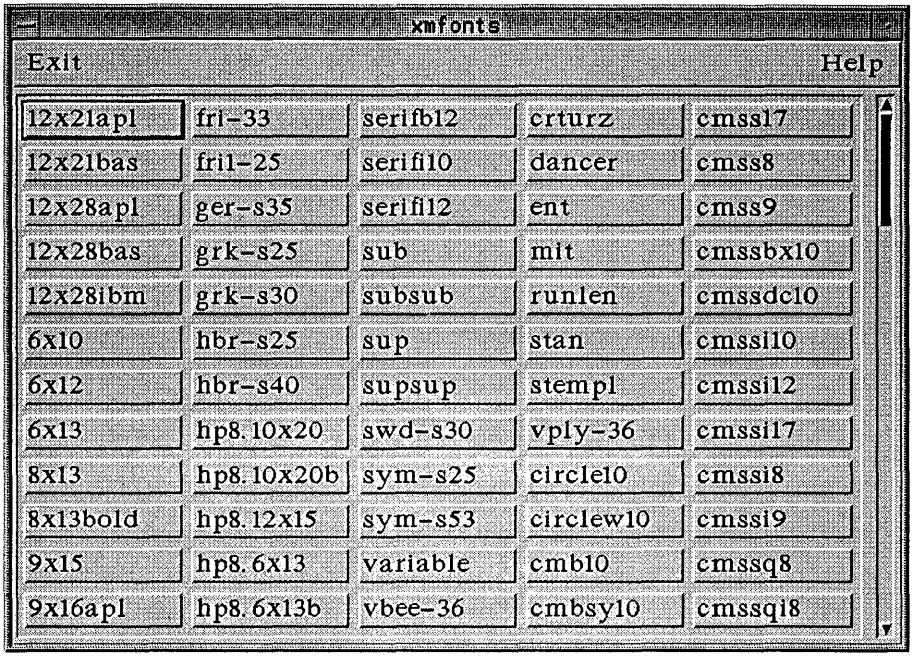


Figure 2-1. Widget Application Screen

Several types of widgets are shown in figure 2-1. The large window is a MainWindow widget within which are a MenuBar and some PushButtons, a RowColumn widget with a number of PushButtonGadgets, and a vertical ScrollBar. The program that produces this window is called `xmfonts`, and it is described in chapter 3.

NOTE

A complete list of resources for each widget can be found in the appropriate man page in the *HP OSF/Motif Programmer's Reference Manual.*

The sections in this chapter divide the widgets into five categories as shown in table 2-1.

TABLE 2-1. Categories of Widgets

Class Name	Widget Class
Shell Widgets	
XmDialogShell	`xmDialogShellWidgetClass`
XmMenuShell	`xmMenuShellWidgetclass`
VendorShell	`VendorShellWidgetClass`
Display Widgets	
Core	`Core`
XmPrimitive	`xmPrimitiveWidgetClass`
XmArrowButton	`xmArrowButtonWidgetClass`
XmDrawnButton	`xmDrawnButtonWidgetClass`
XmLabel	`xmLabelWidgetClass`
XmList	`xmListWidgetClass`
XmPushButton	`xmPushButtonWidgetClass`
XmScrollBar	`xmScrollBarWidgetClass`
XmSeparator	`xmSeparatorWidgetClass`
XmText	`xmTextWidgetClass`
XmToggleButton	`xmToggleButtonWidgetClass`
Container Widgets	
XmManager	`xmManagerWidgetClass`
XmDrawingArea	`xmDrawingAreaWidgetClass`
XmFrame	`xmFrameWidgetClass`
XmMainWindow	`xmMainWindowWidgetClass`
XmPanedWindow	`xmPanedWindowWidgetclass`
XmRowColumn	`xmRowColumnWidgetClass`
XmScale	`xmScaleWidgetClass`
XmScrolledWindow	`xmScrolledWindowWidgetClass`
Dialog Widgets	
XmBulletinBoard	`xmBulletinBoardWidgetClass`
XmCommand	`xmCommandWidgetClass`
XmFileSelectionBox	`xmFileSelectionBoxWidgetClass`
XmForm	`xmFormWidgetClass`
XmMessageBox	`xmMessageBoxWidgetClass`
XmSelectionBox	`xmSelectionBoxWidgetClass`
Menu Widgets	
XmCascadeButton	`xmCascadeButtonWidgetClass`
XmCascadeButtonGadget	`xmCascadeButtonGadgetClass`

2.1.1 Shell Widgets

Shell widgets are top-level widgets that provide the necessary interface with the window manager. Different Shell widget classes are provided for the various categories of top-level widgets. The Xt Intrinsics provide some underlying shells and the OSF/Motif toolkit provides the remaining shells. The Xt Intrinsics provide the following shell classes:

- `Shell` - This is the base class for shell widgets. It is a subclass of Composite and provides resources for all other types of shells.
- `OverrideShell` - This class is used for shell windows that completely bypass the window manager. It is a subclass of Shell.
- `WMShell` - This class contains resources that are necessary for the common window manager protocol. It is a subclass of Shell.
- `VendorShell` - This class contains resources used by vendor-specific window managers. It is a subclass of WMShell.
- `TransientShell` - This class is used for shell windows that can be manipulated by the window manager but cannot be iconified. It is a subclass of VendorShell.
- `TopLevelShell` - This class is used for normal top-level windows. It is a subclass of VendorShell.
- `ApplicationShell` - This class is used for an application's top-level window. It is a subclass of TopLevelShell.

The classes `Shell`, `WMShell`, and `VendorShell` are internal and cannot be instantiated.

The OSF/Motif toolkit provides the following widgets:

XmDialogShell (xmDialogShellWidgetClass)

The DialogShell widget class is a subclass of TransientShell. Instances of this class are used as the parents of modal and modeless Dialogs associated with other top-level windows. DialogShell provides proper communication with the OSF/Motif Window Manager in accordance with the Inter-Client Communications Conventions Manual (ICCCM) for secondary top-level windows, such as Dialogs. See chapter 5, "Dialog Widgets," for more information about how this widget is used by the Dialog widgets.

XmMenuShell (xmMenuShellWidgetClass)

The MenuShell widget class is a subclass of OverrideShell. Instances of this class are used as the parents of MenuPanes. See chapter 6, "Menus," for the specifications of menu widgets and menu shells.

VendorShell (xmVendorShellWidgetClass)

The VendorShell widget class is a subclass of WMShell. It provides the common state information and services needed by the window-manager visible shells.

See chapter 4, "Shell Widgets," for more information.

2.1.2 Display Widgets

The OSF/Motif system provides the following display widgets:

Core

The Core class is used as a supporting superclass for other widget classes. It provides common resources that are needed by all widgets, including x and y location, height, width, window border width, and so on.

XmPrimitive (xmPrimitiveWidgetClass)

The XmPrimitive class is also used as a supporting superclass for other widget classes. It provides resources for border drawing and highlighting, traversal activation and deactivation, and so on.

XmArrowButton (xmArrowButtonWidgetClass)

The ArrowButton widget consists of a directional arrow surrounded by a border shadow. When the ArrowButton is selected, the shadow moves to give the appearance that the ArrowButton has been pressed in. When the ArrowButton is unselected, the shadow moves to give the appearance that the ArrowButton is released, or "out." The ArrowButton has the same functionality as the PushButton. Figure 2-2 shows four ArrowButtons arranged within a RowColumn widget.

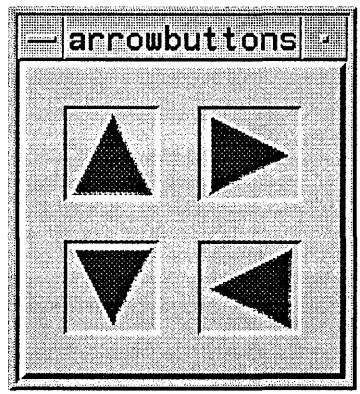


Figure 2-2. ArrowButtons

The direction of the arrow is specified by setting the XmNarrowDirection resource to the appropriate value. The spacing between the ArrowButtons in figure 2-2 was obtained by setting the RowColumn resources XmNmarginWidth, XmNmarginHeight, and XmNspacing to 20.

XmDrawnButton (xmDrawnButtonWidgetClass)

The DrawnButton widget consists of an empty widget window surrounded by a shadow border. It provides the application developer with a graphics area that can have PushButton input semantics.

Callback types are defined for widget exposure and resize to allow the application to redraw or reposition its graphics. If the DrawnButton widget has a highlight and shadow thickness, the application should take care not to draw in this area. This can be done by creating the graphics context to be used for drawing in the widget with a clipping rectangle. The clipping rectangle should take into account the size of the widget's highlight thickness and shadow.

XmLabel (xmLabelWidgetClass)

A Label consists of either text or graphics. It can be instantiated but it is also used as a superclass for button widgets. Label's text is a compound string and can be multidirectional, multiline, multifont, or any combination of these. Label is considered static because it does not accept any button or key input other than the "help" button on the widget. The "help" callback is the only callback defined for Label.

XmList (xmListWidgetClass)

The List widget allows you to make a selection from a list of items. The application defines an array of compound strings, each of which becomes an item in the list. You can set the number of items in the list that are to be visible. You can also choose to have the List appear with a ScrollBar so that you can scroll through the list of items. Items are selected by moving the pointer to the desired item and pressing the mouse button or key defined as "select." The selected item is displayed in inverse colors.

Figure 2-3. List Widget

XmPushButton (xmPushButtonWidgetClass)

This widget consists of a text label or pixmap surrounded by a border shadow. You select the button by moving the mouse cursor to the button and pressing mouse button 1. When the mouse button is pressed, the widget and shadow colors will invert, giving the appearance that the PushButton has been pressed in. When the mouse button is released, the colors will revert to the original color scheme, giving the appearance that the PushButton is "out." PushButtons are used to invoke actions, such as run, cancel, stop, and so on.

XmScrollBar (xmScrollBarWidgetClass)

The ScrollBar widget allows you to view data that is too large to be viewed in its entirety. ScrollBars are combined with a widget that contains the data to be viewed. When you interact with the ScrollBar, the data scrolls. The viewable portion of the data is called the work area.

A ScrollBar consists of two arrows pointing in opposite directions at each end of a narrow rectangle. The rectangle is called the **scroll region**. A smaller rectangle called a slider is positioned within the scroll region. The slider is normally colored to contrast with that of the scroll region. The ratio of the slider size to the scroll region size corresponds to the relationship between the visible data and the total data. For example, if ten percent of the data is visible in the work area, the slider takes up ten percent of the scroll region.

You may place the ScrollBar horizontally or vertically or both. Horizontal ScrollBars are placed at the bottom edge of the work area and vertical ScrollBars are placed on the right edge. The ScrollBar widget is shown in figure 2-4.

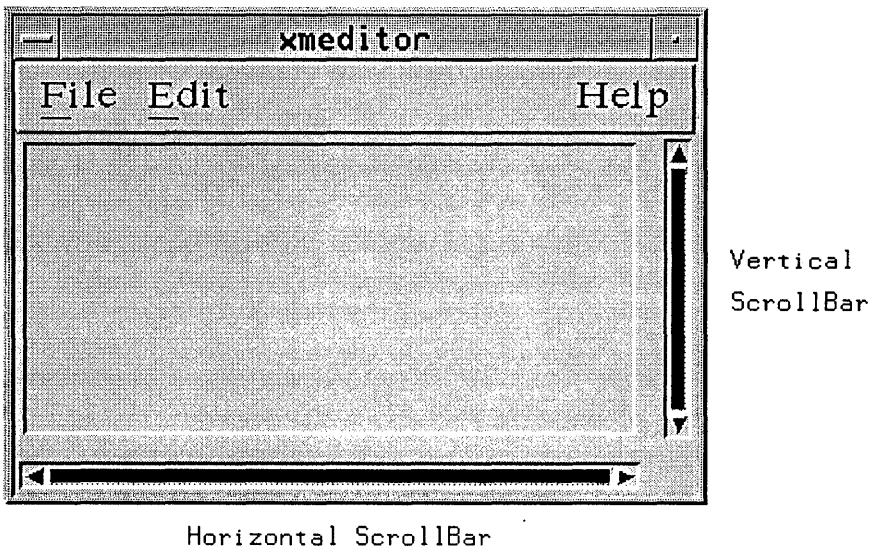


Figure 2-4. ScrollBars

XmSeparator (xmSeparatorWidgetClass)

Separator is a primitive widget to be used as an item separator placed between items in a display. Several different line drawing styles are provided as well as horizontal or vertical orientation.

The line drawing done within the Separator is automatically centered within the height of the widget for a horizontal orientation, and centered within the width of the widget for a

vertical orientation.

The XmNseparatorType of XmNO_LINE is provided as an escape to the application programmer who needs a different style of drawing. A pixmap the height of the widget can be created and used as the background pixmap by building an argument list using the XmNbackgroundPixmap argument type as defined by Core. Whenever the widget is redrawn its background that contains the desired Separator drawing is displayed.

XmText (xmTextWidgetClass)

The Text widget provides a single or multiline text editor that has a user and programmer interface that you can customize. It can be used for single-line string entry, forms entry with verification procedures, multipage document viewing, and full-screen editing. See chapter 7 for more information on the Text widget.

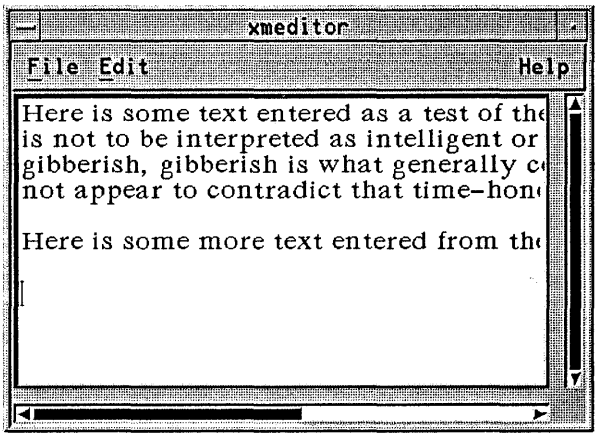


Figure 2-5. Text Widget

XmToggleButton (xmToggleButtonWidgetClass)

This widget consists of a text or graphics label with an indicator (a square or diamond-shaped box) placed to the left of the text or graphics. If the resource `XmNborderWidth` is set to any value greater than zero, the ToggleButton will have a visible border, as shown in figure 2-6.

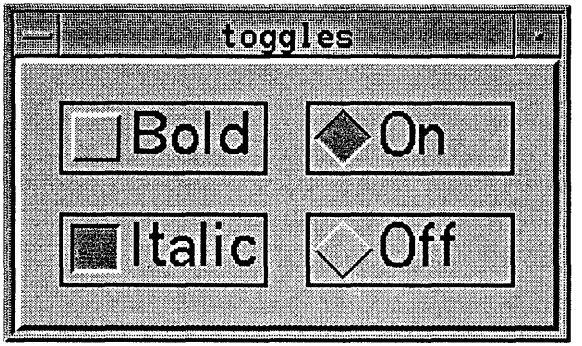


Figure 2-6. ToggleButtons

You can have the ToggleButton appear with a three-dimensional shadow as shown in figure 2-7. To do this, set the resource `XmNshadowThickness` to some value greater than zero, instead of `XmNborderWidth`.

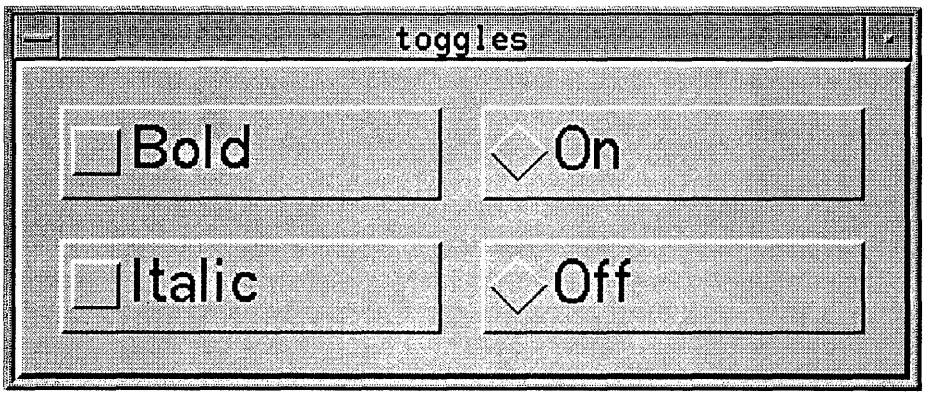


Figure 2-7. ToggleButtons

The indicator will change size to correspond to the font used for the text label. The overall height and width of the ToggleButton can be changed when the Label resource `XmNrecomputeSize` is set to False. If this resource is set to True, then only enough space to draw the ToggleButton is allocated. The ToggleButtons in figure 2-7 were set to a height of 50 pixels and a width of 200 pixels. One way to do this is by setting the applicable resources in a defaults file, as shown below.

```
*XmToggleButton.recomputeSize:     False
*XmToggleButton.height:            50
*XmToggleButton.width:             200
*XmToggleButton.shadowThickness:   2
```

Select the ToggleButton by moving the mouse cursor inside the rectangle and pressing mouse button 1. The indicator is then filled with the selection color, indicating that the ToggleButton is selected.

2.1.3 Container Widgets

Container widgets are Composite widgets that provide applications with general layout functionality. Since they are Composite widgets, Container widgets can have children. All of the container widgets are built from the Core, Composite, Constraint, and XmManager widget classes. The OSF/Motif system provides the following container widgets:

XmManager (xmManagerWidgetClass)

The XmManager class is an OSF/Motif widget meta class and is therefore never instantiated as a widget. Its sole purpose is to act as a supporting superclass for other widget classes. It supports the visual resources, graphics contexts and traversal resources necessary for the graphics and traversal mechanisms. XmManager is built from Core, Composite, and Constraint.

XmDrawingArea (xmDrawingAreaWidgetClass)

The DrawingArea widget is an empty widget that is easily adaptable to a variety of purposes. DrawingArea does no drawing and defines no behavior except for invoking callbacks. Callbacks notify the application when graphics need to be drawn (exposure events or widget resize), and when the widget receives input from the keyboard or mouse. Applications are responsible for defining appearance and behavior as needed in response to DrawingArea callbacks.

DrawingArea is a Composite widget and is a subclass of XmManager. It supports minimal geometry management for multiple widget or gadget children.

XmFrame (xmFrameWidgetClass)

The XmFrame widget is a manager that is used to enclose a single child within a border drawn by the XmFrame widget. It is most often used to enclose other Managers when it is desired to have the same border appearance for the XmManager and XmPrimitive widgets it manages.

XmMainWindow (xmMainWindowWidgetClass)

The XmMainWindow widget provides a standard layout for the primary window of an application. This layout includes a MenuBar, a CommandWindow, a work region, and ScrollBars. Any or all of these areas are optional. The work region and ScrollBars in the MainWindow behave exactly the same as their counterparts in the ScrolledWindow widget. You can think of the MainWindow as an extended ScrolledWindow with an optional MenuBar and an optional CommandWindow.

In a fully loaded MainWindow, the MenuBar spans the top of the window horizontally. The CommandWindow spans the MainWindow horizontally and is placed just below the MenuBar. Any space below the CommandWindow is managed exactly the same as the ScrolledWindow. To create a fully loaded MainWindow, you create a MenuBar, a CommandWindow, two ScrollBars (one horizontal and one vertical), and a widget to use as the work region. You then call `XmMainWindowSetAreas` with those widget ID's.

XmRowColumn (xmRowColumnWidgetClass)

The RowColumn widget is a general purpose RowColumn manager capable of containing any widget type as a child. It requires no special knowledge about how its children function and provides nothing above and beyond support for several different layout styles.

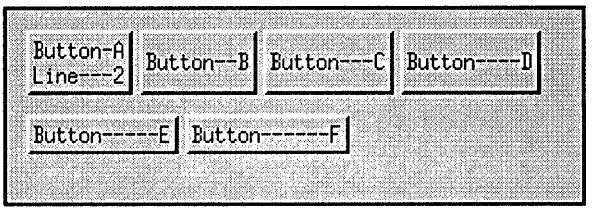


Figure 2-8. RowColumn Widget

The type of layout performed is controlled by how the application has set the various layout resources. It can be configured to lay out its children in either a row or a column fashion. In addition, the application can specify whether the children should be packed tightly together (not into organized rows and columns), or whether each child should be placed in an identically sized box (thus producing a symmetrical look), or whether specific layout should be done (the current x and y positions of the children control their location).

In addition, the application has control over both the spacing that occurs between each row and column, and the margin spacing between the edges of the RowColumn widget and any children that are placed against it.

The RowColumn widget has no three-dimensional visuals associated with it. If you want an application to have a three-dimensional shadow placed around the RowColumn widget, then you should create the RowColumn widget as a child of a Frame widget.

XmScale (xmScaleWidgetClass)

The Scale widget has two basic functions.

- It is used by an application to indicate a value from within a range of values.
- It allows the user to input or modify a value from the same range.

A Scale widget allows you to select a value from a range of displayed values by adjusting an arrow to a position along a line. A Scale has an elongated rectangular region similar to that

of a ScrollBar. Inside this region is a slider that is used to indicate the current value along the Scale. You can modify the value of the Scale by moving the slider within the rectangular region of the Scale. A Scale can also include a set of labels and "tick marks" located outside of the Scale region. These can be used to indicate the relative value at various positions along the scale.

A Scale can be either input and output or output only. An input/output Scale is one whose value can be set by the application and also modified by the user by using the slider. An output-only Scale is one that is used strictly as an indicator of the current value of something and cannot be modified interactively by the user. The Core resource `XmNsensitive` is used to specify whether or not the user can interactively modify the value of the Scale.

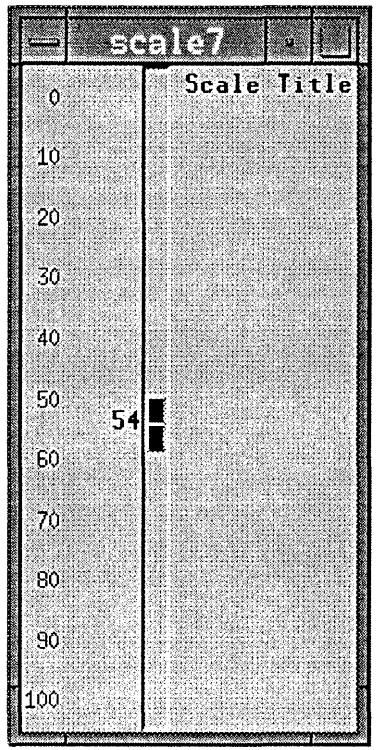


Figure 2-9. Scale Widget

XmScrolledWindow (xmScrolledWindowWidgetClass)

The ScrolledWindow widget combines one or more ScrollBar widgets and a viewing area to implement a visible window onto some other (usually larger) data display. The visible part of the window can be scrolled through the larger display by the use of ScrollBars.

To use the ScrolledWindow, an application first creates a ScrolledWindow widget, the needed ScrollBar widgets, and a widget capable of displaying any desired data as the work area of the ScrolledWindow. ScrolledWindow will position the work area widget and display the ScrollBars if so requested. When the user performs some action on the ScrollBar, the application will be notified through the normal ScrollBar callback interface.

The ScrolledWindow can be configured to operate in an automatic manner, so that it performs all scrolling and display actions with no need for application program involvement. It can also be configured to provide a minimal support framework in which the application is responsible for processing all user input and making all visual changes to the displayed data in response to that input.

When the ScrolledWindow is performing automatic scrolling it will create a clipping window. Conceptually, this window becomes the viewport through which the user examines the larger underlying data area. The application simply creates the desired data, then makes that data the work area of the ScrolledWindow. When the user moves the slider to change the displayed data, the workspace is moved under the viewing area so that a new portion of the data becomes visible.

There are situations where it is impractical for an application to create a large data space and simply display it through a small clipping window. An example of this is a text editor — there would be an undesirable amount of overhead involved with creating a single data area that consisted of a large file. The application should use the concept of a ScrolledWindow (a small viewport onto some larger data), but it should be notified when the user scrolls the viewport so it can bring in more data from storage and update the display area. For this situation the ScrolledWindow can be configured so that it provides only visual layout support. No clipping window is created and the application must maintain the data displayed in the work area as well as respond to user input on the ScrollBars. Figure 2-10 shows a ScrolledWindow with some text in it. Note that the scroll bars indicate that scrolling is possible in either direction.

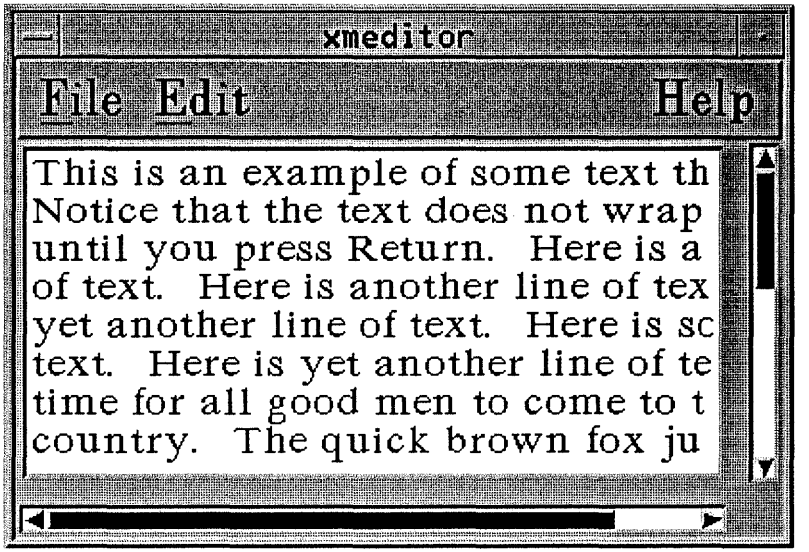


Figure 2-10. ScrolledWindow Widget

Figure 2-11 shows the same window after partially scrolling down. Compare the positions of the vertical scroll bar and the text with those of figure 2-10.

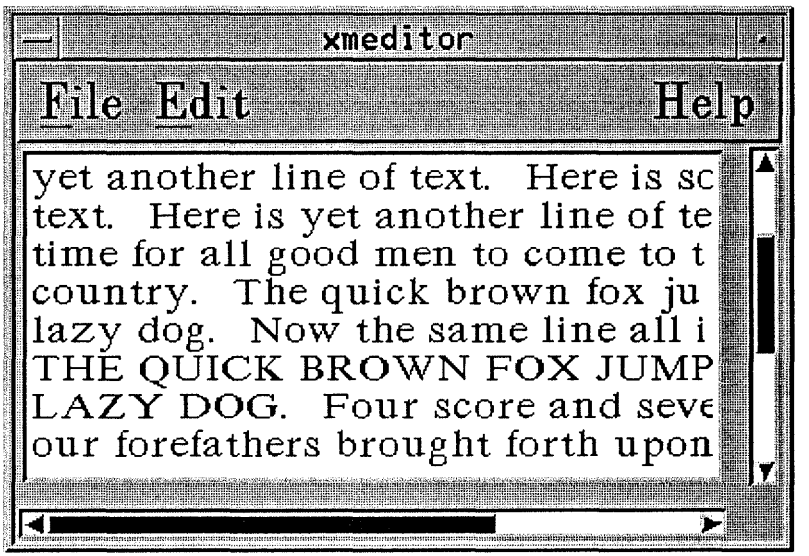


Figure 2-11. ScrolledWindow After Scrolling

XmPanedWindow (xmPanedWindowWidgetClass)

The PanedWindow manager widget is a Composite widget that lays out children in a vertically tiled format. Children appear from top-to-bottom, with the first child inserted appearing at the top of the PanedWindow manager and the last child inserted appearing at the bottom. The PanedWindow manager will grow to match the width of its widest child, and all other children are forced to this width. The height of the PanedWindow manager will be equal to the sum of the heights of all its children, the spacing between them, and the size of the top and bottom margins.

The PanedWindow manager widget is also a constraint widget, which means that it creates and manages a set of constraints for each child. It is possible to specify a minimum and maximum size for each pane. The PanedWindow manager will not allow a pane to be resized below its minimum size nor beyond its maximum size. Also, when the minimum size of a pane is equal to its maximum size, then no control sash will be presented for that pane or for the lowest pane. Figure 2-12 shows an example of a PanedWindow widget with three ArrowButtons as its children

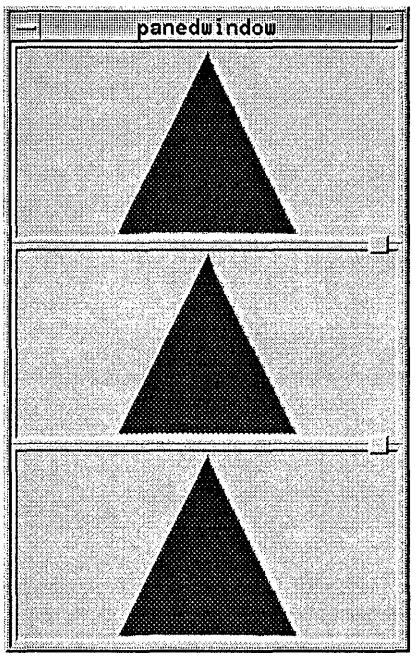


Figure 2-12. PanedWindow Widget

It is possible for you to adjust the size of the panes. To facilitate this adjustment, a pane control sash is created for most children. The sash appears as a square box positioned on the bottom of the pane that it controls (see figure 2-12). You can adjust the size of a pane

by using the mouse. Position the pointer inside the sash and a crosshair appears. Press and hold mouse button 1 and the pointer changes to an arrow pointing up and down. Continue holding mouse button 1 down while you move the pointer to achieve the desired size of the pane. Release mouse button 1 and the panes will be resized. Figure 2-13 shows the PanedWindow after a pane has been resized.

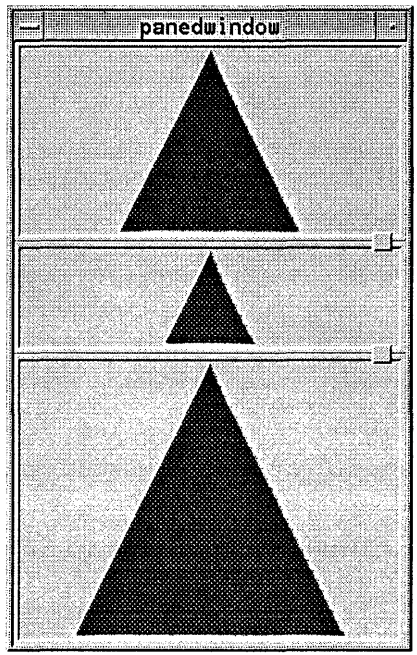


Figure 2-13. PanedWindow Widget After Pane Resizing

2.1.4 Dialog Widgets

Dialog widgets are container widgets that provide applications with layout functionality typically used for popup "dialogs." These widgets are used for interaction tasks such as displaying messages, setting properties, and providing selection from a list of items. Dialog widgets are thus used primarily as an interface between the user and the application. A Dialog widget will normally ask a question or present the user with some information that requires a response. In some cases the application will be suspended until the user provides the response.

A **Dialog** is a collection of widgets, including a DialogShell, a BulletinBoard (or subclass of BulletinBoard or some other container widget), plus various children of the BulletinBoard, such as Label, PushButton and Text widgets. All of the dialog widgets are built from the

Core, Composite, Constraint, and Manager widget classes.

The collection of widgets that compose a Dialog can be built from scratch by building up the necessary argument lists and creating each individual widget in the Dialog. For common interaction tasks, **convenience functions** are defined that create the collection of widgets that comprise a particular Dialog. The collections of widgets created by Dialog convenience functions are referred to as **Convenience Dialogs.**

Convenience Dialogs are either modal or modeless. A modal dialog stops the work session and solicits input from the user. A modeless dialog solicits input from the user, but doesn't interrupt interaction with any application.

Each dialog has one or more convenience functions that create any of the subwidgets in that dialog. For example, MessageBox has several convenience functions:

- `XmCreateMessageDialog`.
- `XmCreateErrorDialog`.
- `XmCreateInformationDialog`.
- `XmCreateQuestionDialog`.
- `XmCreateWarningDialog`.
- `XmCreateWorkingDialog`.

Each of these convenience functions create a DialogShell and a MessageBox. Refer to chapter 5, "Dialog Widgets and Functions," and the individual man pages for more information.

2.1.5 Dialog Widget Descriptions

The following list gives an overview of the Dialog widget set. See the next section for an overview of the convenience dialogs.

XmBulletinBoard (xmBulletinBoardWidgetClass)

The BulletinBoard widget is a composite widget that provides simple geometry management for children widgets. It does not force positioning on its children, but can be set to reject geometry requests that would result in overlapping children. BulletinBoard is the base widget for most dialog widgets, but is also used as a general container widget.

XmCommand (xmCommandWidgetClass)

The Command widget is a subclass of SelectionBox that includes a command history region and a command input region. Command also provides a command history mechanism.

XmFileSelectionBox (xmFileSelectionBoxWidgetclass)

The FileSelectionBox widget is a subclass of SelectionBox and BulletinBoard used to get a selection from a list of alternatives. FileSelectionBox includes an editable text field for the directory mask, a scrolling list of filenames. and an editable text field for the selected file. Four buttons are available: "OK," "Filter," "Cancel," and "Help" by default.

XmForm (xmFormWidgetClass)

The Form widget is a constraint-based manager that provides a layout language used to establish spatial relationships between its children. It maintains these relationships when the Form is resized, new children are added to the Form, or its children are resized, unmanaged, remanaged, or destroyed. Since it is a subclass of BulletinBoard, Form includes the base level of dialog support. Form can also be used as a general container widget.

XmMessageBox (xmMessageBoxWidgetClass)

The MessageBox widget is a subclass of BulletinBoard used to give information to the user. MessageBox includes a symbol and a message. Three buttons are available: "OK," "Cancel," and "Help" by default.

XmSelectionBox (xmSelectionBoxWidgetClass)

The SelectionBox widget is a subclass of BulletinBoard used to get a selection from a list of alternatives. SelectionBox includes a message, an editable text field, and a scrolling list of choices. Four buttons are available: "OK," "Cancel," "Apply," and "Help" by default.

2.1.6 Convenience Dialogs

Convenience Dialogs are collections of widgets that can be created by using convenience functions. Each convenience dialog instantiates a dialog widget as a child of a DialogShell. This section lists the Convenience Dialogs.

BulletinBoardDialog

The BulletinBoardDialog convenience function instantiates a BulletinBoard and a DialogShell. The BulletinBoardDialog is used for interactions not supported by the standard dialog set. Necessary dialog components are added as children of the BulletinBoard.

ErrorDialog

The ErrorDialog convenience function instantiates a MessageBox and a DialogShell. The ErrorDialog is used to warn the user of an invalid or potentially dangerous condition. ErrorDialog includes a symbol and a message. Three buttons are available: "OK," "Cancel," and "Help" by default. The default ErrorDialog symbol is a hexagon with a hand inside.

FileSelectionDialog

The FileSelectionDialog convenience function instantiates a FileSelectionBox and a DialogShell. The FileSelectionDialog is used to select a file. FileSelectionDialog includes an editable text field for the directory mask, a scrolling list of filenames, and an editable text field for the selected file. Four buttons are available: "OK," "Filter," "Cancel," and "Help" by default.

FormDialog

The FormDialog convenience function instantiates a Form and a DialogShell. The FormDialog is used for interactions not supported by the standard dialog set. Necessary dialog components are added as children of the Form.

InformationDialog

The InformationDialog convenience function instantiates a MessageBox and a DialogShell. The InformationDialog is used to give information to the user, such as the status of an action. InformationDialog includes a symbol and a message. Three buttons are available: "OK," "Cancel," and "Help" by default. The default InformationDialog symbol is a square icon with an "i" in the center.

MessageDialog

The MessageDialog convenience function instantiates a MessageBox and a DialogShell. The MessageDialog is used to give information to the user. MessageDialog may include a symbol and a message. There is no symbol by default. Three buttons are available: "OK," "Cancel," and "Help" by default.

PromptDialog

The PromptDialog convenience function instantiates a SelectionBox and a DialogShell. The PromptDialog is used to prompt the user for text input. PromptDialog includes a message and a text input region. Four buttons are available: "OK," "Apply," "Cancel," and "Help" by default.

QuestionDialog

The QuestionDialog convenience function instantiates a MessageBox and a DialogShell. The Question Dialog is used to get the answer to a question from the user. QuestionDialog includes a symbol and a message. Three buttons are available: "OK," "Cancel," and "Help" by default. A "?" is the default QuestionDialog symbol.

SelectionDialog

The SelectionDialog convenience function instantiates a SelectionBox and a DialogShell. The SelectionDialog is used to get a selection from a list of alternatives. SelectionDialog includes a message, an editable text field, and a scrolling list of choices. Four buttons are available: "OK," "Apply," "Cancel," and "Help" by default.

WarningDialog

The WarningDialog convenience function instantiates a MessageBox and a DialogShell. The WarningDialog is used to warn the user of the consequences of an action, and give the user a choice of resolutions. WarningDialog includes a symbol and a message. Three buttons are available: "OK," "Cancel," and "Help" by default. A "!" is the default WarningDialog symbol.

Working Dialog

The WorkingDialog convenience function instantiates a MessageBox and a DialogShell. The WorkingDialog is used to inform the user that there is a time consuming operation in progress and give the user the ability to cancel the operation. WorkingDialog includes a symbol and a message. Three buttons are available: "OK," "Cancel," and "Help" by default. The WorkingDialog symbol is a square icon with an hourglass in the center.

2.1.7 Menu Widgets

The RowColumn widget is the basis for most of the menu system components. It has a built-in ability to behave like a RowColumn manager, a RadioBox, a MenuBar, a Pulldown MenuPane, a Popup MenuPane, and an Option menu. Convenience functions have been provided to easily create these special versions of the RowColumn widget.

The OSF/Motif menu system is composed of the following widgets and convenience functions:

- XmRowColumn (Widget)
- MenuBar (Convenience Function)
- OptionMenu (Convenience Function)
- Pulldown Menupane (Convenience Function)
- Popup Menupane (Convenience Function)

- XmMenuShell (Widget)
- XmCascadeButton (Widget)
- XmSeparator (Widget and Gadget)
- XmLabel (Widget and Gadget)
- XmToggleButton (Widget and Gadget)
- XmPushButton (Widget and Gadget)

Applications are not required to use all of these components to use the menu system.

2.2 Gadgets

Gadgets provide essentially the same functionality as the equivalent primitive widgets. The primary motivation behind providing a set of gadgets is to improve performance, both in execution time and data space. This applies to both the application and server processes and minimizes the amount of lost functionality. The performance difference between widgets and gadgets is dramatic, so it is highly recommended that applications use gadgets whenever possible.

Gadgets can be thought of as a windowless widget. This means that they do not have windows, translations, actions, or popup children. Also, gadgets do not have any of the visual resources found in the XmPrimitive class for primitive widgets. These visuals are referenced by a gadget from its parent.

Examples of display gadgets include buttons, labels and separators. All of these gadgets are built from the classes of Object, RectObj, and XmGadget. The table below shows the gadgets and their class names.

TABLE 2-2. Gadgets

Gadget Name	Gadget Class
Object	`Object`
RectObj	`Rect`
XmGadget	`xmGadgetClass`
XmArrowButtonGadget	`xmArrowButtonGadgetClass`
XmLabelGadget	`xmLabelGadgetClass`
XmPushButtonGadget	`xmPushButtonGadgetClass`
XmSeparatorGadget	`xmSeparatorGadgetClass`
XmToggleButtonGadget	`xmToggleButtonGadgetClass`

The following list provides an overview of the set of display gadgets.

Object (Object)
The Object class is an Xt Intrinsics meta class and is therefore never instantiated. It is used as a supporting superclass to provide common resources to other classes.

RectObj (Rect)
The RectObj class is an Xt Intrinsics meta class and is therefore never instantiated. It is used as a supporting superclass to provide common resources to other classes.

XmGadget (xmGadgetClass)
XmGadget is an OSF/Motif meta class and is therefore never instantiated. It is used as a supporting superclass to provide common resources to other gadget classes.

XmArrowButtonGadget (xmArrowButtonGadgetClass)
ArrowButtonGadget has the same functionality as PushButtonGadget but displays a directional arrow within itself.

XmLabelGadget (xmLabelGadgetClass)
A LabelGadget consists of either text or graphics. It can be instantiated but it is also used as a superclass for button widgets. The LabelGadget's text is a compound string and can be multidirectional, multiline, multifont, or any combination of these. LabelGadget is considered static because it does not accept any button or key input other than the "help" button on the widget. The "help" callback is the only callback defined for LabelGadget.

XmPushButtonGadget (xmPushButtonGadgetClass)
PushButtonGadgets are used to issue commands within an application. PushButtonGadget displays a label with a border-shadowing graphic. When the pushbutton is selected, the shadow moves to give the appearance that the push button has been pressed in. When the pushbutton is unselected, the shadow moves to give the appearance that the pushbutton is "out."

XmSeparatorGadget (xmSeparatorGadgetClass)
SeparatorGadget is used to provide a visual separation between groups of widgets. It can draw horizontal and vertical lines in several different styles.

XmToggleButtonGadget (xmToggleButtonGadgetClass)
XmToggleButtonGadget consists of a text or graphics button face with an indicator (a square or diamond-shaped box) placed to the left of the text or graphics. You select the ToggleButtonGadget by placing the mouse cursor inside the rectangle and pressing mouse button 1. The indicator is then filled with the selection color, indicating that the ToggleButtonGadget is selected. ToggleButtonGadgets are used for setting nontransitory data within an application.

2.3 Convenience Functions

Convenience functions are functions that enable you to create certain widgets or gadgets, or groups of widgets or gadgets, by making just one function call. A Convenience function creates a predetermined set of widgets and returns the *parent* widget's ID. Convenience functions are of the form

```
XmCreate<widgetname>
```

for widgets and gadgets *other than* Dialog widgets. For Dialogs, convenience functions are referred to as **Convenience Dialogs**, and are of the form

```
<DialogWidgetName>Dialog
```

It is very easy to use a convenience function to create a widget. For example, you can use the following code segment to create a Label widget:

```
Widget XmCreateLabel (parent,name,arglist,argcount)

  Widget parent;
  String name;
  Arglist arglist;
  Cardinal argcount;
```

parent Specifies the parent widget for the Label.

name Specifies the resource name for the Label. This name is used for retrieving resources, and therefore it should not be the same as any widget that is a child of the same parent, *unless* identical resource values are to be used for the child widgets.

arglist Specifies the argument list used to override the default values for the Label's resources.

argcount Specifies the number of arguments in the arglist.

The `XmCreate<widgetname>` functions create *unmanaged* widgets. Your application *must* manage the set of widgets before they will be displayed. You can manage each widget separately or as a group. Use this code segment to create and manage each widget separately:

```
Widget w;
```

```
w = XmCreate<widgetname>(parent, name, arglist, argcount);
XtManageChild(w);
```

Use this code segment to create and manage widgets *with the same parent* as a group:

```
int child_count = 0;
Widget w[10];

w[child_count++] = XmCreate<widgetname)(parent, name, arglist,
  argcount)
w[child_count++] = XmCreate<widgetname)(parent, name, arglist,
  argcount)
w[child_count++] = XmCreate<widgetname)(parent, name, arglist,
  argcount)
                                        .
                                        .
                                        .
XtManageChildren(w, child_count);
```

Using OSF/Motif Widgets in Programs 3

This chapter explains how to write applications that use the OSF/Motif widgets.

Writing widget programs involves nine steps:

TABLE 3-1. Steps in Writing Widget Programs

Step	Description	Related Functions
1	Include required header files.	`#include <Xm/Xm.h>` `#include <Xm/`*widget*`.h>`
2	Initialize Xt Intrinsics.	`XtInitialize( . . .)`
3	Add additional top-level windows.	`XtAppCreateShell( . . .)`
	Do steps 4 through 6 for each widget.	
4	Set up argument lists for widget.	`XtSetArg( . . .)`
5	Create the widget.	`XtCreateManagedWidget( . . .)` or `XmCreate`*widget* followed by `XtManageChild(`*widget*`)`
6	Add callback routines.	`XtAddCallback( . . .)`
7	Realize widgets and loop.	`XtRealizeWidget(`*parent*`)` `XtMainLoop()`
8	Link relevant libraries.	`cc +Nd2000 +Ns2000 -o`*application* `\` *application*`.c -lXm -lXt -lX11 -lPW`
9	Create defaults files.	`/usr/lib/X11/app-defaults/`*class* `$HOME/.Xdefaults`

Sections 3.1 through 3.8 of this chapter describe each of the steps except step 3. That step is covered in section 3.11. The sample code segments of each section build a simple widget program (called `xmbutton`) that implements a PushButton widget. The widget tree diagram, the program, the defaults file, and a picture of the output are listed on the following pages.

Program xmbutton Widget Tree Diagram

The following diagram shows the xmbutton widget tree.

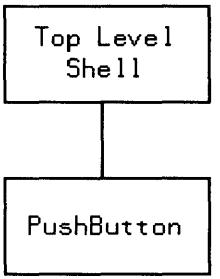


Figure 3-1. xmbutton Widget Tree

Program Listing xmbutton.c

The source code for xmbutton.c is listed below. You can also find it in the directory */usr/contrib/Xm*.

```
/**---------------------------------------------------------
***
***file: xmbutton.c
***
***project: OSF/Motif widgets example programs
***
***description: This program creates a PushButton widget.
***
***
***(c) Copyright 1989 by Open Software Foundation, Inc.
***    All Rights Reserved.
***
***(c) Copyright 1989 by Hewlett-Packard Company.
***
***---------------------------------------------------------


/*  include files  */

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
```

```
/*  functions defined in this program  */

void main();
void activateCB(); /* Callback for the PushButton */


/*  global variables  */

char *btn_text;/* button label pointer for compound string */


/*--------------------------------------------------------------
**main - main logic for xmbutton program
*/
void main (argc,argv)
unsigned int argc;
char     **argv;
{
Widget   toplevel; /* Shell widget      */
Widget   button;   /* PushButton widget */
Arg      args[10]; /* arg list          */
register int n;    /* arg count         */

/*  initialize toolkit  */
toplevel = XtInitialize ("main", "XMdemos", NULL, NULL,
  &argc, argv);

/*  create compound string for the button text  */
btn_text = XmStringCreateLtoR("Push Here",
  XmSTRING_DEFAULT_CHARSET);

/*  set up arglist  */
n = 0;
XtSetArg (args[n], XmNlabelType, XmSTRING);  n++;
XtSetArg (args[n], XmNlabelString, btn_text);  n++;
XtSetArg (args[n], XmNwidth, 250);  n++;
XtSetArg (args[n], XmNheight, 150);  n++;
/*  create button  */
button = XtCreateManagedWidget ("button", xmPushButtonWidgetClass,
  toplevel, args, n);
/*  add callback  */
XtAddCallback (button, XmNactivateCallback, activateCB, NULL);
/*  realize widgets  */
```

```
XtRealizeWidget (toplevel);
/*  process events  */
XtMainLoop ();
}


/*--------------------------------------------------------------
**activateCB - callback for button
*/
void activateCB (w, client_data, call_data)
Widgetw;/*  widget id*/
caddr_tclient_data;/*  data from application   */
caddr_tcall_data;/*  data from widget class  */
{
/*  print message, free compound string memory,
    and terminate program  */
printf ("PushButton selected.\n");
XtFree(btn_text);
exit (0);
}
```

Defaults File XMdemos Partial Listing

NOTE

All the example programs in this manual use the same defaults file XMdemos. This file contains default specifications for general appearance and behavior, plus, in some cases, program unique specifications.

```
!
!XMdemos app-defaults file for OSF/Motif demo programs
!
!general appearance and behavior defaults
!
*foreground:          white
*fontList:            vr-20
*allowShellResize:    true
*borderWidth:         0
*highlightThickness:  2
```

```
*keyboardFocusPolicy: explicit
*menuAccelerator:     <Key>KP_F2
```

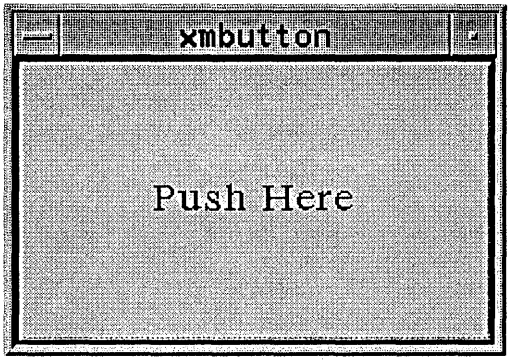


Figure 3-2. Sample Program xmbutton Screen Display

Section 3.9 describes the use of color in screen design and section 3.10 introduces some advanced programming techniques. Section 3.11 presents a more involved sample program.

NOTE

This chapter assumes you have a working knowledge of the C programming language. You should be particularly familiar with pointers and structures. If you are not, be sure to study a book on programming with C. Books on the topic are widely available in computer bookstores.

The following sections describe the process for writing widget programs summarized in table 3-1. Following these steps will help you start writing programs that use the OSF/Motif widgets.

3.1 Including Header Files

Special variables and types of variables used by OSF/Motif programs are defined in header files. Include the appropriate files at the beginning of your program. The man page for each widget specifies the header files that are needed.

Usually this section in your program will look like this:

```
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/widget.h>
```

The order in which you place the header files is very important. Generally speaking, you should follow this format:

1. General header files, such as <stdio.h>.
2. OSF/Motif widget header files, beginning with <Xm.h> and including a header file for *each* widget class you are using in your program. The order of the widget class headers is not critical.
3. For each widget class you are using in your program, replace *widget* (shown in the #include statement above) with the name of the widget class.

The man page for each widget shows the exact spelling of all header files you need. The include files for all widgets are found in the directory /usr/include/Xm. For the PushButton widget in the sample program xmbutton, the header file name is PushB.h. Put a #include statement in your program for each type of widget you use. You need to include a header file only once, even if you use a given widget twice in your program. Don't forget to include any other header files (such as <stdio.h>) that your program may need.

Intrinsic.h defines the Xt structures and variables. Variables common to all OSF/Motif widgets are defined in Xm.h.

3.2 Initializing the Xt Intrinsics

You *must* initialize the XtIntrinsics before making any other calls to XtIntrinsics functions. The function XtInitialize establishes the connection to the display server, parses the command line that invoked the application, loads the resource database, and creates a "shell widget" to serve as the parent of your application widgets.

By passing the command line that invoked your application to XtInitialize, the function can parse the line to allow users to specify certain resources (such as fonts and

colors) for your application at run time. `XtInitialize` scans the command line and removes those options. The rest of your application sees only the remaining options.

The call to `XtInitialize` used by the sample program `xmbutton` is:

```
toplevel = XtInitialize("main", XMdemos, NULL, NULL, &argc, argv);
```

This line names the application shell "main," the application class "XMdemos," passes no additional options, and passes the command line that invoked the application. The first two parameters are used in setting up defaults files. Defaults files are explained in section 3.8, "Creating Defaults Files," later in this chapter.

The syntax of the `XtInitialize` function appears below. Note that it returns a value of type Widget; therefore, the variable `toplevel` in `xmbutton` must be defined as type Widget.

`Widget XtInitialize(`*shell_name*, *application_class*, *options*, *num_options*, *argc*, *argv*`)`

```
String              shell_name;
String              application_class;
XrmOptionDescRec    options[];
Cardinal            num_options;
Cardinal            *argc;
String              argv[];
```

shell_name Specifies the name of the application shell widget instance, which usually is something generic like "main." This name is used by the Xt Intrinsics to search for resources that belong specifically to this shell widget.

application_class Specifies the class name of this application, which usually is the generic name for all instances of this application. By convention, the class name is formed by reversing the case of the application's first letter. The class name is used to locate the files used to initialize the resource database. For example, the sample program "xmbutton" has a class name of "XMdemos."

options Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to `XrmParseCommand`.

num_options Specifies the number of entries in options list.

argc Specifies a pointer to the number of command line parameters.

argv Specifies the command line parameters.

There is an alternate function that you can use to initialize the Xt Intrinsics that is not as convenient as `XtInitialize`; however, it is more flexible because it lets you decide the type of shell you want to use. The function `XtToolkitInitialize` just initializes the

toolkit. It does not open the display or create an application shell. You must do this yourself using `XtOpenDisplay` and `XtAppCreateShell`. The advanced sample program presented in section 3.11 initializes the toolkit in this manner.

3.3 Creating Argument Lists for Widgets

The steps in sections 3.3 through 3.6 must be performed for each widget you wish to create.

Widgets accept argument lists (pairs of resource names and values) that control their appearance and functionality. The list of resources acceptable for a widget comprises not only resources unique to the widget, but also those resources inherited from other widgets. The resources for a given widget are shown in the man page for the widget.

The simplest way to set an element of an argument list is by using the `XtSetArg` macro. Other methods are described later in section 3.10, "Advanced Programming Techniques."

The program segment below declares an array `args` of up to 10 arguments. The size of the array is not important just so long as the number of elements allocated is not less than the number of elements used. The first argument specifies the label for the PushButton. The label is actually a pointer to a compound string that was created by a call to `XmStringCreateLtoR` earlier in the program. See chapter 8 for more information about compound strings. The last two arguments specify that the widget will have a width of 250 pixels and a height of 150 pixels. The third argument specifies the string to display in the PushButton.

```
Arg args[10];
XtSetArg(args[0], XmNlabelString, btn_text);
XtSetArg(args[1], XmNwidth, 250);
XtSetArg(args[2], XmNheight, 150);
```

An alternate method for `XtSetArg` uses a counter, n, rather than a hard-coded index. This method, shown below, makes it easier to add and delete argument assignments. It is the method used in the sample program `xmbutton`.

```
Arg args[10];
Cardinal n=0;
XtSetArg(args[0], XmNlabelString, btn_text); n++;
XtSetArg(args[n], XmNwidth, 250); n++;
XtSetArg(args[n], XmNheight, 150); n++;
```

The variable n contains the number of resources set. It can be passed to the widget create function (explained in section 3.5) as the argument list count.

CAUTION

Do not increment the counter from inside the call to `XtSetArg`. As currently implemented, `XtSetArg` is a macro that dereferences the first argument twice. This means that if you increment the counter from inside the call, it would actually be incremented twice for the one call.

The syntax for using `XtSetArg` is as follows:

```
XtSetArg(arg, name, value)
  Arg       arg;
  String    name;
  XtArgVal  value;
```

arg Specifies the name-value pair to set.

name Specifies the name of the resource.

value Specifies the value of the resource if it will fit in an `XtArgVal`, otherwise the address.

3.4 Creating the Widget

Now that you have established an argument list for the widget, you can create the widget instance. The call to `XtCreateManagedWidget` below comes from the sample program `xmbutton`.

```
button = XtCreateManagedWidget ("button", xmpushButtonWidgetClass,
toplevel, args, n);
```

This call names the newly created widget "button" and defines it to be a PushButton widget (from the class `xmPushButtonWidgetClass`). The class name "XmPushButton" or the name "button" can be used in defaults files (discussed in section 3.8) to refer to this widget. The PushButton's parent is "toplevel," the toplevel shell widget returned by `XtInitialize`. The argument list and number of arguments complete the call. This call will create the widget and notify its parent so that the parent can control its specific layout.

There is another way to create widgets, one that does not automatically manage them. Instead, you manage them when you want them to be displayed. Each widget has a "create function" associated with it. A create function creates the widget it is associated with but does not manage it. You You manage the widget with `XtManageChild`. The advanced

program in section 3.11 uses this method of creating widgets.

Widgets form a hierarchical structure called a widget tree. The widget tree for the program xmbutton is shown in figure 3-1. The widget returned by `XtInitialize` is the invisible parent for the toplevel application widget, in this case `button`. Usually there are several levels of widgets. Widgets at the higher levels are layout widgets (also called manager widgets) that control and coordinate the primitive widgets located at the leaves of the widget tree. The more advanced sample program later in this chapter illustrates multiple levels of widgets.

The syntax for `XtCreateManagedWidget` is described below.

```
Widget XtCreateManagedWidget(name, widget_class, parent, args, num_args)
       String      name;
       WidgetClass widget_class;
       Widget      parent;
       ArgList     args;
       Cardinal    num_args;
```

name Specifies the resource name for the created widget. This name is used for retrieving resources and should not be the same as any other widget that is a child of the same parent if unique values are necessary.

widget_class Specifies the widget class pointer for the created widget.

parent Specifies the parent widget.

args Specifies the argument list to override the resource defaults.

num_args Specifies the number of arguments in *args*. The number of arguments in an argument list can be automatically computed by using the `XtNumber` macro if the list is statically defined.

3.5 Adding Callback Procedures

Callbacks are one of the key features of the OSF/Motif widget set. They allow you to write procedures that will be executed when certain events occur within a widget. These events include mouse button presses, keyboard selections, and cursor movements. Callback procedures are the main mechanism your application uses to actually get things done.

You need to complete three steps to add callbacks:

1. Write the callback procedures.

2. Create an appropriate callback list.
3. Set the widget's callback argument.

Each of these steps is described in the following sections.

3.5.1 Writing a Callback Procedure

Callback procedures return no values, but have three arguments:

- The widget for which the callback is registered.
- Data passed to the callback procedure by the application.
- Data passed to the callback procedure by the widget.

In the sample program `xmbutton`, the callback procedure prints a message to the standard output device (usually the terminal window from which the application was invoked), frees the memory space used by `btn_text` (the PushButton label), and ends the program using the system `exit` call.

```
void activateCB(w, client_data, call_data)
 Widget    w;           /* widget id */
 caddr_t   client_data; /* data from application */
 caddr_t   call_data;   /* data from widget class */
{
/* print message and terminate program */
 fprint("PushButton selected.\n")
 XtFree(btn_text);
 exit(0);
}
```

The variable type `caddr_t` is defined by the Xt Intrinsics as a pointer to an area of memory. The `call_data` argument is used only by a few widgets. The man page for each widget specifies whether it passes any data to its callbacks.

The general syntax of a callback procedure is described below:

```
void CallbackProc(w, client_data, call_data)
      Widget  w;
      caddr_t client_data;
      caddr_t call_data;
```

w Specifies the widget for which this callback is invoked.

client_data Specifies the data that the widget should pass back to the client when the widget invokes the client's callback. This is a way for the client registering the callback to also define client-specific data to be passed to the client: a pointer to additional information about the widget, a reason

for invoking the callback, and so on. It is perfectly normal to have client_data be NULL if all necessary information is in the widget.

call_data Specifies any callback-specific data the widget wants to pass to the client. It is widget-specific and is usually set to NULL. It will be defined in the widget's man page if it is used.

3.5.2 Adding Callbacks

A callback contains information about the callback routine associated with a particular user action.

The sample program `xmbutton` creates a callback by calling the procedure `XtAddCallback`.

```
XtAddCallback (button, XmNactivate, activateCB, NULL);
```

The general syntax of `XtAddCallback` is described below:

```
void XtAddCallback(w, callback_name, callback, client_data)
      Widget          w;
      String          callback_name;
      XtCallbackProc  callback;
      caddr_t         client_data;
```

w Specifies the widget to add the callback to.

callback_name Specifies the callback list within the widget to append to.

callback Specifies the callback procedure to add.

client_data Specifies the client data to be passed to the callback when it is invoked by `XtCallCallbacks`. The client_data parameter is often NULL.

To add more callbacks, just make another call to `XtAddCallback`. In this way you can cause a user event to trigger many callback routines.

You can add a *list* of callbacks by using the function `XtAddCallbacks`.

The general syntax of `XtAddCallbacks` is described below:

```
void XtAddCallbacks(w, callback_name, callbacks)
      Widget          w;
      String          callback_name;
      XtCallbackList  callbacks;
```

w	Specifies the widget to add the callbacks to.
callback_name	Specifies the callback list within the widget to append to.
callbacks	Specifies the null-terminated list of callback procedures and corresponding client data to add.

3.5.3 Setting Widgets' Callback Resources

Many widgets define one or more callback resources. Set the value of the resource to the name of the callback list.

The callback resources for any particular widget are listed in the man page for that widget. The PushButton widget used in the sample program `xmbutton` supports three different kinds of callbacks. Each callback could be set up by specifying the callback list as the value of the appropriate resource.

- Callback(s) invoked when the PushButton widget is activated (argument `XmNactivateCallback`). This is the callback you use in `xmbutton`.
- Callback(s) invoked when the PushButton widget is armed (argument `XmNarmCallback`).
- Callback(s) invoked when the PushButton widget is disarmed (argument `XmNdisarmCallback`).

The translation table for this widget has been set such that an activate action occurs whenever the pointer is within the widget and the user presses mouse button 1. An activate action then causes the widget to invoke each of the callback routines on the callback list pointed to by its `XmNactivateCallback` argument. These routines are invoked in the order in which they appear in the callback list. In the case of the sample program `xmbutton`, only the routine `activateCB` is executed.

3.6 Making the Widget Visible

All widgets are now created and linked together into a widget tree.

`XtRealizeWidget` displays on the screen the widget that is passed to it and the children of that widget.

The final step in the program is to call the Xt Intrinsics routine that causes the application to enter a loop, awaiting action by the user.

Sample code for this section is:

```
XtRealizeWidget(toplevel);
XtMainLoop();
```

The above two statements from the sample program `xmbutton` display the push button widget and cause the program to enter a loop, waiting for user input. The main role of your application is the setting of widget arguments and the writing of callback procedures. Your application passes control to the Xt Intrinsics and the OSF/Motif widgets once the `XtMainLoop` function is called.

The syntax for `XtRealizeWidget` is shown below.

```
void XtRealizeWidget(w)
      Widget w;
```

w Specifies the widget.

3.7 Linking Libraries

When linking the program, be sure to include three libraries:

- `libXm.a`, which contains the OSF/Motif widgets.
- `libXt.a`, which contains the Xt Intrinsics.
- `libX11.a`, which contains the underlying Xlib library.

NOTE

The XmFileSelectionBox widget requires the `libPW.a` library. This can be included after `libX11.a`.

See section 1.5 of chapter 1 for information on compiling the programs in this chapter. The order in which you place the libraries is very important. The order shown above is correct, so be sure that you use the same order when linking in libraries.

3.8 Creating Defaults Files

Up to now, all widget resources have been set by the application using widget argument lists. An additional method for specifying resources is through a set of ASCII files that you can set up for your user. You may also want your user to set up these files to customize the application to individual requirements or preferences.

When writing a program, consider the following factors in deciding whether to specify an argument in a defaults file or in the program itself.

- Using a defaults file provides additional flexibility. Any user can override settings to reflect personal preferences, and a systems administrator can modify the application defaults file for system-wide customization.
- Specifying settings in the program gives the programmer greater control. They cannot be overridden.
- Using defaults files can speed application development. To change a resource value in a defaults file, simply edit the file (using any ASCII editor) and rerun the program. No recompilation or relinking is necessary.
- Using defaults files can simplify your program. Resources in defaults files are specified as strings. When resources are set in your program, they may have to be in some internal format that takes several calls to compute.
- Specifying options in your program may provide more efficient operation for the computer. The process of reading defaults files and interpreting their contents adds processing overhead.

Two files can be used for customization:

- A file located centrally in the directory `/usr/lib/X11/app-defaults` supplies defaults for an entire *class* of applications executing anywhere on the computer system. For example, the file `Mwm` provides default information for the OSF/Motif Window manager.
- A file (`.Xdefaults`,usually pronounced "dot Xdefaults") in each user's home directory can supply default values to all applications started by the user. The file `XMdemos` discussed earlier provides default information for the sample programs described throughout this manual.

All files are of the same format and there is a hierarchy that is observed by the system as described earlier. Chapter 5, "Customizing Your Local X Environment" in the manual *Using the X Window System* contains a detailed discussion of defaults files.

3.8.1 Application Defaults Files

These files are designed to be created by the applications developer or systems administrator. They are located in the directory `/usr/lib/X11/app-defaults` on the machine where the application resides. Application programs specify the file that contains the application defaults when they call `XtInitialize`. The `application_class` argument to that function specifies the name of the application defaults file. Several applications can point to the same file.

The call below (taken from the sample program `xmbutton`) will cause the Xt Intrinsics to look for the file `/usr/lib/X11/app-defaults/XMdemos` for default information.

```
toplevel = XtInitialize("main", "XMdemos", NULL, NULL, &argc, argv
```

The following sample defaults file sets the foreground color to white and background color to black.

```
*background: black
*foreground: white
```

3.8.2 User Defaults Files

Each user can create a `.Xdefaults` file in his or her home directory to specify resource defaults for applications run by that user. User defaults override application and system defaults and allow different users running the same program to specify personal display preferences, such as color and font selection.

The sample file below changes the background color to blue.

```
*background:  blue
```

3.8.3 Defaults File Example

The example below illustrates the interaction of the defaults files with each other and with arguments specified in programs.

Suppose a computer contains the program `xmbutton` as well as the application and user defaults files described above.

To determine the color of the background, the Xt Intrinsics will do the following:

1. Look for the *system* defaults and initialize the background color to `white`. (These defaults are compiled into the widgets.)
2. Look for the *application* defaults file `/usr/lib/X11/app-defaults/XMdemos` and set the color to black.
3. Look for the *user* defaults file `.Xdefaults` and set the background color to blue.

4. If the program sets the background argument (XmNbackground), this will override any defaults that may have been set.

3.9 Using Color

The OSF/Motif widgets have been designed to support both color and monochrome systems in a consistent and attractive manner. This is accomplished by incorporating into each widget a variety of visual attributes. Through proper use of these attributes, the widgets will present a dramatic three-dimensional appearance, giving you the distinct impression that you are directly manipulating the components. This section will describe these color attributes and show you how to use them.

3.9.1 Visual Capabilities and Attributes

The OSF/Motif widgets visual capabilities are based on specialized border and background drawing. The border drawing consists of a band around the widget. The band contains two regions:

- The top and left shadow.
- The bottom and right shadow.

The background drawing within the widget is referred to as background. Figure 3-3 illustrates the drawing areas.

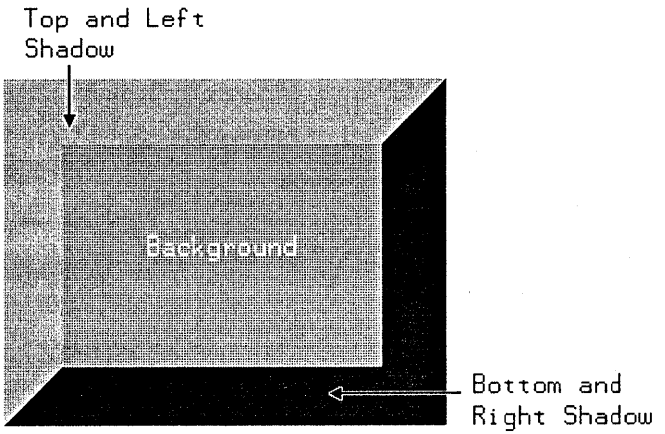


Figure 3-3. Widget Drawing Areas

Each area can be drawn from either a color or a pixmap. The top and left shadow is drawn using these Primitive widget resources:

- XmNtopShadowColor
- XmNtopShadowPixmap

The background is drawn using these Primitive widget resources:

- XmNbackground
- XmNbackgroundPixmap

The bottom and right shadow is drawn using these widget resources:

- XmNbottomShadowColor
- XmNbottomShadowPixmap

All the widgets support the visual attributes for setting the background as described. In general, only primitive widgets support the border drawing. To use the border drawing for manager widgets, a special manager widget, XmFrame, is available. This widget will maintain the geometry of a single child and perform the border and background drawing.

3.9.2 Using the Capabilities

When planning the three-dimensional appearance of your program's windows, consider the following guidelines:

- Any selectable area should appear to be raised.
- Non-selectable areas should appear to be flat. This can be accomplished by setting XmNshadowThickness to 0.

To give the impression that the widget is raised above its parent, set these resources to the value shown:

- Set XmNtopShadowColor to a light color.
- Set XmNbackground to a medium color.
- Set the XmNbottomShadowColor to a dark color.

The foreground, background, and both top and bottom shadow resources are dynamically defaulted. This means that if you do not specify any color for these resources, colors are automatically generated. A black and white color scheme is generated on a monochrome system, while on a color system a set of four colors is generated that displays the correct shading to achieve the three-dimensional appearance. If you specify only a background color, the foreground and both shadow colors are generated (based on the background color) to achieve the three-dimensional appearance. This color generation works best with non-saturated colors, that is, using pure red, green, or blue will give poor results. Also,

colors are only generated at the time of the widget's creation. Changing the background color by using XtSetValues will *not* cause the other colors to be regenerated.

Reversing the top shadow and bottom shadow colors will give the appearance that the widget is set into its parent. Several of the primitive widgets (buttons, toggles, and arrows, for example) automatically reverse their shadowing when selected to achieve the effect of being pressed. They return to their original shadowing when released.

Use coordinated colors such as light blue for the top shadow color, sky blue for the background color, and navy blue for the bottom shadow color to enhance the three-dimensional appearance. Using dissimilar colors loses the effect.

The three-dimensional appearance is more difficult to achieve on monochrome systems. The built-in defaults for all the widgets have been set up for monochrome systems and provide the desired effect. The top shadow is drawn with a 50 percent pixmap, the background is solid white, and the bottom shadow is solid black. This appearance can be further enhanced by setting the background of a manager containing a set of raised children to a pixmap of 25 percent black and 75 percent white.

3.10 Advanced Programming Techniques

The sample program xmbutton described in earlier sections of this chapter illustrated the writing of a very simple widget program. The Xt Intrinsics provide additional mechanisms for programmers.

3.10.1 Setting Argument Values

Section 3.3 described the use of XtSetArg for setting the values of widget arguments. This section describes three additional methods. The code segments show how the earlier sample program could have been rewritten to use the new methods.

Assigning Argument Values

Each element of the type Arg structure can be assigned individually.

```
XmString btn_text;
Arg args[10];

btn_text = XmStringCreateLtoR ("Push Here",
  XmSTRING_DEFAULT_CHARSET);
args[0].name = XmNwidth;
args[0].value = (XtArgVal) 250;
args[1].name = XmNheight;
args[1].value = (XtArgVal) 150;
```

```
args[2].name = XmNlabelString;
args[2].value = (XtArgVal) btn_text;
```

Be sure to keep name-value pairs synchronized. Note that all argument values have been cast to type `XtArgVal`.

Static Initializing

Initializing argument lists at compile time makes it easy to add and delete argument settings in your program. It avoids the need to hard-code the maximum number of arguments when declaring your argument list. These settings are frozen at compile time, however. While the example below shows only a single argument list being created, you can create any number of lists (be sure to declare each list as type `Arg`). Note that you cannot use static initializing to initialize a compound string. You can combine static initializing with run-time assignments to accomplish this, as shown in the next section.

```
static Arg args[] = {
 {XmNwidth, (XtArgVal) 250},
 {XmNheight, (XtArgVal) 150},
};
```

Note that the values of each argument have been cast to variable type `XtArgVal`. When the create widget function is invoked, passing it `XtNumber(args)` will compute the number of elements in the argument list.

```
button = XtCreateManagedWidget("button", xmPushButtonWidgetClass,
  toplevel, args, XtNumber(args));
```

NOTE

Use the macro `XtNumber` *only* if you are declaring the argument list of indefinite size as shown above (`args[]`). `XtNumber` will return the number of elements that have actually been allocated in program memory.

Combining Static Initialization with Run-Time Assignments

The final method for creating argument lists initializes a list at compile time (described in "Static Initializing" above) and then modifies the values of the settings using regular assignment statements. The `XtNumber` macro can be used to count the number of arguments, since the argument list is declared with no definite number of arguments. The values can be changed through assignments at run time, but the size of the argument list

(the number of arguments that can be specified) is frozen at compile time and cannot be extended.

The example below initializes an argument list of three elements. The last is initialized to NULL so it can be given a value later. The value for argument XmNheight is changed in the program from its initialized value of 150 to a run-time value of 250.

```
XmString btn_text;

static Arg args[] = {
 {XmNwidth, (XtArgVal) 500}, /* item 0 */
 {XmNheight, (XtArgVal) 150}, /* item 1 */
 {XmNlabelString, (XtArgVal) NULL}, /* item 2 */
};

btn_text = XmStringCreateLtoR ("Push Here",
  XmSTRING_DEFAULT_CHARSET);

args[1].value = (XtArgVal) 250;
args[2].value = (XtArgVal) btn_string;
```

3.10.2 Manipulating Created Widgets

Widget programs to this point have set up argument lists and callbacks for widgets prior to the widgets' creation. You can also modify widgets after they have been created. Such modification usually occurs in callback routines and is illustrated in the sample program xmfonts discussed later in this chapter.

Retrieving and Modifying Arguments

XtGetValues will return the current value of specified arguments for a created widget. XtSetValues will change the value of specified arguments.

Adding Callbacks and Translations

XtAddCallback will add a callback routine to a widget's callback list after the widget has been created.

Each widget has a translation table that ties user actions (for example, button presses and keyboard presses) to widget actions. Your application can modify the translation table for any widget. This process is described in any manual on the Xt Intrinsics.

Separating Widget Creation and Management

By using `XtCreateManagedWidget`, the sample program automatically adds the newly created widget to its parent's set of managed children. To optimize programs that add a number of widgets to a single parent, you may want to create the widgets using `XtCreateWidget` calls and then add the entire list of children to its parent with a single `XtManageChildren` call. In this way, the parent widget performs its geometry processing of its children only once. This will increase the performance of applications that have a large number of child widgets under a single parent.

Usually, the function `XtRealizeWidget` will display a widget and all of its children. Using the function `XtSetMappedWhenManaged` allows you to turn off automatic mapping (displaying) of particular widgets. Your application can then use `XtMapWidget` to display the widget.

The function `XtDestroyWidget` will destroy a created widget and its children. The destroyed widget is automatically removed from its parent's list of children.

3.11 An Advanced Sample Program

The program presented in this section, `xmfonts`, displays each available font (fonts are found in the directory `/usr/lib/X11/fonts`) as a pushbutton. The source code and the application defaults file for this sample program are listed later in this section. They are located on your system in `/usr/contrib/Xm/xmfonts.c` and `/usr/contrib/Xm/XMdemos`.

You can change the background and foreground colors and other visual attributes by changing the parameters in the app-defaults file `XMdemos`. Remember that `XMdemos` is used as a defaults file for all the example programs in this manual. If you change any of the general defaults at the top of the file, other programs will be affected.

When you run the program, you will see the window shown in figure 3-4.

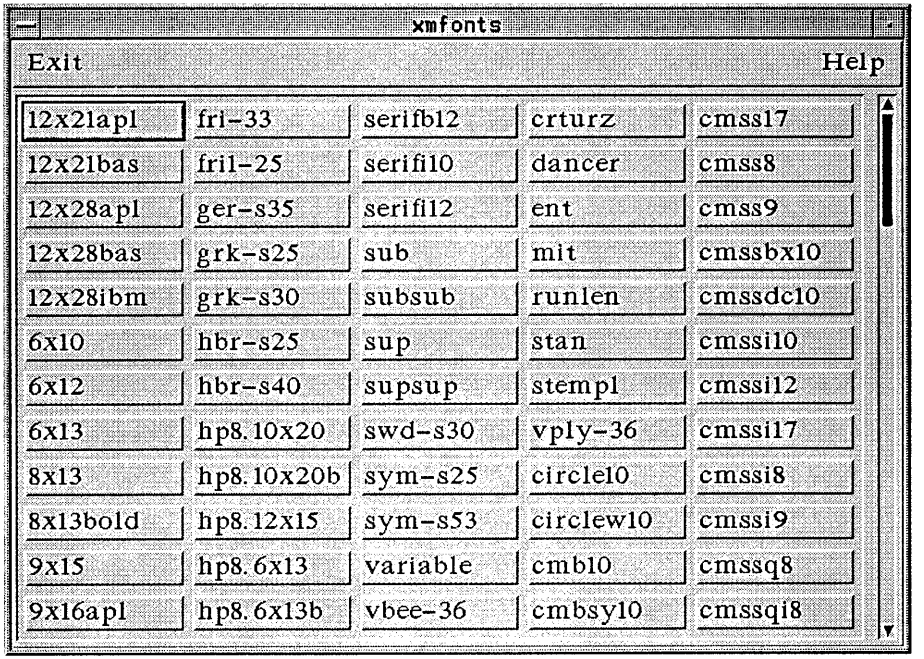


Figure 3-4. Program xmfonts Main Window

Move the pointer to the PushButton representing the font you want to see displayed and press mouse button 1. Text in the selected font is displayed in a separate popup window (see figure 3-6). This window can be removed by pressing the close PushButton or left on the screen to be compared with other text windows that you might select. You can continue this procedure for as long as you desire. Each time the mouse button is pressed, the selected font will be displayed in a separate popup window. When you want to exit the program, move the cursor to the Exit button in the MenuBar, then drag the pointer down until the Quit button appears. Clicking mouse button 1 on the Quit button will terminate the program.

3.11.1 Windows Used in xmfonts

There are three independent windows displayed in this program (see figure 3-4, figure 3-5, and figure 3-6):

Main Window

The main window is the window in which the PushButtons are displayed (see figure 3-4). It is a combination of an application shell, a MainWindow widget, a RowColumn widget, and a number of PushButtonGadgets. The MainWindow widget was chosen because it has MenuBar capability and is a convenient envelope for many applications. Although MainWindow can have three areas (see chapter 2), only two of the areas are needed here, the MenuBar and work region. In this case the MenuBar is the parent of a Pulldown menu for the exit function and a CascadeButton for the "Help" function. The work region consists of a RowColumn widget and a vertical ScrollBar. A number of PushButtonGadgets, one for each font, are placed within the RowColumn widget. These are used instead of PushButtons to improve program performance. To see the difference for yourself, run the program as it exists. Use the ScrollBar to view buttons not displayed. Then change the code in the file xmfonts.c (be sure to move this file to your work directory first) so that the line that now reads

```
button = XmCreatePushButtonGadget(row_column, name, args, n);
```

becomes

```
button = XmCreatePushButton(row_column, name, args, n);
```

Then recompile the program and run it again. You should see considerable difference in the operation of the program, particularly when scrolling through the buttons.

Help Window

The "Help" window is a popup window that is a MessageBox (see figure 3-5).

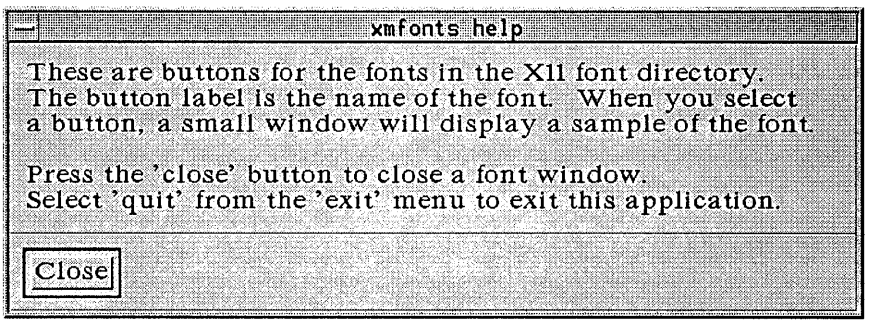


Figure 3-5. Program xmfonts Help Display Window

Font Display Window

The window that displays the selected font is also a popup window that is a MessageBox Dialog (see figure 3-6).

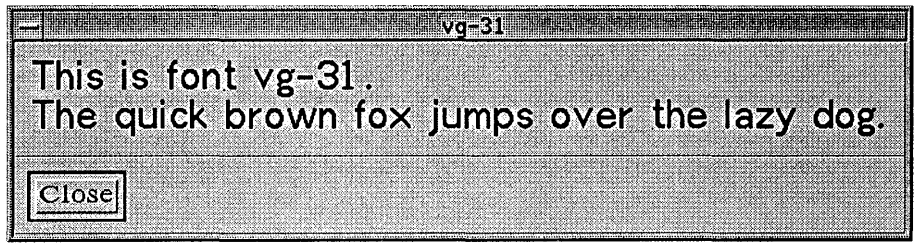


Figure 3-6. Program xmfonts Font Display Window

You can have as many text display windows as you want. You can remove them all by simply exiting the program as explained above, or you can remove each window individually by moving the pointer to the close button on the window and pressing mouse button 1.

3.11.2 Widget Hierarchy

This program produces three separate windows. One contains all the PushButtonGadgets and its shell is created using `XtToolkitInitialize`, `XtOpenDisplay`, and finally `XtAppCreateshell`. Note the difference between this program and `xmbutton`. Since `XtInitialize` opens the display and creates a shell in addition to initializing the toolkit, `xmbutton` did not need to use the functions `XtOpenDisplay` and `XtAppCreateShell`. The other two windows are the "Help" window and the window that displays text in the selected font. Both of these windows are MessageBoxDialogs created by the function `XmCreateMessageDialog`. This function creates a MessageBox widget and a DialogShell widget. The widget tree for `xmfonts` is shown in figure 3-7.

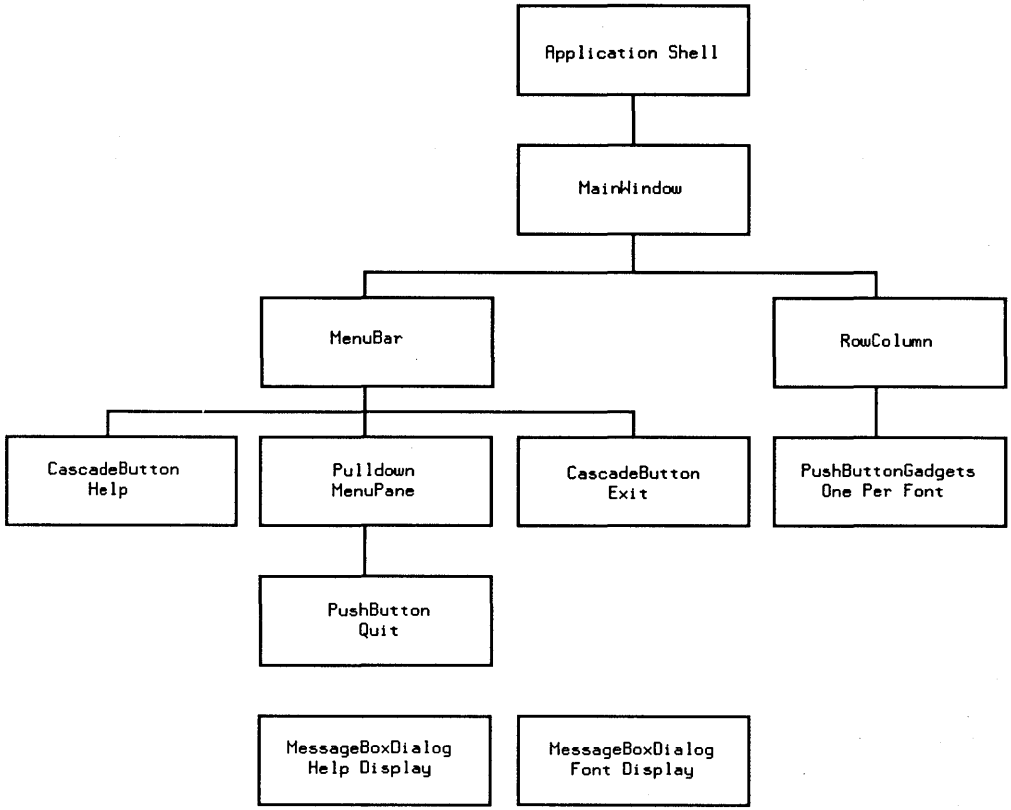


Figure 3-7. xmfonts Widget Tree

3.11.3 Source Code

The source code for `xmfonts` and the default file `XMdemos` are listed in the following sections.

The Program

```
/**---------------------------------------------------------
***
*** file:xmfonts.c
***
*** project:OSF/Motif widgets example programs
***
*** description: This program creates a button for every font in
***              /usr/lib/X11/fonts.  When a button is selected,
***              a text sample is displayed using the font.
***
***
*** (c) Copyright 1989 by Open Software Foundation, Inc.
***    All Rights Reserved.
***
*** (c) Copyright 1989 by Hewlett-Packard Company.
***
***
*** defaults: xfonts.c depends on these defaults:
!
*foreground:           white
*fontList:             vr-20
*allowShellResize:     true
*borderWidth:          0
*highlightThickness:   2
*traversalOn:          true
*keyboardFocusPolicy:  explicit
*menuAccelerator:      <Key>KP_F2
!
xmfonts*XmScrolledWindow.height:  432
xmfonts*XmScrolledWindow.width:   690
xmfonts*menu_bar*background:      #58f
!
***---------------------------------------------------------*/

/*---------------------------------------------------------
** Include Files
```

```
*/
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#ifdef BSD
#include <sys/dir.h>
#else
#include <dirent.h>
#endif

#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/CoreP.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/BulletinB.h>
#include <Xm/CascadeB.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h
#include <Xm/ScrollBar.h>
#include <Xm/ScrolledW.h>

/*--------------------------------------------------------------
** Forward Declarations
*/

void main ();                   /*  main logic for application*/

Widget CreateApplication ();    /*  create main window*/
Widget CreateFontSample ();     /*  create font display window*/
Widget CreateHelp ();           /*  create Help window*/

void SelectFontCB ();           /*  callback for font buttons*/
void CloseCB ();                /*  callback for close button*/
void HelpCB ();                 /*  callback for Help button*/
void QuitCB ();                 /*  callback for quit button*/

/*--------------------------------------------------------------
```

```
** Global Variables
*/

#define MAX_ARGS 20
#define FONT_DIR_NAME "/usr/lib/X11/fonts"
#define TITLE_STRING "X Font Sampler"

static XmStringCharSetcharset =
  (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;

/*--------------------------------------------------------------
** main- main logic for application
*/
void main (argc,argv)
unsigned int  argc;
char          **argv;
{
Display       *display;    /*  Display          */
Widget        app_shell;   /*  ApplicationShell */
Widget        main_window; /*  MainWindow       */


/* Initialize toolkit and open the display.
*/
XtToolkitInitialize();
display = XtOpenDisplay (NULL, NULL, argv[0], "XMdemos", NULL, 0,
  &argc, argv);

if (!display) {
  XtWarning ("xfonts: Can't open display, exiting...");
  exit (0);
}

/*  Create application shell.
*/
app_shell = XtAppCreateShell (argv[0], "XMdemos",
applicationShellWidgetClass, display, NULL, 0);

/*  Create and realize main application window.
*/
main_window = CreateApplication (app_shell);
XtRealizeWidget (app_shell);
```

```
/*  Get and dispatch events.
*/
XtMainLoop ();
}

/*--------------------------------------------------------------
**  CreateApplication- create main window
*/
Widget CreateApplication (parent)
Widget   parent;          /*  parent widget*/
{
Widget   main_window;     /*  MainWindow*/
Widget   menu_bar;        /*  MenuBar*/
Widget   menu_pane;       /*  MenuPane*/
Widget   cascade;         /*  CascadeButton*/
Widget   frame;           /*  Frame*/
Widget   swindow;         /*  ScrolledWindow*/
Widget   row_column;      /*  RowColumn*/
Widget   button;          /*  PushButtonGadget*/
Widget   hsb, vsb;        /*  ScrollBars*/

Arg      args[MAX_ARGS]; /*  arg list*/
register intn;           /*  arg count*/

DIR  *dirp;              /*  directory pointer*/
#ifdef BSD
  struct direct  *item;  /*  entry in directory*/
#else
  struct dirent  *item;  /*  entry in directory*/
#endif
char      name[15];      /*  name string*/
int       len;           /*  string length*/

XmString  label_string;


/*  Create MainWindow.
*/
n = 0;
main_window = XmCreateMainWindow (parent, "main1", args, n);
XtManageChild (main_window);

/*  Create MenuBar in MainWindow.
```

```
*/
n = 0;
menu_bar = XmCreateMenuBar (main_window, "menu_bar", args, n);
XtManageChild (menu_bar);

/*  Create "Exit" PulldownMenu.
*/
n = 0;
menu_pane = XmCreatePulldownMenu (menu_bar, "menu_pane",
  args, n);

n = 0;
button = XmCreatePushButton (menu_pane, "Quit", args, n);
XtManageChild (button);
XtAddCallback (button, XmNactivateCallback, QuitCB, NULL);

n = 0;
XtSetArg (args[n], XmNsubMenuId, menu_pane);  n++;
cascade = XmCreateCascadeButton (menu_bar, "Exit", args, n);
XtManageChild (cascade);

/*  Create "Help" button.
*/
n = 0;
cascade = XmCreateCascadeButton (menu_bar, "Help", args, n);
XtManageChild (cascade);
XtAddCallback (cascade, XmNactivateCallback, HelpCB, NULL);

n = 0;
XtSetArg (args[n], XmNmenuHelpWidget, cascade);  n++;
XtSetValues (menu_bar, args, n);

/*  Create Frame MainWindow and ScrolledWindow in Frame.
*/
n = 0;
XtSetArg (args[n], XmNmarginWidth, 2);  n++;
XtSetArg (args[n], XmNmarginHeight, 2);  n++;
XtSetArg (args[n], XmNshadowThickness, 1);  n++;
XtSetArg (args[n], XmNshadowType, XmSHADOW_OUT);  n++;
frame = XmCreateFrame (main_window, "frame", args, n);
XtManageChild (frame);

n = 0;
```

```
XtSetArg (args[n], XmNscrollBarDisplayPolicy,
  XmAS_NEEDED);  n++;
XtSetArg (args[n], XmNscrollingPolicy, XmAUTOMATIC);  n++;
swindow = XmCreateScrolledWindow (frame, "swindow", args, n);
XtManageChild (swindow);

/*  Create RowColumn in ScrolledWindow to manage buttons.
*/
n = 0;
XtSetArg (args[n], XmNpacking, XmPACK_COLUMN);  n++;
XtSetArg (args[n], XmNnumColumns, 5);  n++;
row_column = XmCreateRowColumn (swindow, "row_column", args, n);
XtManageChild (row_column);

/* Set MainWindow areas and add tab groups
*/
XmMainWindowSetAreas (main_window, menu_bar, NULL, NULL, NULL,
  frame);
n = 0;
XtSetArg (args[n], XmNhorizontalScrollBar, &hsb);  n++;
XtSetArg (args[n], XmNverticalScrollBar, &vsb);  n++;
XtGetValues (main_window, args, n);
XmAddTabGroup (row_column);
if (hsb)
  XmAddTabGroup (hsb);
if (vsb)
  XmAddTabGroup (vsb);

/*  Create a PushButton widget for each font.
*/
/*  open the font directory  */
dirp = opendir (FONT_DIR_NAME);
/*  read one entry each time through the loop  */
for (item = readdir (dirp);  item != NULL;  item = readdir (dirp))
{
  len =  (strlen (item -> d_name));
/*  discard entries that don't end in ".xxx"  */
  if ((len < 5) || (item -> d_name[len-4] != '.')) continue;
/*  copy the name (except extension) from the entry  */
  strncpy (name, item -> d_name, len-4);
  name[len-4] = ' ';
/*  create PushButton in RowCol  */
  n = 0;
```

```
  label_string = XmStringCreateLtoR(name, charset);
  XtSetArg (args[n], XmNlabelString, label_string);  n++;
  button = XmCreatePushButtonGadget (row_column, name, args, n);
  XtManageChild (button);
  XtAddCallback (button, XmNarmCallback, SelectFontCB, NULL);
  XmStringFree (label_string);
}

/  *Return MainWindow.
*/
return (main_window);
}

/*-------------------------------------------------------------
**  CreateFontSample- create font display window
*/
Widget CreateFontSample (parent)
Widget      parent;             /*  parent widget     */
{
Widget      message_box;        /*  MessageBox Dialog */
Widget      button;
Arg         args[MAX_ARGS];     /*  arg list          */
register    intn;               /*  arg count         */

char        *name = NULL;       /*  font name         */
XFontStruct *font = NULL;       /*  font pointer      */
XmFontList  fontlist = NULL;    /*  fontlist pointer  */
static char message[BUFSIZ];    /*  text sample       */
XmString    name_string = NULL;
XmString    message_string = NULL;
XmString    button_string = NULL;


/*  Get font name.
*/
n = 0;
XtSetArg (args[n], XmNlabelString, &name_string); n++;
XtGetValues (parent, args, n);
XmStringGetLtoR (name_string, charset, &name);


/*  Load font and generate message to display.
*/
```

```
if (name)
  font = XLoadQueryFont (XtDisplay (XtParent (parent)), name);
if  (!font)
  sprintf (message, "Unable to load font: %s\0", name);
else
{
  fontlist = XmFontListCreate (font, charset);
  sprintf (message, "  This is font %s.\n\
  The quick brown fox jumps over the lazy dog.\0", name);
}
message_string = XmStringCreateLtoR (message, charset);
button_string = XmStringCreateLtoR ("Close", charset);

/*  Create MessageBox dialog.
*/
n = 0;
if (fontlist)
{
XtSetArg (args[n], XmNlabelFontList, fontlist);  n++;
}
XtSetArg (args[n], XmNdialogTitle, name_string);  n++;
XtSetArg (args[n], XmNokLabelString, button_string);  n++;
XtSetArg (args[n], XmNmessageString, message_string);  n++;
message_box = XmCreateMessageDialog (XtParent (XtParent(parent)),
  "fontbox", args, n);

button = XmMessageBoxGetChild (message_box,
  XmDIALOG_CANCEL_BUTTON);
XtUnmanageChild (button);
button = XmMessageBoxGetChild (message_box,
  XmDIALOG_HELP_BUTTON);
XtUnmanageChild (button);

/*  Free strings and return MessageBox.
*/
if (fontlist) XtFree (fontlist);
if (name_string) XtFree (name_string);
if (message_string) XtFree (message_string);
if (button_string) XtFree (button_string);
return (message_box);
}

/*--------------------------------------------------------------
```

```
**  CreateHelp- create Help window
*/
Widget CreateHelp (parent)
Widget      parent;         /*  parent widget  */
{
Widget      button;
Widget      message_box;    /*  Message Dialog */
Arg         args[MAX_ARGS]; /*  arg list       */
register    intn;           /*  arg count      */

static char message[BUFSIZ]; /*  Help text     */
XmString    title_string = NULL;
XmString    message_string = NULL;
XmString    button_string = NULL;

/*  Generate message to display.
*/
sprintf (message, "\
These are buttons for the fonts in the X11 font directory.  \n\
The button label is the name of the font.  When you select \n\
a button, a small window will display a sample of the font.  \n\n\
Press the 'close' button to close a font window.  \n\
Select 'quit' from the 'exit' menu to exit this application.\0");
message_string = XmStringCreateLtoR (message, charset);
button_string = XmStringCreateLtoR ("Close", charset);
title_string = XmStringCreateLtoR ("xmfonts help", charset);


/*  Create MessageBox dialog.
*/
n = 0;
XtSetArg (args[n], XmNdialogTitle, title_string);  n++;
XtSetArg (args[n], XmNokLabelString, button_string);  n++;
XtSetArg (args[n], XmNmessageString, message_string);  n++;
message_box = XmCreateMessageDialog (parent, "helpbox", args, n);

button = XmMessageBoxGetChild (message_box,
  XmDIALOG_CANCEL_BUTTON);
XtUnmanageChild (button);
button = XmMessageBoxGetChild (message_box,
  XmDIALOG_HELP_BUTTON);
XtUnmanageChild (button);
```

```
/*  Free strings and return MessageBox.
*/
if (title_string) XtFree (title_string);
if (message_string) XtFree (message_string);
if (button_string) XtFree (button_string);
return (message_box);
}


/*--------------------------------------------------------------
**  SelectFontCB- callback for font buttons
*/
void SelectFontCB (w, client_data, call_data)
Widget   w;             /* widget id              */
caddr_t  client_data;   /* data from application  */
caddr_t  call_data;     /* data from widget class */
{
Widget   message_box;


/*  Create font sample window.
*/
message_box = CreateFontSample (w);


/*  Display font sample window.
*/
XtManageChild (message_box);
}


/*--------------------------------------------------------------
**  CloseCB- callback for close button
*/
void CloseCB  (w, client_data, call_data)
Widget       w;            /* widget id              */
caddr_t      client_data;  /* font pointer           */
caddr_t      call_data;    /* data from widget class */
{
XFontStruct  *font= (XFontStruct *) client_data;
Widget       message_box= XtParent (w);
Widget       shell= XtParent (message_box);
```

```
/*  Free font.
*/
if (font) XFreeFont (XtDisplay (w), font);


/*  Unmanage and destroy widgets.
*/
XtUnmanageChild (message_box);
XtDestroyWidget (shell);
}


/*--------------------------------------------------------------
**  HelpCB- callback for Help button
*/
void HelpCB (w, client_data, call_data)
Widget   w;           /*  widget id               */
caddr_t  client_data; /*  data from application   */
caddr_t  call_data;   /*  data from widget class  */
{
Widget   message_box; /*  MessageBox              */


/*  Create Help window.
*/
message_box = CreateHelp (w);


/*  Display Help window.
*/
XtManageChild (message_box);
}


/*--------------------------------------------------------------
**  QuitCB- callback for quit button
*/
void QuitCB (w, client_data, call_data)
Widget   w;           /*  widget id               */
caddr_t  client_data; /*  data from application   */
caddr_t  call_data;   /*  data from widget class  */
{
```

```
/*  Terminate the application.
*/
exit (0);
}
```

The Defaults File

This file should be placed in the directory `/usr/lib/X11/app-defaults` as a part of `XMdemos`.

```
!
!XMdemos app-defaults file for OSF/Motif demo programs
!
!general appearance and behavior defaults
!
*foreground:            white
*fontList:              vr-20
*allowShellResize:      true
*borderWidth:           0
*highlightThickness:    2
*traversalOn:           true
*keyboardFocusPolicy:   explicit
*menuAccelerator:       <Key>KP_F2
!
xmfonts*XmScrolledWindow.height:  432
xmfonts*XmScrolledWindow.width:   690
xmfonts*menu_bar*background:      #58f
!
```

Shell Widgets 4

Shell widgets are used to provide communication between the widgets in an application and the window manager. An application's widgets are arranged in a hierarchy, with upper-level widgets acting as the parents of lower- level widgets. Widgets at the top of the hierarchy do not have normal parent widgets, but have a Shell as the parent. Different Shell widgets are provided for the various categories of top-level widgets, including Dialogs and MenuPanes. Figure 4-1 shows the hierarchy of the Shell widgets. Keep in mind that Shell is a subclass of Composite (see figure 1-4 in chapter 1).

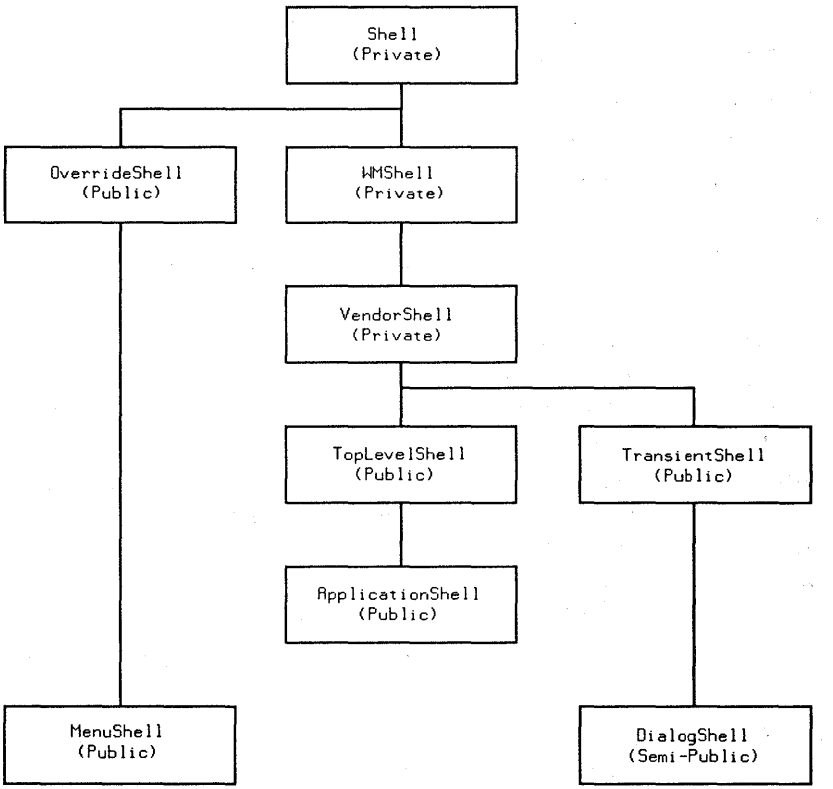


Figure 4-1. Shell Widget Hierarchy

The classes Shell, WmShell, and VendorShell are private and should not be instantiated. The other Shell classes are for public use, although a DialogShell is normally created by a convenience function as part of a set rather than by itself. Each of the Shell classes has a man page (man pages can be found in the *HP OSF/Motif Programmer's Reference Manual*) that has information on the resources belonging to the specific Shell widget.

4.1 Descriptions of Shell Widgets

The Shell widgets shown in figure 4-1 are of two types: Private and Public. This means that the Public widgets are those that you should instantiate, either individually or, as in the case of DialogShell, as part of a set. The Private widgets are those that you should *not* instantiate. These widgets typically just supply resources to Shell widgets that are lower in the hierarchy. Keep in mind the hierarchy diagram of figure 4-1 as you study the definitions of the Shell widgets.

- **Shell** - Shell is the "base" class for all shell widgets. It provides the fields needed by all the Shell widgets. Shell is a subclass of Composite (see figure 1-2).
- **OverrideShell** - OverrideShell is used for shell windows that bypass the window manager. Popup menus are one example of where an OverrideShell might be used.
- **WMShell** - WMShell contains fields that are needed by the common window manager protocol.
- **VendorShell** - VendorShell contains fields that are used by vendor-specific window managers.
- **TopLevelShell** - TopLevelShell is used for normal top-level windows. It is *not* the root shell used by an application, rather it is normally used to create "peer" top-level windows in situations where an application needs more than one set of windows. The root shell is normally the ApplicationShell.
- **ApplicationShell** - ApplicationShell is an application's top-level or root window. This is the shell that is created by `XtInitialize`. An application should not have more than one ApplicationShell. Subsequent top-level shells should be of class TopLevelShell and are created by `XtAppCreateShell`. These top-level shells can be considered the root of a second widget tree for the application.
- **MenuShell** - MenuShell is used as the parent of Popup and Pulldown MenuPanes. It is a subclass of OverrideShell.
- **DialogShell** - DialogShell is the parent of Dialogs. Although it can be instantiated by itself, it is normally instantiated as part of a set by one of the convenience dialogs. For example, `XmCreateErrorDialog` creates a DialogShell and a MessageBox as its child. See chapter 5, "Dialog Widgets and Functions," for more information.

4.2 Shell Widget Appearance

Most Shell widgets are invisible, however the type of Shell class can have an impact on how its children are displayed. For example, children of a TransientShell (typically Dialogs) by default have no buttons on the window manager frame that surrounds the window. Also, as long as the transient window is visible, it will remain above the window from which it is transient.

Dialog Widgets and Functions 5

Dialog widgets are container widgets that provide a means of communicating between the user of an application and the application itself. A Dialog widget will normally ask a question or present some information to the user. In some cases, the application is suspended until the user provides a response.

This chapter explains the Dialog widgets that are available and how they can be used in your application.

5.1 Dialog Widgets and Menus

There are two types of Dialog widgets: single-reply and multiple-reply. A single-reply Dialog widget consists of a single question, and a single reply is expected. A multiple-reply Dialog widget consists of a number of questions that require a number of responses. Generally speaking, a single-reply Dialog widget is **modal** in nature. This means that a reply is required before the application can continue. A multiple-reply Dialog widget is usually **modeless**. It does not require a reply and does not stop the progress of the application.

There are many similarities between Dialog widgets and menus, and it may be somewhat confusing to understand when to use a Dialog widget and when to use a menu. You should understand the differences between dialog widgets and menus in order to make this determination.

A menu is short-lived. It exists only while a selection is being made, then it disappears. A Dialog widget stays visible until it is told to disappear.

A menu is usually modal. Until the user of the application makes some selection on the menu, interaction with any other part of the application is not possible. Since multiple-reply Dialog widgets are modeless, interaction with other parts of the application is possible. Thus, if a modeless state is required, a multiple-reply or modeless single-reply Dialog widget should be used instead of a menu. A menu is faster when there is a need to identify current settings or make a single selection. When multiple selections are needed, a menu disappears after each selection and has to be displayed over and over, so in this case a Dialog widget would be better suited.

5.2 A List of the Dialog Widgets

The following list identifies the Dialog widgets by name and provides a brief description of each widget's function. Each widget will be described in more detail in later sections of this chapter. Additional information can be found in the man page for the respective widget. Man pages are contained in the *HP OSF/Motif Programmer's Reference Manual.*

- `XmDialogShell`. This widget is used as the parent for all Dialogs. It is automatically created by Dialog "convenience functions" (described in a later section).
- `XmBulletinBoard`. This is a composite widget that provides simple geometry management for its child widgets. It is the base widget for most Dialog widgets, but is also used as a container widget.
- `XmCommand`. This widget is a subclass of the `XmSelectionBox` widget. It includes a command line input text field, a command line prompt, and a command history region.
- `XmFileSelectionBox`. This widget is used to traverse through file system directories. You can view the files in the directories and then select a single file on which you intend to perform some action.
- `XmForm`. The Form widget is a constraint-based Manager widget that establishes and maintains spatial relationships between its children. These relationships are maintained even though the Form or any of its children are resized, unmanaged, remanaged, or destroyed.
- `XmMessageBox`. This widget is used to pass information to the user of an application. It contains up to three standard PushButtons ("OK," "Cancel," and "Help" by default), a message symbol, and the message itself.
- `XmSelectionBox`. This widget provides the capability of selecting a single item from a list of items.

5.3 Convenience Dialogs

We have used the term "Dialog widget" to describe a particular type of widget. When used by itself, the word "Dialog" takes on a special meaning. A "Dialog" is a collection of widgets that are used for a specific purpose. A Dialog normally consists of a DialogShell, some BulletinBoard resources, and various other widgets such as Label, PushButton, and Text. The collection of widgets that forms a Dialog can be built from "scratch" using argument lists and creating each individual widget of the Dialog. However, there is an easier, more convenient method to create the Dialog. Functions (called, appropriately,

"Dialog convenience functions") exist that create the collection of widgets that make up the Dialog in one step. The Dialogs that are created by these convenience functions are referred to as "convenience Dialogs." The following table identifies the convenience dialogs that are available.

TABLE 5-1. Convenience Dialogs

Convenience Dialog	Definition	Convenience Function
BulletinBoardDialog	Used for interactions that are not supported by the standard dialog set.	XmCreateBulletinBoardDialog
ErrorDialog	This Dialog instantiates a MessageBox and a DialogShell. It uses a message and a symbol (circle with backslash) to warn the user that an error has occurred.	XmCreateErrorDialog
FileSelectionDialog	This Dialog instantiates a FileSelectionBox and a DialogShell.	XmCreateFileSelectionDialog.
FormDialog	This Dialog instantiates a Form and a DialogShell.	XmCreateFormDialog.
InformationDialog	This Dialog instantiates a MessageBox and a DialogShell. This Dialog provides information to the user and it has a symbol that consists of a large lower-case letter "i" that is positioned on the left side of the MessageBox.	XmCreateInformationDialog.
PromptDialog	This Dialog instantiates a SelectionBox and a DialogShell. It is used to prompt the user for input.	XmCreatePromptDialog.

TABLE 5-1. Convenience Dialogs (Continued)

Convenience Dialog	Definition	Convenience Function
QuestionDialog	This Dialog instantiates a MessageBox and a DialogShell. It is used to get an answer from the user. It has a symbol that consists of a large question mark that is positioned on the left side of the MessageBox.	XmCreateQuestionDialog.
SelectionDialog	This Dialog instantiates a SelectionBox and a DialogShell.	XmCreateSelectionDialog.
WarningDialog	This Dialog instantiates a MessageBox and a DialogShell. It is used to warn the user of some potential danger. It has a symbol that consists of a large exclamation point contained within a triangle. The symbol is positioned on the left side of the MessageBox.	XmCreateWarningDialog.
WorkingDialog	This Dialog instantiates a MessageBox and a DialogShell. It is used to inform the user that a potentially time-consuming operation is in progress. It has a symbol that is an hourglass positioned on the left side of the MessageBox.	XmCreateWorkingDialog.

5.4 Using Dialogs and Convenience Functions

Now that you have an idea of what Dialogs and convenience dialogs are available, you can learn how and when to use them in an application. This section explains each dialog and its associated convenience functions. Code segments are included to help you understand how to use them in your own applications. Don't forget that more detailed information on these widgets can be found in the respective man pages in the *HP OSF/Motif Programmer's Reference Manual.*

5.4.1 XmDialogShell

XmDialogShell is the Shell parent widget for all Dialogs. It provides the necessary communication with the window manager to allow the Dialogs to be managed and unmanaged. XmDialogShell is automatically created by the Dialog convenience functions and is used as the parent widget for XmBulletinBoard or any subclass of XmBulletinBoard. It can also be directly instantiated by using either of the two create functions that are available:

```
Widget XtCreatePopupShell (name, xmDialogShellWidgetClass, parent,
  arglist,argcount)

Widget XmCreateDialogShell (parent, name, arglist, argcount)
```

Both of these create functions create an instance of a DialogShell and return the associated widget ID. The following code segment shows you how to create `XmDialogShell` using `XtCreatePopupShell`:

```
Arg  args;
int  n;
Widget  dialog_shell;
Widget  parent_shell;

n = 0;
dialog_shell = XtCreatePopupShell ("dialog_shell",
  xmDialogShellWidgetClass, parent_shell, args, n);
```

The next code segment shows you how to create `XmDialogShell` using `XmCreateDialogShell`:

```
Arg  args;
int  n;
Widget  dialog_shell;
Widget  parent_shell;

n = 0;
dialog_shell = XmCreateDialogShell (parent_shell, "dialog_shell",
  args, n);
```

Remember that `XmDialogShell` is *automatically* created for you when you create any of the convenience Dialogs. You do not need to use either of the functions described above when you create a convenience Dialog. Also, note that XmDialogShell should be popped up by using XtManageChild on its Dialog child. If the child is created using XtCreateManagedWidget, it will try to popup the shell before it has been realized. This will result in an error.

5.4.2 XmBulletinBoard

XmBulletinBoard is a composite widget that can be instantiated alone. Its main purpose with Dialog widgets, however, is as a superclass widget that supplies resources to the subclass Dialog widgets. All of the other Dialog widgets except XmDialogShell are built in part from XmBulletinBoard. Refer to the XmBulletinBoard man page in the *HP OSF/Motif Programmer's Reference Manual* for a description of the XmBulletinBoard resources.

XmBulletinBoard can be directly instantiated by using either of the two create functions that are available:

```
Widget XtCreateWidget (name, xmBulletinBoardWidgetClass, parent,
   arglist,argcount)

Widget XmCreateBulletinBoard (parent, name, arglist, argcount)
```

Both of these create functions create an instance of a BulletinBoard and return the associated widget ID. The following code segment shows you how to create an instance of XmBulletinBoard using XmCreateWidget:

```
Arg  args;
int  n;
Widget  bboard;
Widget  parent_shell;

n = 0;
bboard = XtCreateWidget ("bboard", xmBulletinBoardWidgetClass,
   parent_shell, args, n);
XtManageChild (bboard);
```

The next code segment shows you how to create an instance of XmBulletinBoard using XmCreateBulletinBoard:

```
Arg  args;
int  n;
Widget  bboard;
Widget  parent_shell;
```

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

Remember that you do not need to use either of the create functions described above if you are creating convenience Dialogs.

5.4.3 XmCommand

XmCommand allows you to choose one selection from a list of selections. It is very much like the XmSelectionBox widget except that it has the capability to record selections in a "history region." The history region is accessible so that choices can be made from it as well as entering a choice from the keyboard. Figure 5-1 shows an example of a Command widget.

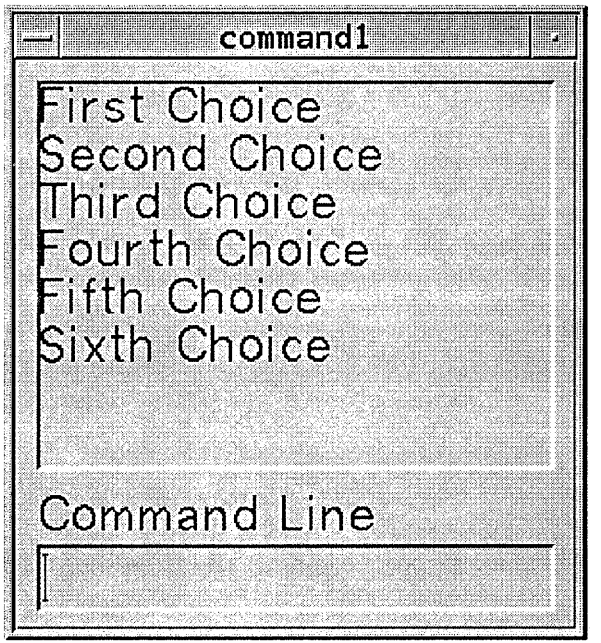


Figure 5-1. XmCommand Widget

The history region is displayed in the top box and the current selection is displayed in the command line box at the bottom. The history region will scroll automatically as the need arises. In figure 5-2 for example, more entries have been placed in the history region, causing the vertical ScrollBar to appear.

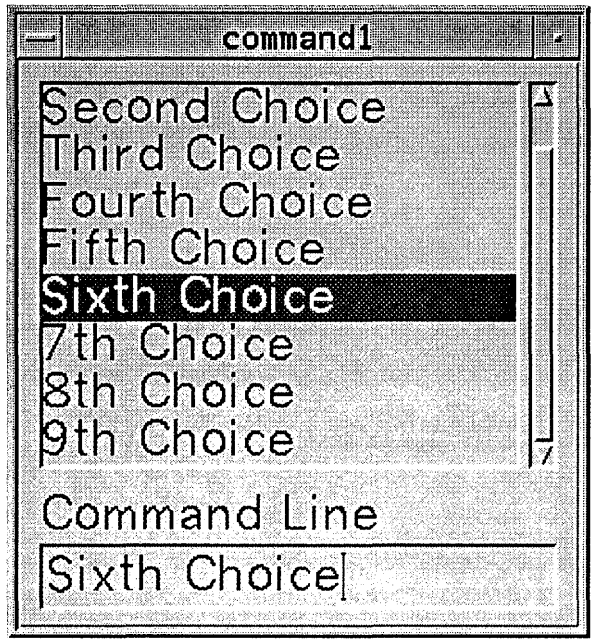


Figure 5-2. XmCommand Widget with Scrolled History Region

XmCommand can be directly instantiated by using either of the two create functions that are available:

```
Widget XtCreateWidget (name, xmCommandWidgetClass, parent,
  arglist,argcount)

Widget XmCreateCommand (parent, name, arglist, argcount)
```

Both of these create functions create an instance of a Command widget and return the associated widget ID. The following code segment shows you how to create an instance of XmCommand using XmCreateWidget:

```
Arg  args;
int  n;
Widget  command;
Widget  parent_shell;
```

```
n = 0;
command = XtCreateWidget ("command", xmCommandWidgetClass,
    parent_shell, args, n);
XtManageChild (command);
```

The next code segment shows you how to create an instance of XmCommand using XmCreateCommand:

```
Arg  args;
int  n;
Widget  command;
Widget  parent_shell;

n = 0;
command = XmCreateCommand (parent_shell, "command", args, n);
XtManageChild (command);
```

There are several other functions associated with XmCommand that perform certain operations to the command area string or the history region string. These functions are explained in the following list:

- **XmCommandAppendValue.**
  ```
  void XmCommandAppendValue (widget, command)

  Widget  widget;
  XmString  command;
  ```
  This function appends the passed, null-terminated command string to the end of the string that is currently displayed in the command line.

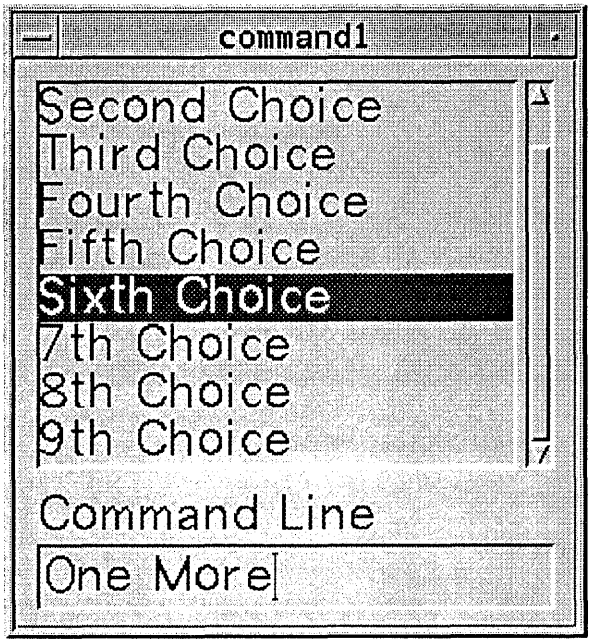


Figure 5-3. Results of XmCommand Append Value Operation

The code segment to accomplish this is shown below.

```
XmString  str;
Widget  w;

str = XmStringCreateLtoR ("addValue", XmSTRING_DEFAULT_CHARSET);
XmCommandAppendValue (w, str);
```

- **XmCommandSetValue.**
  `void XmCommandSetValue (widget, command)`

  ```
  Widget  widget;
  XmString  command;
  ```

  This function replaces the string that is currently displayed in the command line with the passed, null-terminated string.

- **XmCommandError.**
  `void XmCommandError (widget, error)`

```
Widget  widget;
XmString  error;
```

This function displays an error message in the history region.

- **XmCommandGetChild.**

```
void XmCommandGetChild (widget, child)

Widget  widget;
uns char  child;
```

This function returns the widget ID of the given child. The function takes these child types:

- XmDIALOG_PROMPT_LABEL.
- XmDIALOG_COMMAND_TEXT.
- XmDIALOG_HISTORY_LIST.

5.4.4 XmFileSelectionBox

XmFileSelectionBox is a widget very similar to XmSelectionBox. The difference is that XmFileSelectionBox is used to traverse through directories, viewing the names of the files and finally selecting a file on which to perform some action. One thing to remember about this widget: You *must* link in the library PW (-lPW) in order for FileSelectionBox to work properly. Figure 5-4 shows an example of a FileSelectionBox.

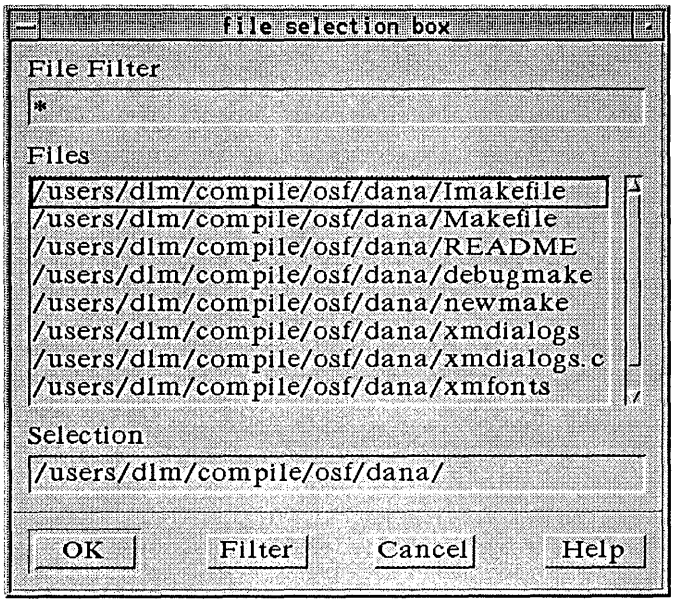


Figure 5-4. XmFileSelectionBox

XmFileSelectionBox can be directly instantiated by using either of the two create functions that are available:

```
Widget XtCreateWidget (name, xmFileSelectionBoxWidgetClass, parent,
   arglist,argcount)

Widget XmCreateFileSelectionBox (parent, name, arglist, argcount)
```

Both of these create functions create an instance of FileSelectionBox widget and return the associated widget ID. The following code segment shows you how to create an instance of XmFileSelectionBox using XmCreateWidget:

```
Arg  args;
int  n;
Widget  fselbox;
Widget  parent_shell;
```

```
n = 0;
fselbox = XtCreateWidget ("fselbox", xmFileSelectionBoxWidgetClass,
   parent_shell, args, n);
XtManageChild (fselbox);
```

The next code segment shows you how to create an instance of XmFileSelectionBox using XmCreateFileSelectionBox:

```
Arg  args;
int  n;
Widget  fselbox;
Widget  parent_shell;

n = 0;
fselbox = XmCreateFileSelectionBox (parent_shell, "fselbox", args, n);
XtManageChild (fselbox);
```

In the example shown in figure 5-4, the "File Filter" is an asterisk, indicating that all files in the directory should be listed. Here, all files in the directory /users/dlm/compile/osf/dana are listed. The Selection window at the bottom of the XmFileSelectionBox specifies the directory. Once a file has been selected, you can press the "OK" button and, depending on what the application has in the callback associated with this button, perform some action on the selected file. Note that you can select a file as many times as desired. The operation on that file is not performed until you select the "OK" PushButton.

You can change the File Filter by moving the pointer into the File Filter window and clicking mouse button 1. This "activates" the window and it is highlighted. You can then enter the desired filter by typing it in from the keyboard. When you have finished entering the new filter, move the pointer into the "Filter" button at the bottom of the main window and click mouse button 1. The files that meet the criteria established in the File filter window are then displayed in the "Files" window. Figure 5-5 shows the results of such an operation. Here we have changed the File Filter from "*" to "x*" and the files that meet that criteria are listed.

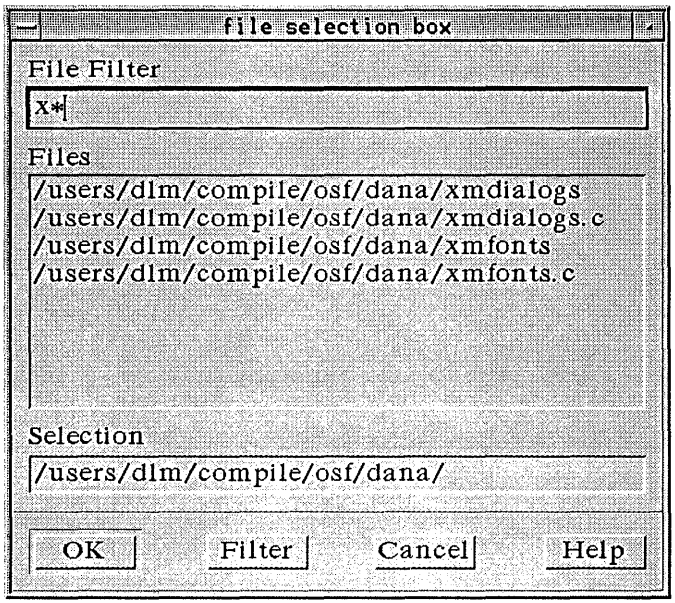


Figure 5-5. XmFileSelectionBox With New File Filter

5.4.5 XmForm

The Form widget is a container widget that has no input characteristics of its own. Constraints are placed on the Form widget's children. These constraints define attachments for each of the four sides of each child, and the attachments determine the layout of the Form widget when any resizing occurs. The child widgets' attachments can be to the Form widget, to another child of the Form widget, to a relative position within the Form widget, or to the initial position of the child.

XmForm can be directly instantiated by using either of the two create functions that are available:

```
Widget XtCreateWidget (name, xmFormWidgetClass, parent,
  arglist, argcount)

Widget XmCreateForm (parent, name, arglist, argcount)
```

Both of these create functions create an instance of the Form widget and return the associated widget ID. The following code segment shows you how to create an instance of XmForm using XmCreateWidget:

```
Arg  args;
int  n;
Widget  form;
Widget  parent_shell;

n = 0;
form = XtCreateWidget ("form", xmFormWidgetClass, parent_shell,
  args, n);
XtManageChild (form);
```

The next code segment shows you how to create an instance of XmForm using XmCreateForm:

```
Arg  args;
int  n;
Widget  form;
Widget  parent_shell;

n = 0;
form = XmCreateForm (parent_shell, "form", args, n);
XtManageChild (form);
```

You can create a Form Dialog by using the convenience function XmFormCreateDialog. This function creates and returns a Form widget as a child of a DialogShell widget.

```
Arg  args;
int  n;
Widget  form;
Widget  parent_shell;

n = 0;
form = XmCreateFormDialog (parent_shell, "form", args, n);
XtManageChild (form);
```

Figure 5-6 shows an example of a Form widget.

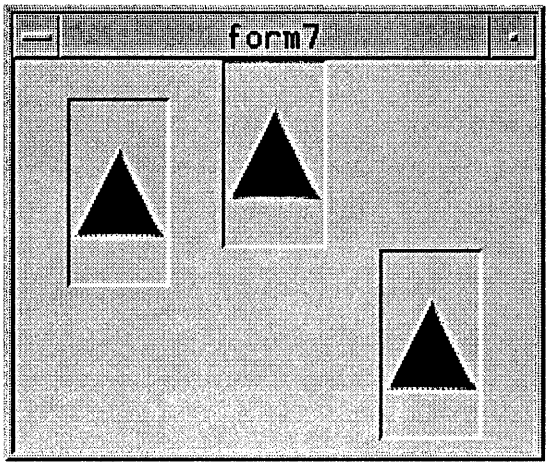


Figure 5-6. Form Widget with ArrowButtons

There are three ArrowButton widgets positioned within the Form widget. The ArrowButton on the left is set so that its left side position is offset an amount equal to ten percent of the Form widget's width, its right side position is offset 30 percent of the Form widget's width, and the top is set to a *fixed* offset of 20 pixels from the top of the Form widget. When the Form widget is resized, the spatial relationships between the Form widget and its children remain the same. This fixed relationship is shown in figure 5-7.

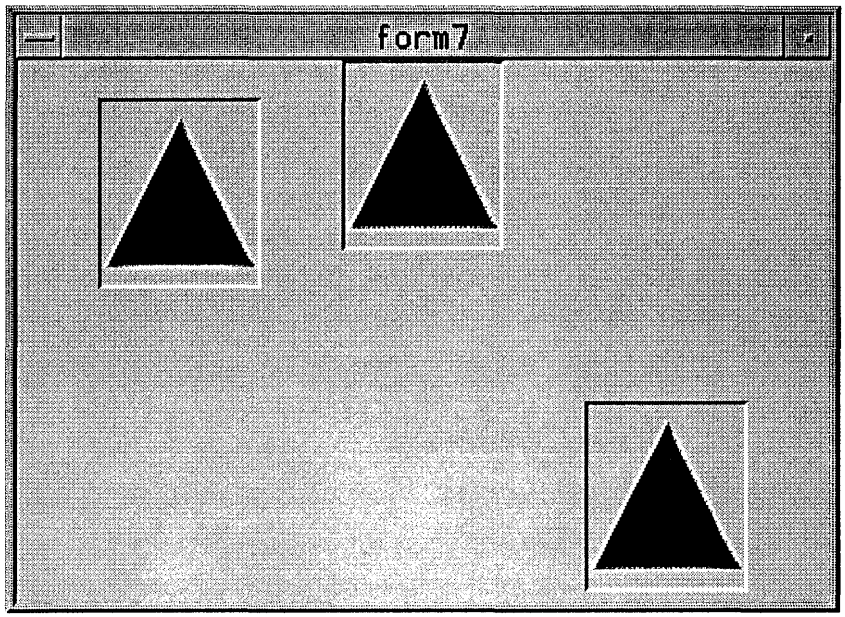


Figure 5-7. Form Widget after Resizing

The code segment that positions the ArrowButton on the left is shown below.

```
Widget  arrow1;
int  n;

n = 0;
XtSetArg (args[n], XmNleftAttachment, XmATTACH_POSITION); n++;
XtSetArg (args[n], XmNleftPosition, 10); n++;
XtSetArg (args[n], XmNrightAttachment, XmATTACH_POSITION); n++;
XtSetArg (args[n], XmNrightPosition, 30); n++;
XtSetArg (args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg (args[n], XmNtopOffset, 20); n++;
arrow1 = XtCreateManagedWidget("arrow1", xmArrowButtonWidgetClass,
   form, args, n);
```

Note that the position of the top side of the ArrowButton is a constant value (20 pixels in this case), regardless of any resizing operations that may occur. This is because the XmNtopAttachment resource is set to XmATTACH_FORM as opposed to XmATTACH_POSITION.

5.4.6 XmMessageBox

A MessageBox is used just as its name implies, to pass messages to the user of an application. MessageBox is a subclass of BulletinBoard and inherits a large number of the BulletinBoard resources. Convenience Dialogs based on MessageBox are provided for several user-interaction functions: providing information, asking questions, and notifying the user if errors occur. Figure 5-8 shows some examples of the MessageBox.

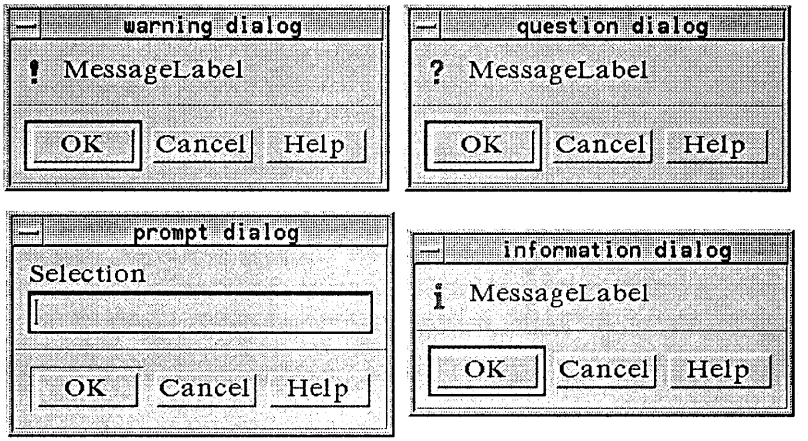


Figure 5-8. MessageBox Examples

XmMessageBox can be directly instantiated by using either of the two create functions that are available:

```
Widget XtCreateWidget (name, xmMessageBoxWidgetClass, parent,
   arglist, argcount)

Widget XmCreateMessageBox (parent, name, arglist, argcount)
```

Both of these create functions create an instance of the MessageBox widget and return the associated widget ID. The following code segment shows you how to create an instance of XmMessageBox using XmCreateWidget:

```
Arg  args;
int  n;
Widget  msgbox;
Widget  parent_shell;
```

```
n = 0;
msgbox = XtCreateWidget ("msgbox", xmMessageBoxWidgetClass,
    parent_shell, args, n);
XtManageChild (msgbox);
```

The next code segment shows you how to create an instance of XmMessageBox using XmCreateMessageBox:

```
Arg  args;
int  n;
Widget  msgbox;
Widget  parent_shell;

n = 0;
msgbox = XmCreateMessageBox (parent_shell, "msgbox", args, n);
XtManageChild (msgbox);
```

You can create a MessageBox Dialog by using the convenience function XmMessageCreateDialog. This function creates and returns a MessageBox widget as a child of a DialogShell widget.

```
Arg  args;
int  n;
Widget  msgbox;
Widget  parent_shell;

n = 0;
msgbox = XmCreateMessageDialog (parent_shell, "msgbox", args, n);
XtManageChild (msgbox);
```

A MessageBox can contain a message symbol, a message, and up to three standard PushButtons (the default buttons are "OK," "Cancel," and "Help"). The symbol (if any) appears in the upper left part of the MessageBox (see figure 5-8), the message appears in the top and center-to right side, and the buttons appear along the bottom edge.

The defaults for the button labels and the message symbols can be changed. The button labels can be changed by setting a resource in the program or in a defaults file. For example, if you wanted to change the "OK" label to "Close," you could use this code segment:

```
n = 0;
XtSetArg (args[n], XmNokLabelString, XmStringCreateLtoR ("Close",
   XmSTRING_DEFAULT_CHARSET)); n++;
messageD = XmCreateMessageDialog (parent, "fontbox", args, n);
```

XmStringCreateLtoR is a compound string function. See chapter 8, "Additional Functionality," for more information on this function and compound strings in general.

You can make the same change by using the following statement in a defaults file:

```
*msgbox.okLabelString: Close
```

Note that the above statement applies only to the MessageBox widget whose name is "msgbox" as specified in the XmCreateMessageDialog or XmCreateMessageBox.

The message label can be changed in the same manner as the button. Use the resource XmNmessageString instead of XmNokLabelString.

There are several other convenience dialogs that allow you to create special versions of MessageBox. These convenience dialogs are XmCreateErrorDialog, XmCreateInformationDialog, XmCreateQuestionDialog, XmCreateWarningDialog, and XmCreateWorkingDialog. You use these functions to create the appropriate dialogs in exactly the same manner as described earlier for the XmMessageBoxDialog.

5.4.7 XmSelectionBox

SelectionBox is a general dialog widget that allows you to select an item from a list of items. SelectionBox can contain a label, a list of items from which to choose, a selection text edit window that allows you to enter a selection directly, and three PushButtons ("OK," "Cancel," and "Help"). An example of a SelectionBox widget is shown in figure 5-9.

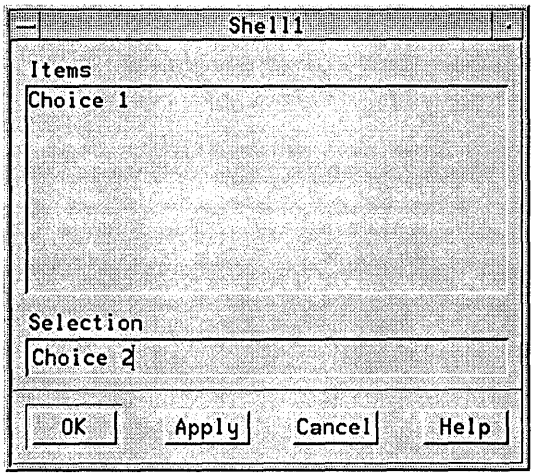


Figure 5-9. SelectionBox Widget

The user can select an item from the SelectionBox in either of two ways:

- By scrolling through the list of items and selecting one. The item you select will be displayed in the SelectionBox text edit window.
- By entering the item name directly into the text edit window.

You may select a new item as many times as you desire since no action is taken until you move the pointer to the "OK" button and click mouse button 1.

XmSelectionBox can be directly instantiated by using either of the two create functions that are available:

```
Widget XtCreateWidget (name, xmSelectionBoxWidgetClass, parent,
   arglist,argcount)

Widget XmCreateSelectionBox (parent, name, arglist, argcount)
```

Both of these create functions create an instance of the SelectionBox widget and return the associated widget ID. The following code segment shows you how to create an instance of XmSelectionBox using XmCreateWidget:

```
Arg  args;
int  n;
Widget  selectbox;
Widget  parent_shell;

n = 0;
selectbox = XtCreateWidget ("selectbox", xmSelectionBoxWidgetClass,
   parent_shell, args, n);
XtManageChild (selectbox);
```

The next code segment shows you how to create an instance of XmSelectionBox using XmCreateSelectionBox:

```
Arg  args;
int  n;
Widget  selectbox;
Widget  parent_shell;

n = 0;
selectbox = XmCreateSelectionBox (parent_shell, "selectbox",
  args, n);
XtManageChild (selectbox);
```

You can create a SelectionBox Dialog by using the convenience function XmSelectionBoxCreateDialog. This function creates and returns a SelectionBox widget as a child of a DialogShell widget.

```
Arg  args;
int  n;
Widget  selectbox;
Widget  parent;

n = 0;
selectbox = XmCreateSelectionBoxDialog (parent, "selectbox", args,
XtManageChild (selectbox);
```

The defaults for the button labels and the list and text window labels can be changed. The button labels can be changed by setting a resource in the program or in a defaults file. For example, if you wanted to change the "OK" label to "Close," you could use this code segment:

```
n = 0;
XtSetArg (args[n], XmNokLabelString, XmStringCreateLtoR ("Close",
   XmSTRING_DEFAULT_CHARSET)); n++;
selectbox = XmCreateSelectionBoxDialog (parent, "selectbox", args,
```

The following code segment shows how to set the list items that appear in the list window.

```
XmString  item[5];
   .
   .
item[0] = XmStringCreateLtoR ("one", XmSTRING_DEFAULT_CHARSET);
item[1] = XmStringCreateLtoR ("two", XmSTRING_DEFAULT_CHARSET);
item[2] = XmStringCreateLtoR ("three", XmSTRING_DEFAULT_CHARSET);
item[3] = NULL
   .
   .

n = 0;
XtSetArg (args[n], XmNlistItems, item); n++;
XtSetArg (args[n], XmNlistItemCount, 3); n++;
```

An array of type `XmString` is defined to hold the list items. Since only three of the five entries in the array are used, the fourth is set to NULL to identify the end of the list. Each entry is set manually, and then the argument list is set in the normal manner.

Menus 6

The menu system provides three types of menus – Popup, Pulldown, and Option menus. This chapter describes how to implement and use these menus. The chapter includes:

- An overview of the menu system.
- Guidelines for creating and interacting with each menu type.
- A description of the mouse and keyboard interfaces for each menu type.

6.1 Overview of the Menu System

6.1.1 An Introduction

There are three types of menu systems:

- Popup menu systems.
- Pulldown menu systems.
- Option menu systems.

The term "menu system" refers to a combination of various widgets that create the visual and interactive behavior of a menu. For example, a Pulldown menu system characteristically consists of a MenuBar containing a number of CascadeButtons; the CascadeButtons are used to post various Pulldown MenuPanes which, in turn, contain various buttons.

The major widget components of menu systems are RowColumn widgets that are configured to behave as:

- Popup MenuPanes.
- Pulldown MenuPanes.
- MenuBars.
- Option menus.

For example, a Popup MenuPane is a RowColumn widget created to behave as a Popup MenuPane; likewise, a MenuBar is a type of RowColumn widget.

The menu system provides the ability to manually create the major menu widgets by creating RowColumn widgets of the appropriate types. For example, a Popup MenuPane can be created by creating a MenuShell widget and a child RowColumn widget of type XmMENU_POPUP. However, a set of convenience functions is provided that automatically create RowColumn widgets of the appropriate type and, when necessary, a parent MenuShell. (Popup and Pulldown MenuPanes require MenuShell parents; MenuBars and Option menus do not have MenuShell parents.) For example, the XmCreatePopupMenu convenience function creates a RowColumn widget configured to act as a Popup MenuPane and automatically creates its parent MenuShell.

Most of the instructions and examples in this chapter use the convenience functions. Creating menus by separately creating MenuShells and RowColumn widgets is discussed at the end of the chapter.

6.1.2 Convenience Functions and Widgets Used to Create Menus

A menu is constructed from combinations of widgets created explicitly or by using convenience functions.

The following convenience functions create RowColumn widgets that act as MenuPanes. MenuPanes are "transient" features in an application – they are not displayed until they are posted by a particular event, and they are unposted at the conclusion of some other event.

- **XmCreatePopupMenu** convenience function. This function automatically creates a Popup MenuPane and its required parent MenuShell.
- **XmCreatePulldownMenu** convenience function. This function automatically creates a Pulldown MenuPane and its required parent MenuShell.

The following convenience functions create RowColumn widgets configured to act as other components of menu systems. These are nontransient features of an application.

- **XmCreateMenuBar** convenience function. This function automatically creates a MenuBar. Menubars are typically used as the basis for building Pulldown menu systems. A MenuBar is the top-level component of a Pulldown menu system.
- **XmCreateOptionMenu** convenience function. This function automatically creates an Option menu.

In addition to the RowColumn widgets created by convenience functions, the following widgets and gadgets are used in menu systems:

- **XmCascadeButton** and **XmCascadeButtonGadget**. CascadeButtons are used as the visual means to display Pulldown menus, Option menus, and submenus.

- **XmSeparator** and **XmSeparatorGadget.** The Separator widget is used to separate unrelated buttons or groups of buttons within a MenuPane.
- **XmLabel** and **XmLabelGadget.** The Label widget is used to provide a title for a MenuPane.
- **XmPushButton** and **XmPushButtonGadget.** PushButtons provide the means for selecting an item from a menu.
- **XmToggleButton** and **XmToggleButtonGadget.** ToggleButtons provide a way to set nontransitory states using menus.
- **XmRowColumn.** The RowColumn widget is a general-purpose RowColumn manager. Popup MenuPanes, Pulldown MenuPanes, MenuBars, and Option menus are *types* of RowColumn widgets. When menu RowColumn widgets are created without using convenience functions, the resource `XmNrowColumnType` specifies the type of MenuPane created.
- **XmMenuShell.** The MenuShell widget is a shell widget designed to contain a Popup or Pulldown MenuPane as its child. The convenience functions that create Popup and Pulldown menus automatically create their parent MenuShell.

Note that the menu system does not implement Popup MenuPanes, Pulldown MenuPanes, Option menus, and MenuBars as separate widget classes. For example, no PopupMenu widget class exists; rather, a convenience function exists for creating the appropriately configured RowColumn widget.

6.1.3 Introducing the Three Menu Types

Popup Menu System

A Popup menu system consists of a single Popup MenuPane containing a combination of Label, PushButton, ToggleButton, and Separator widgets or gadgets. In addition, the MenuPane can contain CascadeButtons or CascadeButtonGadgets that are used to access Pulldown MenuPanes that function as submenus of the Popup MenuPane.

Figure 6-1 shows the top level of a Popup menu system.

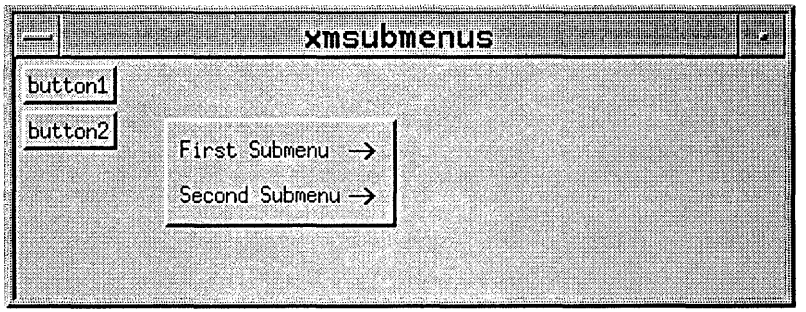


Figure 6-1. Top Level of a Popup Menu System

The CascadeButtons have arrows that indicate the presence of submenus. Moving the pointer to the "First Submenu" button displays its submenu.

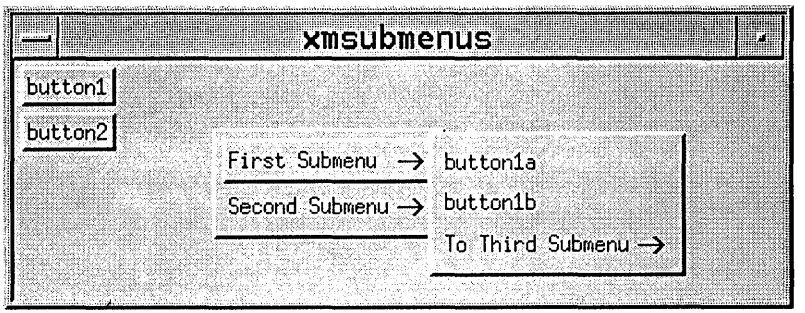


Figure 6-2. Submenu of a Popup Menu System

The submenu contains two PushButtons and one CascadeButton. Moving the pointer to the "Third Submenu" button displays its submenu.

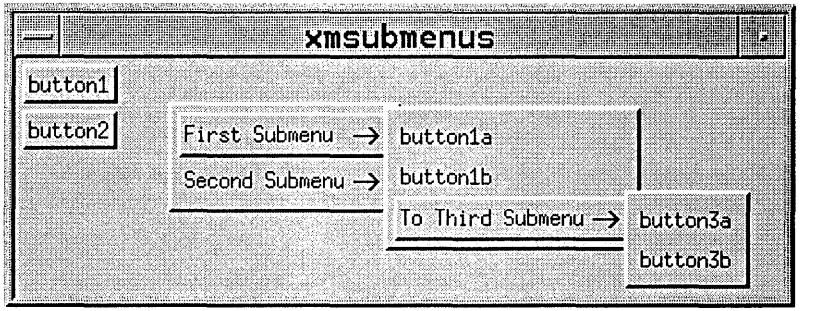


Figure 6-3. Popup Menu System With Two Cascading Submenus

Pulldown Menu System

A Pulldown menu system typically consists of a MenuBar, a set of CascadeButtons parented from the MenuBar, and a Pulldown MenuPane attached to each CascadeButton. The CascadeButtons are displayed within the MenuBar and provide the means for displaying the MenuPanes. In addition, the menu system may include Label, PushButton, ToggleButton, and Separator widgets or gadgets.

Figure 6-4 shows a MenuBar, which is the top level of a Pulldown menu system.

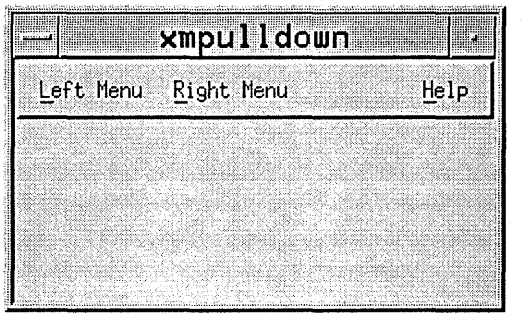


Figure 6-4. Menu Bar of a Pulldown Menu System

Moving the pointer to "Right Menu" and pressing mouse button 1 displays its Pulldown MenuPane.

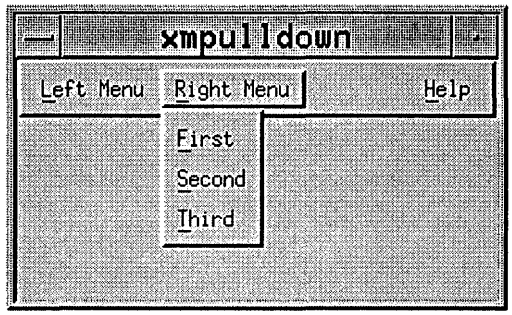


Figure 6-5. Displaying a Pulldown MenuPane

Option Menu System

Visually, an Option menu is composed of three areas:

- A descriptive LabelGadget. Typically, the label describes the types of options available.
- A Pulldown MenuPane containing PushButtons or PushButtonGadgets. The buttons represent the available options.
- A selection area consisting of a CascadeButtonGadget that contains a label string. The label string reflects the most recent option chosen from the Pulldown MenuPane.

The top level of an Option menu system shows the descriptive label and selection area.

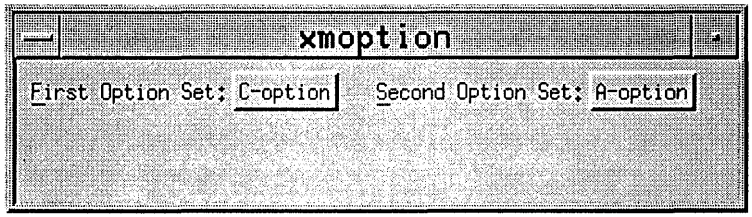


Figure 6-6. Top Level of an Option Menu System

Pressing mouse button 1 while the pointer is in the selection area displays the Pulldown MenuPane containing the options.

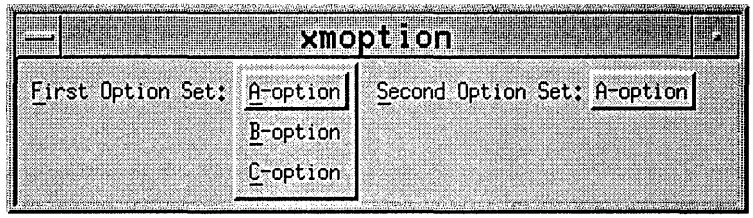


Figure 6-7. The Pulldown MenuPane in an Option Menu System

6.2 Creating Popup Menu Systems

The top level of a Popup menu system is a Popup MenuPane. Popup MenuPanes are implemented as XmRowColumn widgets configured to operate as a Popup MenuPanes. The Popup MenuPane may contain CascadeButtons or CascadeButtonGadgets, which are used to access Pulldown MenuPanes that act as submenus of the Popup MenuPane. (Submenus are discussed later in this chapter.)

A Popup MenuPane displays a three-dimensional shadow around the edge of the MenuPane unless the feature has been disabled by the application.

The Popup MenuPane must be the child of a MenuShell widget. If the Popup MenuPane is created using the convenience function, a MenuShell is automatically created as the real parent of the MenuPane. If the Popup MenuPane is created without using the convenience function, the MenuShell widget must be created first.

6.2.1 Popup MenuPane Convenience Function

A Popup MenuPane is created using the convenience function:

`Widget XmCreatePopupMenu(`*parent*`,` *name*`,` *arglist*`,` *argcount*`)`

`XmCreatePopupMenu` creates a Popup MenuPane and a parent MenuShell, and returns the widget ID for the MenuPane. The Popup MenuPane is created as a RowColumn widget with the `XmNrowColumnType` resource set to `XmMENU_POPUP`. This resource cannot be changed by the application.

6.2.2 Event Handlers for Popup Menu Systems

Popup menu systems require an event handler procedure that is called when a specified event (usually, a ButtonPress) occurs in the widget(s) to which the Popup menu system is attached. Usually, this is the `parent` specified by the `XmCreatePopupMenu` function and the parent's descendents. The event handler procedure should test that the proper mouse button has been pressed and then displays the Popup MenuPane.

The `XtAddEventHandler` function registers the event handler procedure with the dispatch mechanism. It has the syntax:

```
void XtAddEventHandler(w, event_mask, nonmaskable, proc, client_data)
        Widget w;
        EventMask event_mask;
        Boolean nonmaskable;
        XtEventHandler proc;
        caddr_t client_data;
```

w	Specifies the widget to add the callback to.
event_mask	Specifies the event mask for which to call this procedure.
nonmaskable	Specifies whether this procedure should be called on the nonmaskable events.
proc	Specifies the client event handler procedure.
client_data	Specifies additional data to be passed to the client's event handler.

For example, the line

```
XtAddEventHandler(rc, ButtonPressMask, False, PostIt, popup);
```

registers the procedure `PostIt` for the event `ButtonPress` within the widget `rc` and all of rc's descendents.

6.2.3 Procedure for Creating a Popup Menu.

The following steps create a Popup menu. Following each step is a code segment that accomplishes the task.

1. Use the `XmCreatePopupMenu` convenience function to automatically create the Popup MenuPane and its required parent MenuShell. Register the event handler. The following lines creates a Popup MenuPane as a child of widget `form1`.

```
popup = XmCreatePopupMenu(form1, "popup", NULL, 0);
XtAddEventHandler(form1, ButtonPressMask, False, PostIt,
      popup);
```

2. Create the contents of the MenuPane. The following segment creates a title (LabelGadget), SeparatorGadget, and three PushButtonGadgets.

```
XtSetArg(args[0], XmNlabelString, XmStringCreate("Menu
      Title", XmSTRING_DEFAULT_CHARSET));
item[0] = XmCreateLabelGadget(popup, "title", args, 1);
item[1] = XmCreateSeparatorGadget(popup, "separator",
      NULL, 0);
item[2] = XmCreatePushButtonGadget(popup, "button1",
      NULL, 0);
item[3] = XmCreatePushButtonGadget(popup, "button2",
      NULL, 0);
item[4] = XmCreatePushButtonGadget(popup, "button3",
      NULL, 0);
XtManageChildren(item, 5);
```

The following illustration shows the parenting relationships to use when creating a Popup menu system using convenience functions.

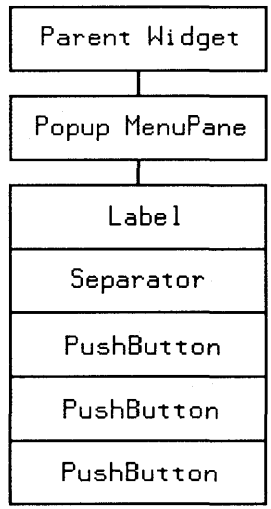


Figure 6-8. Creating a Popup Menu System With Convenience Functions

Mouse button 3 is the default primary means of interacting with a Popup menu.

A Popup MenuPane is not visible until it is displayed by the user. The Popup menu system is normally associated with a particular widget and all of that widget's descendents. The Popup MenuPane is posted (displayed) by moving the pointer into the associated widget or one of its descendents and then pressing mouse button 3. If the Popup menu system includes any Pulldown MenuPane submenus, they are not displayed until the pointer is moved into the associated CascadeButton widget or gadget.

The application is responsible for posting the Popup MenuPane by using `XtManageChild` to display the MenuPane. This is usually done in the event handler added to the Popup MenuPane's parent. All visible MenuPanes (the Popup MenuPane and any displayed submenus) are automatically unposted when the user has completed interacting with the menu.

Once a MenuPane has been posted, menu items are armed when the pointer enters them and disarmed when it leaves. If the pointer is moved into a CascadeButton or CascadeButtonGadget, the associated submenu is posted. Releasing mouse button 3 while a menu item is armed activates the menu item. If the pointer is not within a menu item when mouse button 3 is released, then all visible MenuPanes are unposted.

Ordinarily, a Popup menu is positioned to the right and beneath the pointer. However, if this placement causes a portion of the MenuPane to be inaccessible, the menu may be automatically repositioned to force the Popup MenuPane on the screen.

The mouse button used to interact with Popup menus can be changed using the RowColumn resource `XmNwhichButton`.

The Keyboard Interface

Keyboard traversal is activated and deactivated by the user. When a user is interacting with the menu using the mouse, traversal is enabled by releasing the mouse button while the pointer is within any CascadeButton or CascadeButtonGadget; releasing the mouse button posts the associated submenu and enables traversal for all the MenuPanes in the Popup menu system. When traversal is enabled:

- The directional keys traverse the menu hierarchy.
- The Return key selects the currently-armed menu item.
- The ESC key unposts all the MenuPanes in the Popup menu system.
- Pressing a mnemonic for a menu item in the most recently posted MenuPane selects that item.
- Pressing an accelerator for a menu item selects that item.
- Pressing mouse button 3 disables traversal and reenables interactive operation.

An accelerator can be associated with a Popup menu. The default accelerator is Function Key 4 (f4). When f4 is pressed while the pointer is located within the associated widget

or one of its children, the first MenuPane in the Popup menu hierarchy is posted and traversal is enabled. The user interacts with the menu as described previously for keyboard traversal.

Use the resource `XmNmenuAccelerator` to change the accelerator.

6.2.4 Sample Program

This sample program creates a Popup menu system. The menu can be posted using the mouse or by using the accelerator CTRL P. Items can be selected using the mouse or by using the underlined mnemonics.

The source code for this program is located on your system in `/usr/contrib/Xm/xmpopup.c`.

```
/* Popup Menu Example */

#include <Xm/Xm.h>
#include <Xm/LabelG.h>
#include <Xm/PushBG.h>
#include <Xm/CascadeBG.h>
#include <Xm/SeparatoG.h>
#include <Xm/RowColumn.h>


/*********** Callback for the Pushbuttons *********************/

void ButtonCB (w, client_data, call_data)
Widget      w;    /*  widget id*/
caddr_t     client_data;/*  data from application   */
caddr_t     call_data;/*  data from widget class  */
{
   /*  print message and terminate program  */
   printf ("Button %s selected.\n", client_data);

}

/************** Event Handler for Popup Menu *****************/

PostIt (w, popup, event)
Widget w;
Widget popup;
XButtonEvent * event;
{
```

```
    if (event->button != Button3)
        return;
    XmMenuPosition(popup, event);
    XtManageChild(popup);
}

/************************Main Logic for Program **************/


void main (argc, argv)
int argc;
char **argv;
{
    Widget toplevel, popup, rc;
    Widget buttons[2], popupBtn[5];
    Arg args[5];

/* Initialize toolkit */

    toplevel = XtInitialize (argv[0], "PopupMenu", NULL, 0,
        &argc, argv);

/* Create RowColumn in toplevel with two PushButtonGadgets */

    XtSetArg(args[0], XmNwidth, 150);
    XtSetArg(args[1], XmNheight, 125);
    XtSetArg(args[2], XmNresizeWidth, False);
    XtSetArg(args[3], XmNresizeHeight, False);
    XtSetArg(args[4], XmNadjustLast, False);
    rc = XmCreateRowColumn(toplevel, "rc", args, 5);
    XtManageChild(rc);

    buttons[0] = XmCreatePushButtonGadget (rc, "buttonA",
        NULL, 0);
    XtAddCallback(buttons[0], XmNactivateCallback, ButtonCB,
        "A");

    buttons[1] = XmCreatePushButtonGadget (rc, "buttonB",
        NULL, 0);
    XtAddCallback(buttons[1], XmNactivateCallback, ButtonCB,
        "B");
    XtManageChildren (buttons, 2);
```

```
/* Create popup menu with accelerator CTRL P */

   XtSetArg(args[0], XmNmenuAccelerator, "Ctrl <Key> p");
   popup = XmCreatePopupMenu(rc, "popup", args, 1);
   XtAddEventHandler(rc, ButtonPressMask, False, PostIt, popup);


/* Create title for the popup menu and a separator */

   XtSetArg(args[0], XmNlabelString,
      XmStringCreate("Menu Title", XmSTRING_DEFAULT_CHARSET));
   popupBtn[0] = XmCreateLabelGadget(popup, "Title", args, 1);

   popupBtn[1] = XmCreateSeparatorGadget(popup, "separator",
      NULL, 0);

/* Create three PushButtonGadgets in the popup menu */

   XtSetArg(args[0], XmNmnemonic, '1');
   popupBtn[2] = XmCreatePushButtonGadget(popup, "button1",
      args, 1);
   XtAddCallback(popupBtn[2], XmNactivateCallback, ButtonCB,
      "1");

   XtSetArg(args[0], XmNmnemonic, '2');
   popupBtn[3] = XmCreatePushButtonGadget(popup, "button2",
      args, 1);
   XtAddCallback(popupBtn[3], XmNactivateCallback, ButtonCB,
      "2");

   XtSetArg(args[0], XmNmnemonic, '3');
   popupBtn[4] = XmCreatePushButtonGadget (popup, "button3",
      args, 1);
   XtAddCallback (popupBtn[4], XmNactivateCallback, ButtonCB,
      "3");
   XtManageChildren (popupBtn, 5);

/* Get and dispatch events */

   XtRealizeWidget(toplevel);

   XtMainLoop();
}
```

6.3 Creating a Pulldown Menu System

The basis of a Pulldown menu system is a MenuBar containing a set of CascadeButtons. (CascadeButtonGadgets are not allowed as the children of a MenuBar.) The CascadeButtons are used to display Pulldown MenuPanes. One of the CascadeButtons (typically, the one that is used to display help information) may be treated specially. This button is always positioned at the lower right corner of the MenuBar.

Two convenience functions, `XmCreateMenuBar` and `XmCreatePulldownMenu`, create the appropriate RowColumn widgets.

In addition to their use in Pulldown menu systems, Pulldown MenuPanes are used to create submenus in both Popup and Pulldown menu systems. Submenus are discussed in the next section.

6.3.1 MenuBar Create Function

A MenuBar is created using the convenience function:

`Widget XmCreateMenuBar(`*parent*, *name*, *arglist*, *argcount*`)`

`XmCreateMenuBar` creates a MenuBar as a RowColumn widget with the `XmNrowColumnType` resource set to `XmMENU_BAR`. This resource cannot be changed by the application. No MenuShell is created for MenuBar.

The MenuBar displays a three-dimensional shadow around its edge unless this feature has been disabled by the application.

6.3.2 Pulldown MenuPane Create Function

A Pulldown MenuPane is created using the convenience function:

`Widget XmCreatePulldownMenu(`*parent*, *name*, *arglist*, *argcount*`)`

To create a Pulldown MenuPane that is displayed using a CascadeButton in a MenuBar, specify the MenuBar as the *parent* in the `XmCreatePulldownMenu` function.

`XmCreatePulldownMenu` creates a Pulldown MenuPane and a parent MenuShell, and returns the widget ID for the MenuPane. The Pulldown MenuPane is created as a RowColumn widget with the `XmNrowColumnType` resource set to `XmMENU_PULLDOWN`. This resource cannot be changed by the application.

6.3.3 Creating a Help Button

The MenuBar resource XmNmenuHelpWidget specifies a CascadeButton that will be positioned at the lower right corner of the MenuBar. Typically, this CascadeButton is used to display help information. The Pulldown menu sample program creates a help button.

6.3.4 Procedure for Creating a Pulldown Menu

The following steps create a Pulldown menu. Following each step is a code segment that accomplishes the task.

1. Use the XmCreateMenuBar convenience function to create the MenuBar. The following lines create the MenuBar as the child of widget form1.

```
menubar = XmCreateMenuBar(form1, "menubar", NULL, 0);
XtManageChild(menubar);
```

2. Create one or more Pulldown MenuPanes as submenus (children) of the MenuBar.

```
pulldown1 = XmCreatePulldownMenu(menubar, "pulldown1",
      NULL, 0);
pulldown2 = XmCreatePulldownMenu(menubar, "pulldown2",
      NULL, 0);
```

3. Create a CascadeButton widget for each Pulldown MenuPane. The CascadeButtons and MenuPanes must have the same parent (in this case, menubar). Use the resource XmNsubMenuId to attach each CascadeButton to its MenuPane.

```
XtSetArg(args[0], XmNsubMenuId, pulldown1);
cascade[0] = XmCreateCascadeButton(menubar, "cascade1",
      args, 1);

XtSetArg(args[0], XmNsubMenuId, pulldown2);
cascade[1] = XmCreateCascadeButton(menubar, "cascade2",
      args, 1);
XtManageChildren(cascade, 2);
```

4. Create one or more buttons in each Pulldown MenuPane. The following lines create two PushButtonGadgets in each MenuPane.

```
pbutton1[0] = XmCreatePushButtonGadget(pulldown1,
      "button1a", NULL, 0);
pbutton1[1] = XmCreatePushButtonGadget(pulldown1,
      "button1b", NULL, 0);
XtManageChildren(pbutton1, 2);

pbutton2[0] = XmCreatePushButtonGadget(pulldown2,
      "button2a", NULL, 0);
```

```
pbutton2[1] = XmCreatePushButtonGadget(pulldown2,
      "button2b", NULL, 0);
XtManageChildren(pbutton2, 2);
```

The following illustration shows the parenting relationships and attachments (dashed lines) to use when creating a Pulldown menu system using convenience functions. The system includes a CascadeButton MenuPane designated as a help menu.

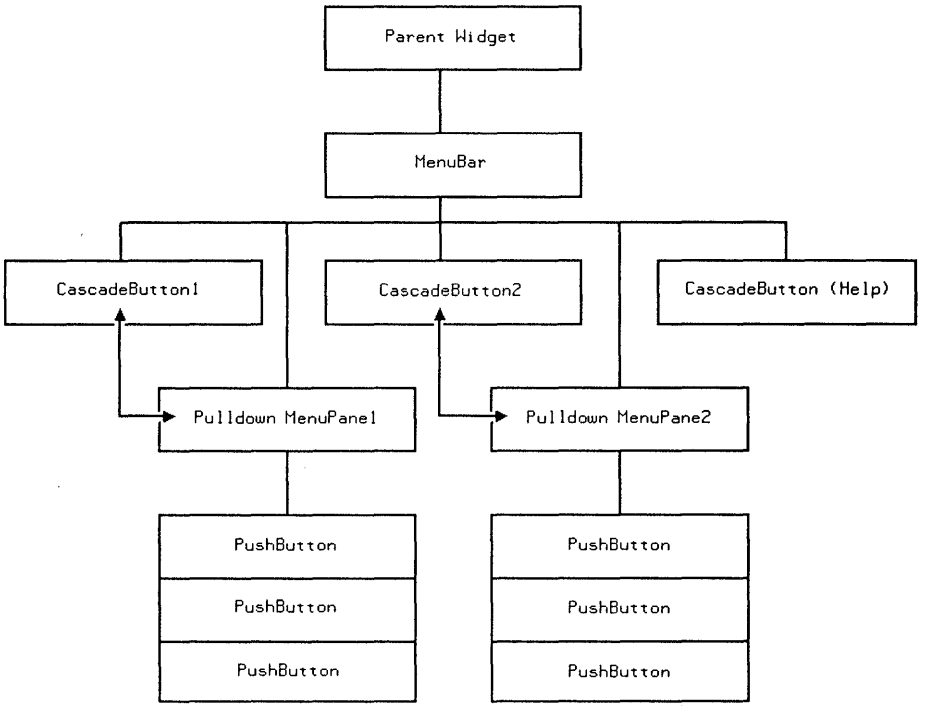


Figure 6-9. Creating a Pulldown Menu System With Convenience Functions

6.3.5 Interacting With Pulldown Menus

Mouse Input

Mouse button 1 is the default primary means of interacting with a Pulldown menu.

Pressing mouse button 1 while the pointer is positioned in a CascadeButton in the MenuBar "arms" and highlights the CascadeButton. If the CascadeButton has an associated Pulldown MenuPane, the MenuPane is posted. At this point, the pointer can be:

- Moved down into the Pulldown MenuPane. Menu items are "armed" when the pointer enters them and "disarmed" when it leaves. If the pointer is moved into a CascadeButton widget or gadget, the associated submenu is posted. Releasing mouse button 1 while an item is "armed" activates the menu item.
- Moved to a different CascadeButton within the MenuBar. This unposts the current Pulldown MenuPane and posts the MenuPane attached to the other CascadeButton.

Releasing mouse button 1 while the pointer is outside the menu hierarchy unposts all visible submenus and disarms the MenuBar.

The mouse button used to interact with Pulldown menus can be changed using the RowColumn resource `XmNwhichButton` for the MenuBar.

The Keyboard Interface

Keyboard traversal is activated and deactivated by the user. When a user is interacting with the menu using the mouse, traversal is enabled by either of the following:

- Releasing the mouse button while the pointer is within any CascadeButton widget or gadget. Releasing the mouse button posts the associated submenu and enables traversal for all the MenuPanes in the Pulldown menu system.
- Pressing Functon Key 10 (F10). This highlights the first CascadeButton in the MenuBar and enables traversal. The left and right arrow keys traverse to other CascadeButtons in the MenuBar. Pressing the down arrow key, the up arrow key, or the Return key posts the Pulldown MenuPane associated with the highlighted CascadeButton.

When traversal is enabled,

- The directional keys traverse the menu hierarchy.
- The Return key selects the currently armed menu item.
- The ESC key unposts all the submenus and disarms the CascadeButton in the MenuBar.
- Pressing a mnemonic for a menu item in the most recently posted MenuPane selects that item.
- Pressing an accelerator for a menu item selects that item.
- Pressing mouse button 1 disables traversal and reenables interactive operation.

Mnemonics can be used to post the Pulldown MenuPanes. The mnemonics are resources of the CascadeButtons in the MenuBar. To use a mnemonic associated with a MenuBar CascadeButton, preface it with the "Meta" modifier key.

Traversal is enabled when a menu is posted using a mnemonic. The user interacts with the menu as described previously for keyboard traversal.

6.3.6 Sample Program

The following sample program creates a Pulldown menu system consisting of a MenuBar containing three CascadeButtons. One CascadeButton is designated as a help button; the other two are attached to Pulldown MenuPanes.

All the CascadeButtons and PushButtons have mnemonics.

The source code for this program is located on your system in /usr/contrib/Xm/xmpulldown.c.

```
/* Pulldown Menu Example */

#include <Xm/Xm.h>
#include <Xm/RowColumn.h>
#include <Xm/PushBG.h>
#include <Xm/Form.h>
#include <Xm/CascadeB.h>

/****Callback for the pushbuttons in the pulldown menu*********/

void ButtonCB (w, client_data, call_data)
Widget      w;    /*  widget id*/
caddr_t     client_data;/*  data from application   */
caddr_t     call_data;/*  data from widget class  */
{
   /*  print message and terminate program  */
   printf ("Button %s selected.\n", client_data);

}

/**********************Main Logic*********************************/

void main (argc, argv)
unsigned int argc;
char **argv;

{
   Widget toplevel, form, menubar;
   Widget menubarBtn[3], pulldowns[2];
   Widget buttons1[3], buttons2[3];
   Arg args [4];
```

```
/* Initialize toolkit and create form and menubar */

   toplevel = XtInitialize (argv[0], "PulldownMenu", NULL, 0,
      &argc, argv);

   XtSetArg(args[0], XmNwidth, 250);
   XtSetArg(args[1], XmNheight, 125);
   form = XmCreateForm(toplevel, "form", (ArgList) args, 2);
   XtManageChild(form);

   XtSetArg(args[0], XmNtopAttachment, XmATTACH_FORM);
   XtSetArg(args[1], XmNrightAttachment, XmATTACH_FORM);
   XtSetArg(args[2], XmNleftAttachment, XmATTACH_FORM);
   menubar = XmCreateMenuBar(form, "menubar", args, 4);
   XtManageChild(menubar);

/* Create help button in menubar */

   XtSetArg(args[0], XmNlabelString, XmStringCreate("Help",
            XmSTRING_DEFAULT_CHARSET));
   XtSetArg(args[1], XmNmnemonic, 'H');
   menubarBtn[0] = XmCreateCascadeButton(menubar, "help",
      args, 2);
   XtAddCallback(menubarBtn[0], XmNactivateCallback, ButtonCB,
      "Help");

   XtSetArg(args[0], XmNmenuHelpWidget, (XtArgVal)menubarBtn[0]);
   XtSetValues(menubar, args, 1);


/* Create 2 Pulldown MenuPanes and 2 cascade buttons in menubar */

   pulldowns[0] = XmCreatePulldownMenu(menubar, "pulldown1",
      NULL, 0);

   XtSetArg(args[0], XmNsubMenuId, pulldowns[0]);
   XtSetArg(args[1], XmNlabelString, XmStringCreate("Left Menu",
            XmSTRING_DEFAULT_CHARSET));
   XtSetArg(args[2], XmNmnemonic, 'L');
   menubarBtn[1] = XmCreateCascadeButton(menubar, "button1",
      args, 3);

   pulldowns[1] = XmCreatePulldownMenu(menubar, "pulldown2",
```

```
    NULL, 0);

XtSetArg(args[0], XmNsubMenuId, pulldowns[1]);
XtSetArg(args[1], XmNlabelString, XmStringCreate
  ("Right Menu", XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[2], XmNmnemonic, 'R');
menubarBtn[2] = XmCreateCascadeButton(menubar, "button2",
    args, 3);
XtManageChildren(menubarBtn, 3);


XtSetArg(args[0], XmNlabelString, XmStringCreate("First",
          XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[1], XmNmnemonic, 'F');
buttons1[0] = XmCreatePushButtonGadget(pulldowns[0],
    "button1a", args, 2);
XtAddCallback(buttons1[0], XmNactivateCallback, ButtonCB,
    "Left-First");

XtSetArg(args[0], XmNlabelString, XmStringCreate("Second",
          XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[1], XmNmnemonic, 'S');
buttons1[1] = XmCreatePushButtonGadget(pulldowns[0],
    "button1b", args, 2);
XtAddCallback(buttons1[1], XmNactivateCallback, ButtonCB,
    "Left-Second");

XtSetArg(args[0], XmNlabelString, XmStringCreate("Third",
          XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[1], XmNmnemonic, 'T');
buttons1[2] = XmCreatePushButtonGadget(pulldowns[0],
    "button1c", args, 2);
XtAddCallback(buttons1[2], XmNactivateCallback, ButtonCB,
    "Left-Third");
XtManageChildren(buttons1, 3);


XtSetArg(args[0], XmNlabelString, XmStringCreate("First",
          XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[1], XmNmnemonic, 'F');
buttons2[0] = XmCreatePushButtonGadget(pulldowns[1],
    "button2a", args, 2);
XtAddCallback(buttons2[0], XmNactivateCallback, ButtonCB,
```

```
        "Right-First");

    XtSetArg(args[0], XmNlabelString, XmStringCreate("Second",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'S');
    buttons2[1] = XmCreatePushButtonGadget(pulldowns[1],
        "button2b", args, 2);
    XtAddCallback(buttons2[1], XmNactivateCallback, ButtonCB,
        "Right-Second");

    XtSetArg(args[0], XmNlabelString, XmStringCreate("Third",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'T');
    buttons2[2] = XmCreatePushButtonGadget(pulldowns[1],
        "button2c", args, 2);
    XtAddCallback(buttons2[2], XmNactivateCallback, ButtonCB,
        "Right-Third");
    XtManageChildren(buttons2, 3);

/* Get and dispatch events */

    XtRealizeWidget(toplevel);

    XtMainLoop();
}
```

6.4 Creating Submenus

Submenus are implemented using Pulldown MenuPanes attached to CascadeButton widgets or gadgets. The submenu and the CascadeButton to which it is attached are children of the MenuPane (Popup or Pulldown) from which the submenu cascades.

The MenuShell of the submenu is created automatically if the submenu is created using `XmCreatePulldownMenu` convenience function:

`Widget CreatePulldownMenu(`*parent*, *name*, *arglist*, *argcount*`);`

The submenu's MenuShell is created as the child of the `parent`'s MenuShell.

6.4.1 Procedure for Creating Submenus

The following steps create a submenu of a Popup menu, and then create a submenu of that submenu. Following each step is a code segment that accomplishes the task.

1. Create the Popup MenuPane. The following line creates the Popup MenuPane as a child of widget form1.

```
popup = XmCreatePopupMenu(form1, "popup", NULL, 0);
```

2. To create a submenu, create a Pulldown MenuPane and CascadeButtonGadget as the children of the Popup MenuPane. Use the resource XmNsubMenuId to attach the CascadeButtonGadget to the MenuPane.

```
submenu1 = XmCreatePulldownMenu(popup, "submenu1",
      NULL, 0);

XtSetArg(args[0], XmNsubMenuID, submenu1);
cascade1 = XmCreateCascadeButtonGadget(popup, "cascade1",
      args, 1);
XtManageChild(cascade1);
```

3. To create a submenu of submenu1, create a Pulldown MenuPane and CascadeButtonGadget as children of submenu1.

```
submenu3 = XmCreatePulldownMenu(submenu1,
      "submenu3", NULL, 0);

XtSetArg(args[0], XmNsubMenuID, submenu3);
cascade3 = XmCreateCascadeButtonGadget(submenu1,
      "cascade3", args, 1);
XtManageChild(cascade3);
```

The following illustration shows the parenting relationships and attachments (dashed lines) to use when creating submenus in a Popup menu system using convenience functions. This system contains two submenus beneath the top level Popup MenuPane. The first submenu contains two PushButtons and one CascadeButtonGadget (CascadeButton3). The CascadeButtonGadget is used to access the submenu (Pulldown MenuPane3) that cascades from the first submenu.

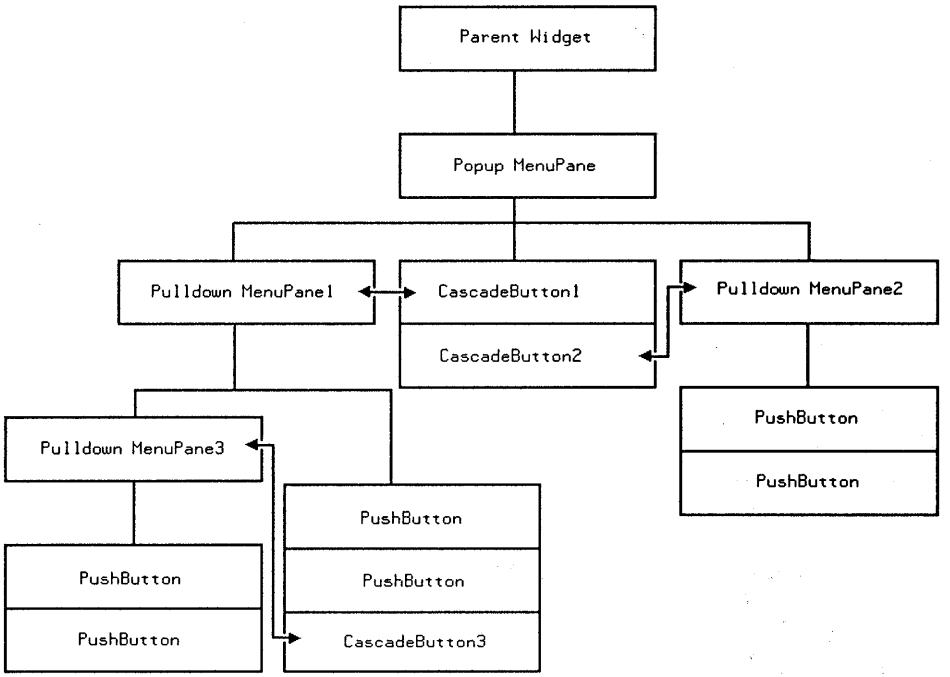


Figure 6-10. Creating Submenus With Convenience Functions

6.4.2 Interacting With Submenus

The means of interacting with submenus is explained in previous sections of this chapter, "Interacting with Popup Menus" and "Interacting With Pulldown Menus."

6.4.3 Sample Program

The following program creates the popup menu system (with submenus) that is illustrated in figure 6-10. The source code for this program is located on your system in `/usr/contrib/Xm/xmsubmenus.c`.

```
/* Popup Menu with Submenus Example */

#include <Xm/Xm.h>
#include <Xm/PushBG.h>
#include <Xm/CascadeBG.h>
#include <Xm/RowColumn.h>
```

```
/*********** Callback for the Pushbuttons *********************/

void ButtonCB (w, client_data, call_data)
Widget      w;    /*  widget id*/
caddr_t     client_data;/*  data from application   */
caddr_t     call_data;/*  data from widget class  */
{
   /*  print message and terminate program  */
   printf ("Button %s selected.\n", client_data);

}

/************** Event Handler for Popup Menu *****************/

PostIt (w, popup, event)
Widget w;
Widget popup;
XButtonEvent * event;
{
   if (event->button != Button3)
      return;
   XmMenuPosition(popup, event);
   XtManageChild(popup);
}


/******************Main Logic for Program *********************/


void main (argc, argv)
int argc;
char **argv;
{
   Widget toplevel, popup, rc;
   Widget submenu1, submenu2, submenu3, buttons[2];
   Widget popupBtn[2], sub1Btn[3], sub2Btn[2], sub3Btn[2];
   Arg args[5];

/* Initialize toolkit */
   toplevel = XtInitialize (argv[0], "PopupMenu", NULL, 0,
      &argc, argv);

/* Create RowColumn in toplevel with two pushbuttons */
```

```
    XtSetArg(args[0], XmNwidth, 400);
    XtSetArg(args[1], XmNheight, 125);
    XtSetArg(args[2], XmNresizeWidth, False);
    XtSetArg(args[3], XmNresizeHeight, False);
    XtSetArg(args[4], XmNadjustLast, False);
    rc = XmCreateRowColumn(toplevel, "rc", args, 5);
    XtManageChild(rc);

    buttons[0] = XmCreatePushButtonGadget(rc, "button1", NULL, 0);
    XtAddCallback(buttons[0], XmNactivateCallback, ButtonCB, "1");

    buttons[1] = XmCreatePushButtonGadget(rc, "button2", NULL, 0);
    XtAddCallback(buttons[1], XmNactivateCallback, ButtonCB, "2");
    XtManageChildren(buttons, 2);

/* Create popup menu */

    popup = XmCreatePopupMenu(rc, "popup", NULL, 0);
    XtAddEventHandler(rc, ButtonPressMask, False, PostIt, popup);

/* Create two submenus and CascadeButtons in the popup menu */

    submenu1 = (Widget)XmCreatePulldownMenu(popup, "submenu1",
       NULL, 0);

    XtSetArg(args[0], XmNsubMenuId, submenu1);
    XtSetArg(args[1], XmNlabelString, XmStringCreate
      ("First Submenu", XmSTRING_DEFAULT_CHARSET));
    popupBtn[0] = XmCreateCascadeButtonGadget(popup, "cbutton1",
      args, 2);

    submenu2 = (Widget)XmCreatePulldownMenu(popup, "submenu2",
       NULL, 0);

    XtSetArg(args[0], XmNsubMenuId, submenu2);
    XtSetArg(args[1], XmNlabelString, XmStringCreate
      ("Second Submenu", XmSTRING_DEFAULT_CHARSET));
    popupBtn[1] = XmCreateCascadeButtonGadget(popup, "cbutton2",
      args, 2);
    XtManageChildren(popupBtn, 2);

/* Create pushbuttons in submenu1 and submenu2. */
```

```
    sub1Btn[0] = XmCreatePushButtonGadget(submenu1, "button1a",
       NULL, 0);
    XtAddCallback(sub1Btn[0], XmNactivateCallback, ButtonCB, "1a");

    sub1Btn[1] = XmCreatePushButtonGadget(submenu1, "button1b",
       NULL, 0);
    XtAddCallback(sub1Btn[1], XmNactivateCallback, ButtonCB, "1b");

    sub2Btn[0] = XmCreatePushButtonGadget(submenu2, "button2a",
        NULL, 0);
    XtAddCallback(sub2Btn[0], XmNactivateCallback, ButtonCB, "2a");

    sub2Btn[1] = XmCreatePushButtonGadget(submenu2, "button2b",
       NULL, 0);
    XtAddCallback(sub2Btn[1], XmNactivateCallback, ButtonCB, "2b");
    XtManageChildren(sub2Btn, 2);

/* Create a submenu of submenu 1 */

    submenu3 = (Widget)XmCreatePulldownMenu(submenu1, "submenu3",
       NULL, 0);

    XtSetArg(args[0], XmNsubMenuId, submenu3);
    XtSetArg(args[1], XmNlabelString, XmStringCreate
      ("To Third Submenu", XmSTRING_DEFAULT_CHARSET));
    sub1Btn[2] = XmCreateCascadeButtonGadget(submenu1, "cbutton3",
       args, 2);
    XtManageChildren(sub1Btn, 3);

/* Create pushbuttons in submenu 3 */

    sub3Btn[0] = XmCreatePushButtonGadget(submenu3, "button3a",
       NULL, 0);
    XtAddCallback(sub3Btn[0], XmNactivateCallback, ButtonCB, "3a");

    sub3Btn[1] = XmCreatePushButtonGadget(submenu3, "button3b", NULL
       0);
    XtAddCallback(sub3Btn[1], XmNactivateCallback, ButtonCB, "3b");
    XtManageChildren(sub3Btn, 2);

/* Get and dispatch events */

    XtRealizeWidget(toplevel);
```

```
    XtMainLoop();
}
```

6.5 Creating Option Menu Systems

The basis of an Option menu system is the following:

- An Option menu. An Option menu is created by the convenience function `XmCreateOptionMenu`. It is a specialized RowColumn manager composed of two "internal" gadgets:
  - A selection area. The selection area is a specialized CascadeButtonGadget. It provides the means for displaying an associated Pulldown MenuPane, and it displays the `labelString` of the last item selected from the Pulldown MenuPane. The `XmNmenuHistory` resource defines the initial item displayed. (The default is the first item in the Pulldown MenuPane.)
  - A label. The label is a specialized LabelGadget, and is displayed to the left of the selection area.
- A Pulldown MenuPane attached to the Option menu. The Pulldown MenuPane contains a PushButton or PushButtonGadget for each available option.

The Option menu typically does not display any three-dimensional visuals around itself or its internal label. The internal CascadeButtonGadget has a three-dimensional shadow. This can be changed by the application using the standard visual-related resources.

6.5.1 Option MenuPane Create Function

An Option menu can be created using this convenience function:

`Widget XmCreateOptionMenu(`*parent*, *name*, *arglist*, *argcount*`)`

`XmCreateOptionMenu` automatically creates an Option menu and two "internal" gadgets – a CascadeButtonGadget (selection area) and LabelGadget (label area). The function returns the widget ID of the Option menu. The Option menu is created as a RowColumn widget with the `XmNrowColumnType` resource set to `XmMENU_OPTION`.

The two internal gadgets can be accessed separately using the following functions:

- `Widget XmOptionLabelGadget(`*option_menu*`)` returns the ID of the LabelGadget.
- `Widget XmOptionButtonGadget(`*option_menu*`)` returns the ID of the CascadeButtonGadget.

These functions allow the application to have more control over the visuals associated with the label and selection areas.

6.5.2 Procedure for Creating an Option Menu.

The following steps create an Option menu. Following each step is a code segment that accomplishes the task.

1. Create the Pulldown MenuPane that will contain the selection items.

```
optionsubmenu = XmCreatePulldownMenu(form1,
      "optionsubmenu", NULL, 0);
```

2. Create the selection items for the Pulldown MenuPane.

```
option[0] = XmCreatePushButtonGadget(optionsubmenu,
      "option1", NULL, 0);
option[1] = XmCreatePushButtonGadget(optionsubmenu,
      "option2", NULL, 0);
XtManageChildren(option, 2);
```

3. Use the XmCreateOptionMenu convenience function to create the Option menu and attach it to the Pulldown MenuPane. Also specify a string for the Label area.

```
string = XmStringCreate("Options:",
      XmSTRING_DEFAULT_CHARSET);
XtSetArg(args[0], XmNlabelString, string);
XtSetArg(args[1], XmNsubMenuId, optionsubmenu);
option_menu = XmCreateOptionMenu(form1, "option_menu",
      args, 2);
XmStringFree(string);
XtManageChild(option_menu);
```

The components of the Option menu system *must* be created in the order shown above. (You cannot use XtSetValues to specify the Option menu's submenu.)

The following illustration shows the parenting relationships and attachments (dashed lines) used to create Option menu systems using convenience functions. Each Pulldown MenuPane contains three options from which to choose.

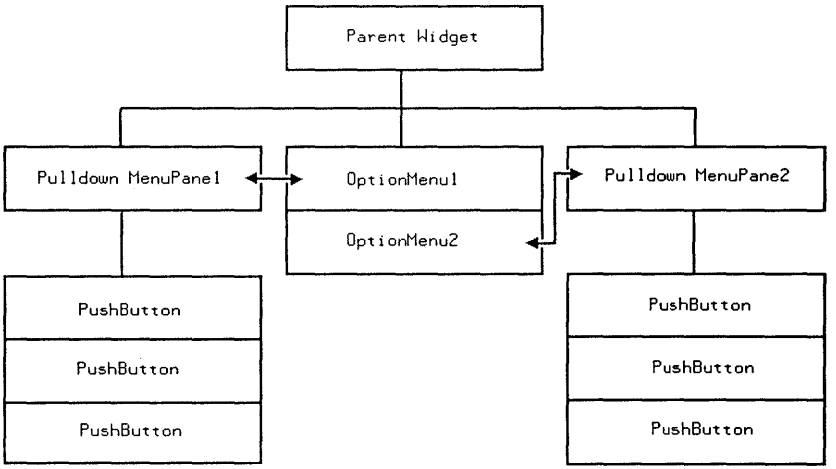


Figure 6-11. Creating Option Menu Systems Using Convenience Functions

6.5.3 Interacting With Option Menus

Mouse Input

The Pulldown MenuPane is posted by moving the mouse pointer over the selection area and pressing mouse button 1. The Pulldown MenuPane is positioned so that the last selected item is directly over the selection area; the MenuPane is *not* repositioned if a portion is inaccessible.

Menu items are armed when the pointer enters them and disarmed when it leaves. Releasing mouse button 1 while a menu item is armed selects the menu item and changes the label in the selection area.

The mouse button used to interact with Option menus can be changed using the RowColumn resource `XmNwhichButton`.

The Keyboard Interface

A mnemonic can be associated with the Option menu. Typing the mnemonic posts the Pulldown MenuPane and enables traversal.

When traversal is enabled,

- The directional keys traverse the menu hierarchy.
- The Return key selects the currently armed menu item.

- The ESC key unposts the Pulldown MenuPane.
- Pressing the mouse button disables traversal and reenables interactive operation.
- Pressing a mnemonic or accelerator for a menu item selects that item.

6.5.4 Sample Program

The following sample program contains two option menus with three options each. The Pulldown MenuPanes can be posted using the mouse or by using the mnemonics F and S. When a MenuPane has been posted using a mnemonic, a mnemonic can then be used to select an option.

The source code for this program is located on your system in /usr/contrib/Xm/xmoption.c.

```
/* Option Menu Example */

#include <Xm/Xm.h>
#include <Xm/PushBG.h>
#include <Xm/CascadeBG.h>
#include <Xm/RowColumn.h>


/*********** Callback for the Pushbuttons ********************/

void ButtonCB (w, client_data, call_data)
Widget      w;    /*  widget id*/
caddr_t     client_data;/*  data from application   */
caddr_t     call_data;/*  data from widget class  */
{
   /*  print message and terminate program  */
   printf ("Option %s selected.\n", client_data);

}

/*******************Main Logic for Program ******************/


void main (argc, argv)
int argc;
char **argv;
{
   Widget toplevel, pulldown1, pulldown2, rc;
```

```
    Widget option_menus[2], options1[3], options2[3];
    Arg args[6];

/* Initialize toolkit */
    toplevel = XtInitialize (argv[0], "OptionMenu", NULL,
       0, &argc, argv);

/* Create RowColumn in toplevel */

    XtSetArg(args[0], XmNwidth, 375);
    XtSetArg(args[1], XmNheight, 75);
    XtSetArg(args[2], XmNresizeWidth, False);
    XtSetArg(args[3], XmNresizeHeight, False);
    XtSetArg(args[4], XmNnumColumns, 2);
    XtSetArg(args[5], XmNpacking, XmPACK_COLUMN);
    rc = XmCreateRowColumn(toplevel, "rc", args, 6);
    XtManageChild(rc);

/* Create two pulldown menus in rc */

    pulldown1 = (Widget)XmCreatePulldownMenu(rc, "pulldown1",
       NULL, 0);

    XtSetArg(args[0], XmNlabelString, XmStringCreate("A-option",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'A');
    options1[0] = XmCreatePushButtonGadget(pulldown1, "option1a",
       args, 2);
    XtAddCallback(options1[0], XmNactivateCallback, ButtonCB, "1A");

    XtSetArg(args[0], XmNlabelString, XmStringCreate("B-option",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'B');
    options1[1] = XmCreatePushButtonGadget(pulldown1, "option1b",
       args, 2);
    XtAddCallback(options1[1], XmNactivateCallback, ButtonCB, "1B");

    XtSetArg(args[0], XmNlabelString, XmStringCreate("C-option",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'C');
    options1[2] = XmCreatePushButtonGadget(pulldown1, "option1c",
       args, 2);
    XtAddCallback(options1[2], XmNactivateCallback, ButtonCB, "1C");
```

```
    XtManageChildren(options1, 3);


    pulldown2 = (Widget)XmCreatePulldownMenu(rc, "pulldown2",
       NULL, 0);

    XtSetArg(args[0], XmNlabelString, XmStringCreate("A-option",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'A');
    options2[0] = XmCreatePushButtonGadget(pulldown2, "option2a",
       args, 2);
    XtAddCallback(options2[0], XmNactivateCallback,
       ButtonCB, "2A");

    XtSetArg(args[0], XmNlabelString, XmStringCreate("B-option",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'B');
    options2[1] = XmCreatePushButtonGadget(pulldown2, "option2b",
       args, 2);
    XtAddCallback(options2[1], XmNactivateCallback,
       ButtonCB, "2B");

    XtSetArg(args[0], XmNlabelString, XmStringCreate("C-option",
             XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'C');
    options2[2] = XmCreatePushButtonGadget(pulldown2, "option2c",
       args, 2);
    XtAddCallback(options2[2], XmNactivateCallback,
       ButtonCB, "2C");
    XtManageChildren(options2, 3);

/* Create option menus and attach the two pulldown menus */

    XtSetArg(args[0], XmNlabelString, XmStringCreate
      ("First Option Set:", XmSTRING_DEFAULT_CHARSET));
    XtSetArg(args[1], XmNmnemonic, 'F');
    XtSetArg(args[2], XmNsubMenuId, pulldown1);
    XtSetArg(args[3], XmNmenuHistory, options1[2]);
    option_menus[0] = XmCreateOptionMenu(rc, "option_menu1",
       args, 4);

    XtSetArg(args[0], XmNlabelString, XmStringCreate
      ("Second Option Set:", XmSTRING_DEFAULT_CHARSET));
```

```
    XtSetArg(args[1], XmNmnemonic, 'S');
    XtSetArg(args[2], XmNsubMenuId, pulldown2);
    XtSetArg(args[3], XmNmenuHistory, options2[0]);
    option_menus[1] = XmCreateOptionMenu(rc, "option_menu2",
        args, 4);
    XtManageChildren(option_menus, 2);

/* Get and dispatch events */

    XtRealizeWidget(toplevel);

    XtMainLoop();
}
```

6.6 Selecting A Menu Cursor

An application can select a specific menu cursor that is used whenever a menu is displayed. This feature provides consistent appearance within menus that belong to the same application. The default menu cursor is `arrow`.

The menu cursor can be specified at the application start-up time by the resource `XmNmenuCursor`. This resource is *not* associated with a particular widget, and can *only* be set at application start-up time. The resource can be set two ways:

- By setting a resource in a defaults file. For example,

  ```
  *menuCursor: star
  ```

  sets star as the menu cursor.

- By using the `-xrm` command line argument. For example, the following command line specifies a clock as the menu cursor for the application named "myprog":

  ```
  myprog -xrm "*menuCursor:  clock"
  ```

The cursor can be specified programmatically using the `XmSetMenuCursor` function:

```
void XmSetMenuCursor(display, cursorId)
        Display *display;
        Cursor cursorId;
```

display Specifies the display for which the cursor is used.

cursorId Specifies the menu cursor.

After the function is executed, any menu displayed by the application on the specified display uses the menu cursor identified in the variable *cursorId*. This allows the application to use different menu cursors on different displays.

The function `XmGetMenuCursor` returns the cursor ID of the current menu cursor for a specified display:

```
Cursor XmGetMenuCursor(display)
       Display *display;
```

display Specifies the display.

If the application has not created any menus, no cursor is defined and the function returns the value "None."

The following list shows the valid cursor names.

arrow	based_arrow_down	based_arrow_up	boat
bogosity	bottom_left_corner	bottom_right_corner	bottom_side
bottom_tee	box_spiral	centr_ptr	circle
clock	coffee_mug	cross	cross_reverse
crosshair	diamond_cross	dot	dotbox
double_arrow	draft_large	draft_small	exchange
fleur	gobbler	gumby	hand1
hand2	heart	icon	iron_cross
left_ptr	left_side	left_tee	leftbutton
ll_angle	lr_angle	man	middlebutton
mouse	pencil	pirate	plus
question_arrow	right_ptr	right_side	right_tee
rightbutton	rtl_logo	sailboat	sb_down_arrow
sb_h_double_arrow	sb_left_arrow	sb_right_arrow	sb_up_arrow
sb_v_double_arrow	shuttle	sizing	spider
spraycan	star	target	tcross
top_left_arrow	top_left_corner	top_right_corner	top_side
top_tee	trek	ul_angle	umbrella
watch	xterm		

6.7 Creating Menus Without Using Convenience Functions

Applications that use the menu system convenience functions do not need to explicitly create MenuShell widgets; the `XmCreatePopupMenu` and `XmCreatePulldownMenu` functions create a Popup or Pulldown MenuPane and the parent MenuShell.

If an application requires access to individual MenuShells in an application, the MenuShells and MenuPanes can be created by using the standard X Toolkit create routines or by using the create functions for MenuShells and RowColumn Widgets.

6.7.1 Functions for Creating Menus

Three functions are used in creating menu systems.

- The MenuShell specific create function:

  `Widget XmCreateMenuShell` (*parent*, *name*, *arglist*, *argcount*)

  creates an instance of a MenuShell widget and returns the associated widget ID.

- The X Toolkit function:

  `Widget XtCreatePopupShell`(*name*, *widget_class*, *parent*, *args*, *num_args*)

  can be used to create a MenuShell for a Popup or Pulldown MenuPane.

- The RowColumn specific create function:

  `Widget XmCreateRowColumn` (*parent*, *name*, *arglist*, *argcount*)

  creates an instance of a RowColumn widget and returns the associated widget ID.

6.7.2 Parenting Relationships

The parenting relationships required to create a menu system without using convenience functions depend on the type of menu system being built:

- If the MenuShell is for a Popup MenuPane, the MenuShell must be the parent of the Popup MenuPane (see figure 6-12).
- If the MenuShell is for a MenuPane that is pulled down from a MenuBar, the MenuShell must be created as a child of the MenuBar (see figure 6-13).
- If the MenuShell is for a submenu MenuPane that is pulled down from a Popup or another Pulldown MenuPane, the MenuShell must be created as a child of the Popup or Pulldown MenuPane's parent MenuShell (see figure 6-14).

- If the MenuShell is for a Pulldown MenuPane in an Option menu, the MenuShell must have the same parent as the Option menu (see figure 6-15).

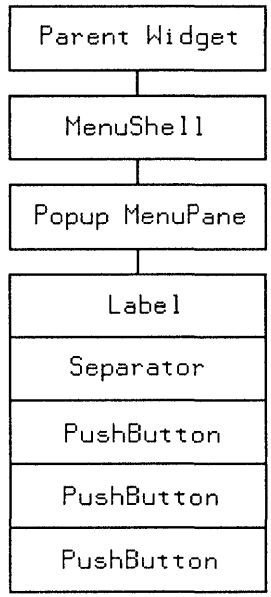


Figure 6-12. Creating a Popup Menu System Without Using Convenience Functions

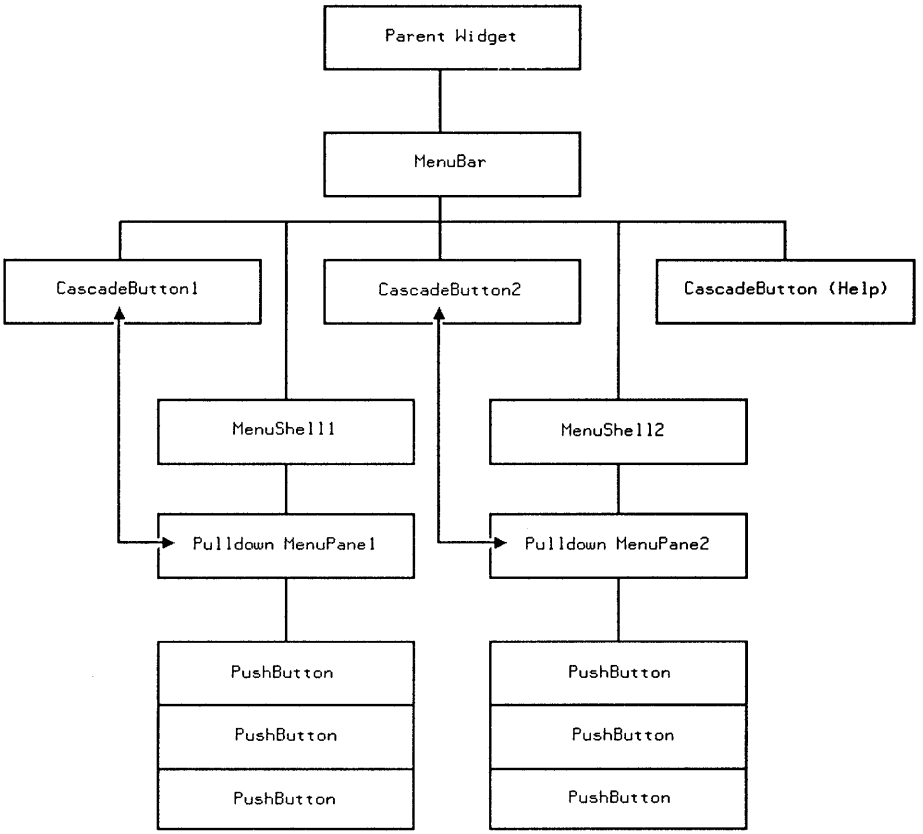


Figure 6-13. Creating a Pulldown Menu System Without Using Convenience Functions

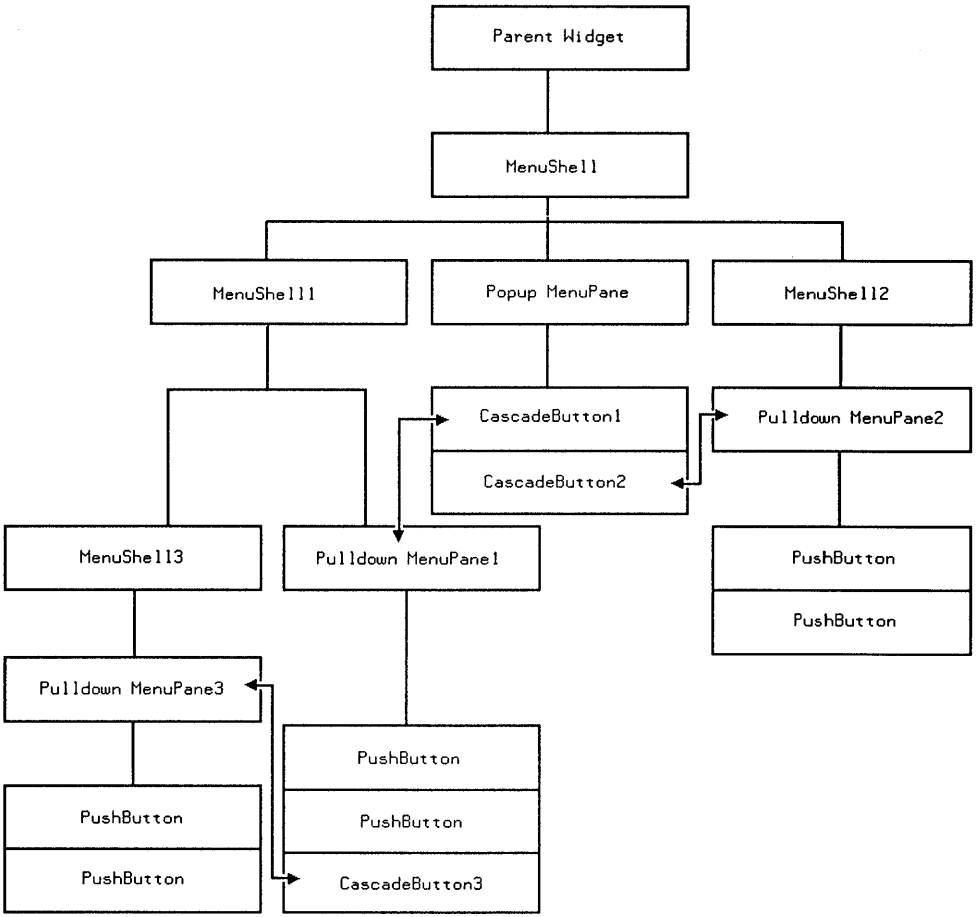


Figure 6-14. Creating Submenus Without Using Convenience Functions

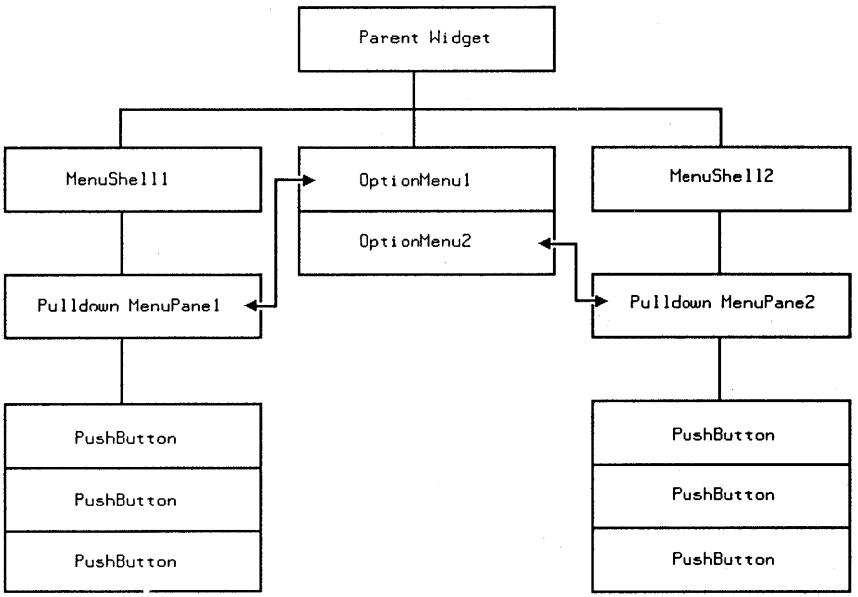


Figure 6-15. Creating an Option Menu System Without Using Convenience Functions

6.7.3 Sample Program

The following program creates a Popup menu system without using the convenience functions. Figure 6-14 illustrates the menu system created by this program.

The source code for this program is located on your system in `/usr/contrib/Xm/xmmenushel.c`.

```
/* Popup Menu and Submenus created with MenuShells */

#include <Xm/Xm.h>
#include <Xm/MenuShell.h>
#include <Xm/PushBG.h>
#include <Xm/CascadeBG.h>
#include <Xm/RowColumn.h>


/*********** Callback for the Pushbuttons *********************/

void ButtonCB (w, client_data, call_data)
```

```
Widget      w;    /*  widget id*/
caddr_t     client_data;/*  data from application   */
caddr_t     call_data;/*  data from widget class  */
{
   /*  print message and terminate program  */
   printf ("Button %s selected.\n", client_data);

}

/************** Event Handler for Popup Menu *****************/

PostIt (w, popup, event)
Widget w;
Widget popup;
XButtonEvent * event;
{
   if (event->button != Button3)
      return;
   XmMenuPosition(popup, event);
   XtManageChild(popup);
}


/************************Main Logic for Program **************/


void main (argc, argv)
int argc;
char **argv;
{
   Widget toplevel, rc, buttons[2];
   Widget popupshell, mshell1, mshell2, mshell3;
   Widget popup, submenu1, submenu2, submenu3;
   Widget popupBtn[2], sub1Btn[3], sub2Btn[2], sub3Btn[2];
   Arg args[5];

/* Initialize toolkit */

   toplevel = XtInitialize (argv[0], "PopupMenu", NULL, 0,
      &argc, argv);

/* Create RowColumn in toplevel with two pushbuttons */
```

```
    XtSetArg(args[0], XmNwidth, 150);
    XtSetArg(args[1], XmNheight, 50);
    XtSetArg(args[2], XmNresizeWidth, False);
    XtSetArg(args[3], XmNresizeHeight, False);
    XtSetArg(args[4], XmNadjustLast, False);
    rc = XmCreateRowColumn(toplevel, "rc", args, 5);
    XtManageChild(rc);

    buttons[0] = XmCreatePushButtonGadget(rc, "button1", NULL, 0);
    XtAddCallback(buttons[0], XmNactivateCallback, ButtonCB, "1");

    buttons[1] = XmCreatePushButtonGadget(rc, "button2", NULL, 0);
    XtAddCallback(buttons[1], XmNactivateCallback, ButtonCB, "2");
    XtManageChildren(buttons, 2);

/* Create MenuShell for a Popup MenuPane */

    XtSetArg(args[0], XmNheight, 100);
    XtSetArg(args[1], XmNwidth, 100);
    popupshell = XmCreateMenuShell(rc, "popupshell", args, 2);

/* Create RowColumn Widget configured as Popup MenuPane */

    XtSetArg(args[0], XmNrowColumnType, XmMENU_POPUP);
    popup = XmCreateRowColumn (popupshell, "popup", args, 1);
    XtAddEventHandler(rc, ButtonPressMask, False, PostIt, popup);

/* Create MenuShells and Pulldown MenuPanes for two submenus */

    XtSetArg(args[0], XmNheight, 100);
    XtSetArg(args[1], XmNwidth, 100);
    mshell1 = XmCreateMenuShell (popupshell, "mshell1", args, 2);

    XtSetArg(args[0], XmNrowColumnType, XmMENU_PULLDOWN);
    submenu1 = XmCreateRowColumn (mshell1, "submenu1", args, 1);

    XtSetArg(args[0], XmNheight, 100);
    XtSetArg(args[1], XmNwidth, 100);
    mshell2 = XmCreateMenuShell (popupshell, "mshell2", args, 2);

    XtSetArg(args[0], XmNrowColumnType, XmMENU_PULLDOWN);
    submenu2 = XmCreateRowColumn (mshell2, "submenu2", args, 1);
```

```
/* Create two Cascade Buttons in the Popup MenuPane */

   XtSetArg(args[0], XmNsubMenuId, submenu1);
   XtSetArg(args[1], XmNlabelString, XmStringCreate
     ("First Submenu", XmSTRING_DEFAULT_CHARSET));
   popupBtn[0] = XmCreateCascadeButtonGadget(popup, "cbutton1",
      args, 2);

   XtSetArg(args[0], XmNsubMenuId, submenu2);
   XtSetArg(args[1], XmNlabelString, XmStringCreate
     ("Second Submenu", XmSTRING_DEFAULT_CHARSET));
   popupBtn[1] = XmCreateCascadeButtonGadget(popup, "cbutton2",
      args, 2);
   XtManageChildren (popupBtn, 2);

/* Create pushbuttons in MenuPanes submenu1 and submenu2 */

   sub1Btn[0] = XmCreatePushButtonGadget(submenu1, "button1a",
      NULL, 0);
   XtAddCallback(sub1Btn[0], XmNactivateCallback, ButtonCB, "1a");

   sub1Btn[1] = XmCreatePushButtonGadget(submenu1, "button1b",
      NULL, 0);
   XtAddCallback(sub1Btn[1], XmNactivateCallback, ButtonCB, "1b");

   sub2Btn[0] = XmCreatePushButtonGadget(submenu2, "button2a",
      NULL, 0);
   XtAddCallback(sub2Btn[0], XmNactivateCallback, ButtonCB, "2a");

   sub2Btn[1] = XmCreatePushButtonGadget(submenu2, "button2b",
      NULL, 0);
   XtAddCallback(sub2Btn[1], XmNactivateCallback, ButtonCB, "2b");
   XtManageChildren (sub2Btn, 2);


/* Create a MenuShell for the submenu of submenu1 */

   XtSetArg(args[0], XmNheight, 100);
   XtSetArg(args[1], XmNwidth, 100);
   mshell3 = XmCreateMenuShell (mshell1, "mshell3", args, 2);

/* Create the MenuPane for the submenu of submenu1 */
```

```
    XtSetArg(args[0], XmNrowColumnType, XmMENU_PULLDOWN);
    submenu3 = XmCreateRowColumn (mshell3, "submenu3", args, 1);

/* Create the Cascade Button in submenu1 for accessing submenu3 */

    XtSetArg(args[0], XmNsubMenuId, submenu3);
    XtSetArg(args[1], XmNlabelString, XmStringCreate
      ("To Third Submenu", XmSTRING_DEFAULT_CHARSET));
    sub1Btn[2] = XmCreateCascadeButtonGadget(submenu1, "cbutton3",
       args, 2);
    XtManageChildren(sub1Btn, 3);

/* Create pushbuttons in submenu */

    sub3Btn[0] = XmCreatePushButtonGadget(submenu3, "button3a",
       NULL, 0);
    XtAddCallback(sub3Btn[0], XmNactivateCallback, ButtonCB, "3a");

    sub3Btn[1] = XmCreatePushButtonGadget(submenu3, "button3b",
       NULL, 0);
    XtAddCallback(sub3Btn[1], XmNactivateCallback, ButtonCB, "3b");
    XtManageChildren (sub3Btn, 2);

/* Get and dispatch events */

    XtRealizeWidget(toplevel);

    XtMainLoop();
}
```

Specialized Widgets 7

There are several widgets in the OSF/Motif widget set that fall into the category of "specialized widgets." This is because these widgets are somewhat more complex to use and accomplish more complex feats than the "normal" widget. There are four widgets that fall into this category:

- Form
- List
- RowColumn
- Text

The Form widget is explained in chapter 5, "Dialog Widgets." The others are explained in this chapter.

7.1 List Widget

The List widget allows you to make a selection from a list of items. The application defines an array of compound strings, each of which becomes an item in the list. You can set the number of items in the list that are to be visible. You can also choose to have the List appear with a ScrollBar so that you can scroll through the list of items. Items are selected by moving the pointer to the desired item and pressing the mouse button or key defined as "select." The selected item is displayed in inverse color.

7.1.1 List Functions

There are a number of functions associated with the List widget that are available to perform a variety of tasks. These functions are listed below and each has its own man page in the *HP OSF/Motif Programmer's Reference Manual.*

TABLE 7-1. List Widget Functions

Function	Description
XtCreateWidget	Basic widget create function.
XmCreateList	Specific create function for XmList.
XmCreateScrolledList	Create function for ScrolledList.
XmListAddItem	Add an item (possibly selected) to the list.
XmListAddItemUnselected	Add an item (unselected) to the list.
XmListDeleteItem	Delete item from the list.
XmListDeletePos	The item at the specified position is deleted.
XmListDeselectItem	If the specified item is currently selected, it is unhighlighted and removed from the selected list.
XmListDeselectAllItems	*All* the currently selected items in the list are unhighlighted and removed from the selection list.
XmListSelectItem	The item is highlighted and added to the current select list.
XmListSetHorizPos	If the horizontal ScrollBar is visible, the XmNvalue resource of the ScrollBar is set to the specified position and the visible portion of the List is updated.
XmListSetItem	Make the specified item the first in the list.
XmListSetPos	Make the item at the specified position the first visible position.
XmListSetBottomItem	Make the specified item the last visible item in the list.
XmListSetBottomPos	Make the item in the specified position the last visible position.
XmListSelectPos	The item in the specified position is highlighted and added to the current selected list.
XmListDeselectPos	Delete the item at the specified position from the selected list and unhighlight it.
XmListItemExists	The function returns True if the specified item exists.

The use of many of these functions will be described in the sections that follow. Actual code segments accompanied by illustrations show the results of certain programming actions.

7.1.2 Using the List Widget

Figure 7-1 shows an example of a list widget. You have seen other examples in earlier chapters of this manual. For example, the lists displayed in the FileSelectionBox and SelectionBox widgets are actually List widgets.

Figure 7-1. List Widget

The program that produces the List widget shown above can be found in the file `/usr/contrib/Xm/xmlist.c`. You can compile and link this program using the procedure described in chapter 1. Segments of this program are used in this section to describe how to accomplish certain functions associated with the list widget.

In figure 7-1, the two lines that appear in inverse color indicate items that have been selected by the application. Any number of items can be selected. This is accomplished as shown in the following code segment.

```
Arg Args[20];

static char *CharSelectedItems[2] = {
"New Item List",
"New Policy"};

#define NUM_SELECTED_ITEMS 2
int i;
XmStringCharSet  cs = "ISOLatin1";
XmString SelectedItems[NUM_SELECTED_ITEMS];

/* Create compound strings for selected items */
for (i = 0; i < NUM_SELECTED_ITEMS; i++)
SelectedItems[i] =
   (XmString) XmStringCreateLtoR(CharSelectedItems[i],cs);

/* Set the resource values */
i = 0;
XtSetArg(args[n], XmNselectedItems, (XtArgVal) SelectedItems); i++;
XtSetArg(args[n], XmNselectedCount,
   (XtArgVal) NUM_SELECTED_ITEMS); i++;
```

As you can see, you can include as many items as you want in the selected list.

In figure 7-2 you can see that one of the items in the list is "Five Visible." To select this item, move the pointer into the window and position it anywhere on the "Five Visible" line. Clicking mouse button 1 selects the item and it is highlighted, as shown in figure 7-2.

Figure 7-2. List Widget Before Selection Action

When you "double click" on mouse button 1, the action is performed. In this case, now only five items are displayed in the list, as shown in figure 7-3.

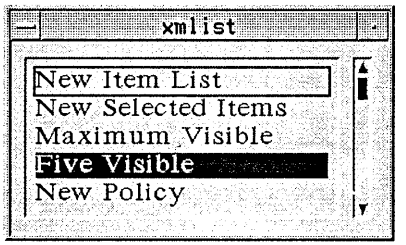


Figure 7-3. List Widget After Selecting "Five Visible"

Note the ScrollBar in the figure 7-3. There are the same number of items in the list as

before, but now only five of them are visible at any given time.

Callback

The callback procedure that is executed when a double click occurs performs the necessary steps to accomplish the task given by the selected item in the list. In the above situation, the task was to make "Five Visible." The code segments to accomplish this are shown below.

```
/* Add the callback */
XtAddCallback(outer_box, XmNdefaultActionCallback,
   DoubleClickProc, NULL);

/*************************************************************************
 *
 * DoubleClickProc is the XmDEFAULT_ACTION callback.  It functions
 * a big case statement, comparing the item that was double-clicked
 * to the items in the list.  When it finds a match, it takes the
 * appropriate action.
 *
 *************************************************************************
void  DoubleClickProc(w,closure,call_data)
  Widget w;
  caddr_t  closure, call_data;
{
  int j;
  XmListCallbackStruct *cb =  (XmListCallbackStruct *)call_data;
  int   i = 0;
  unsigned char k;

  DumpListCBStruct(call_data);

  if (XmStringCompare(cb->item,ListItems[0]))
  /* Set a new item list */
  {
     XtSetArg(Args[i], XmNitems, (XtArgVal) NewListItems); i++;
     XtSetArg(Args[i], XmNitemCount, (XtArgVal) NUM_NEW_LIST_ITEMS)
     i++;
  }

  if (XmStringCompare(cb->item,NewListItems[0]))
 /* Set the original Item List */
  {
     XtSetArg(Args[i], XmNitems, (XtArgVal) ListItems); i++;
```

```
    XtSetArg(Args[i], XmNitemCount, (XtArgVal) NUM_LIST_ITEMS);
    i++;
}

if (XmStringCompare(cb->item,ListItems[1]))
/* Set a new Selected Item List */
{
 XtSetArg(Args[i], XmNselectedItems, (XtArgVal) SelectedItems);
 i++;
 XtSetArg(Args[i], XmNselectedItemCount,
  (XtArgVal) NUM_SELECTED_ITEMS); i++;
}

if (XmStringCompare(cb->item,ListItems[2]))
/* Make all items visible by getting the current item  */
{
 XtSetArg(Args[0], XmNitemCount, &j);
 XtGetValues(w, Args, 1);
 XtSetArg(Args[i], XmNvisibleItemCount, (XtArgVal) j); i++;
}

if (XmStringCompare(cb->item,ListItems[3]))
/* Make five items visible */
{
 XtSetArg(Args[i], XmNvisibleItemCount, (XtArgVal) 5); i++;
}

if (XmStringCompare(cb->item,ListItems[4]))
/* Set a new selection policy */
{
 XtSetArg(Args[0], XmNselectionPolicy, &k);
 XtGetValues(w, Args, 1);
 if (k == XmSINGLE_SELECT)
  k = XmMULTIPLE_SELECT;
  else
  if (k == XmMULTIPLE_SELECT)
     k = XmBROWSE_SELECT;
  else
    if (k == XmBROWSE_SELECT)
     k = XmEXTENDED_SELECT;
    else
    if (k == XmEXTENDED_SELECT)
       k = XmSINGLE_SELECT;
```

```
    XtSetArg(Args[i], XmNselectionPolicy, (XtArgVal)k ); i++;
   }

   if (XmStringCompare(cb->item,ListItems[5]))
   /* Increase the spacing between items */
   {
    Spacing += 2;
    XtSetArg(Args[i], XmNlistSpacing, (XtArgVal)Spacing ); i++;
   }

   if (XmStringCompare(cb->item,ListItems[6]))
   /* Change the font the items are displayed in */
   {
    if (curfont == font1)
       curfont = font2;
    else
       if (curfont == font2)
         curfont = font3;
      else
         curfont = font1;
    XtSetArg(Args[i], XmNfontList, (XtArgVal) curfont ); i++;
   }

   if (XmStringCompare(cb->item,ListItems[21]))
   /* Set automatic selection ON */
   {
    XtSetArg(Args[i], XmNautomaticSelection, (XtArgVal) TRUE);
    i++;
   }

   if (XmStringCompare(cb->item,ListItems[22]))
   /* Set automatic selection OFF */
   {
    XtSetArg(Args[i], XmNautomaticSelection, (XtArgVal) FALSE);
    i++;
   }
/****************
 *
 * If we have set any arguments, do the SetValues and return.
 *
 ****************/
   if (i > 0)
   {
```

```
  XtSetValues(w,Args,i);
  return;
}

if (XmStringCompare(cb->item,ListItems[7]))
/* Add an item at the first position in the list   */
{
 XmListAddItem(w,FirstItem,1);
}

if (XmStringCompare(cb->item,ListItems[8]))
/* Add an item at the last position in the list   */
{
 XmListAddItem(w,LastItem,0);
}

if (XmStringCompare(cb->item,ListItems[9]))
/* Add an item at the fifth position in the list   */
{
 XmListAddItem(w,MiddleItem,5);
}

if (XmStringCompare(cb->item,ListItems[10]))
/* Delete the 'Middle Item' list element       */
{
 XmListDeleteItem(w, MiddleItem);
}

if (XmStringCompare(cb->item,ListItems[11]))
/* Delete the last item   */
{
 XmListDeletePos(w,0);
}

if (XmStringCompare(cb->item,ListItems[12]))
/* Select the first item */
{
  XmListSelectPos(w,1,TRUE);
}

if (XmStringCompare(cb->item,ListItems[13]))
/* Select the last item */
{
```

```
    XmListSelectPos(w,0,TRUE);
  }

  if (XmStringCompare(cb->item,ListItems[14]))
  /* Deselect the first item */
  {
    XmListDeselectPos(w,1);
  }

  if (XmStringCompare(cb->item,ListItems[15]))
  /* Deselect the last item */
  {
    XmListDeselectPos(w,0);
  }

  if (XmStringCompare(cb->item,ListItems[16]))
  /* Deselect all selected items */
  {
    XmListDeselectAllItems(w);
  }

  if (XmStringCompare(cb->item,ListItems[17]))
  /* Make the fifth item the top */
  {
    XmListSetPos(w,5);
  }

  if (XmStringCompare(cb->item,ListItems[18]))
  /* Make the first item the top */
  {
    XmListSetPos(w,1);
  }

  if (XmStringCompare(cb->item,ListItems[19]))
  /* Make the fifth item bottom */
  {
    XmListSetBottomPos(w,5);
  }

  if (XmStringCompare(cb->item,ListItems[20]))
  /* Make the last item the bottom */
  {
    XmListSetBottomPos(w,0);
```

```
    }

    if (XmStringCompare(cb->item,ListItems[23]))
    /* End the program. */
    {
      exit(0);
    }

}
```

This callback is the heart of the program. Basically, it compares the current selected item in the callback structure (see the List man page for a description of this structure) to the the original list of items. When a match is found, the necessary action is taken. For example, if "Five Visible" is selected, the resource XmNvisibleItemCount is reset to 5 and XtSetValues is called. Note the use of the functions presented in the table at the beginning of this chapter. For example, if "Add Item at Top" is selected, the function `XmListAddItem` is called with the position value set to 1. If "Add Item at End" is selected, `XmListAddItem` is called with the position value set to 0. You can get an idea of how other list functions are used by examining the rest of the callback code segment.

Selection Policies

The List widget has four selection policies that can be set programmatically. The policies are defined below.

- Single Selection. This policy allows you to move the pointer to the list item you want and when you click mouse button 1, that item is highlighted. No highlighting or selection occurs while you are moving the pointer, and only one item can be selected at a time.
- Multiple Selection. This policy allows you to select more than one item on the list. Selection and highlighting occur only after you click mouse button 1, but you can select as many items as you want. You can *deselect* any selected item by clicking mouse button 1 when the pointer is on that item.
- Extended selection. This policy allows you to select more than one item on the list *without clicking mouse button 1 for each item*. When you press and hold mouse button one on an item and then drag the pointer up or down from that point, all items between the initial item and the pointer are highlighted. Relasing mouse button 1 stops the selection proces and those items selected remain highlighted.
- Browse selection. When you press and hold mouse button 1 on an item, that item is selected and highlighted. Dragging the pointer up or down from that point causes each succeeding item to be selected and highlighted while the preceding item is unselected and unhighlighted. When you release the mouse button, the item on which the pointer rests is selected and highlighted.

The policy selections can be changed in the sample program by double clicking on the "New Policy" item.

Figure 7-4. List Widget Multiple Selection

Figures 7-2 and 7-3 show single selection, figure 7-4 shows multiple selection, and figure 7-5 shows extended selection.

Figure 7-5. List Widget Extended Selection

The code segments shown in the Callback section provide an example of how selection policy can be changed programmatically. The appropriate segment from that section is shown below.

```
if (XmStringCompare(cb->item,ListItems[4]))
/* New Selection policy */
{
XtSetArg(myArgs[0], XmNselectionPolicy, &k);
XtGetValues(w, Args, 1);
if (k == XmSINGLE_SELECT)
 k = XmMULTIPLE_SELECT;
 else
  if (k == XmMULTIPLE_SELECT)
   k = XmBROWSE_SELECT;
  else
```

```
    if (k == XmBROWSE_SELECT)
     k = XmEXTENDED_SELECT;
    else
   if (k == XmEXTENDED_SELECT)
      k = XmSINGLE_SELECT;
/* Set the new values into the widget */
 XtSetArg(Args[i], XmNselectionPolicy, (XtArgVal)k);
```

The variable k is defined as an unsigned char to match the type of the resource XmNselectionPolicy. The argument array is set up with a call to `XtSetArg`, and the current selection policy is obtained by the call to `XtGetValues`. The selection policy is then changed and the new selection policy is reset into the widget with a call to `XtSetArg`.

7.2 RowColumn Widget

The RowColumn widget is a general purpose RowColumn manager capable of containing any widget type as a child. It requires no special knowledge about how its children function, and provides nothing above and beyond support for several different layout styles.

The RowColumn widget has no three-dimensional visuals associated with it. If an application wishes to have a three-dimensional shadow placed around the RowColumn widget, it should create the RowColumn as a child of a Frame widget.

7.2.1 RowColumn Types

The OSF/Motif system provides several types of RowColumn widgets. The widget type is specified using the XmNrowColumnType resource. The possible settings for this resource are as follows:

- XmWORK_AREA. The XmWORK_AREA type provides the generalized RowColumn manager. It is the default type when the widget is created using the `XmCreateRowColumn` function or the `XtCreateWidget` X-toolkit function.
- Four settings for creating OSF/Motif menus.
  - XmMENU_POPUP
  - XmMENU_BAR
  - XmMENU_PULLDOWN
  - XmMENU_OPTION

The specific create functions for Popup MenuPanes, MenuBars, Pulldown MenuPanes, and Option menus create RowColumn widgets set to these types.

The various types of menus are covered in chapter 6. The rest of this section deals only with the XmWORK_AREA type.

7.2.2 RowColumn Functions

The following functions create RowColumn widgets of default type XmWORK_AREA:

```
Widget XtCreateWidget (name, xmRowColumnWidgetClass, parent,
arglist, argcount)
```

```
Widget XmCreateRowColumn (parent, name, arglist, argcount)
```

Both create functions create an instance of a RowColumn widget and return the associated widget ID. `XtCreateWidget()` is the standard X Toolkit create function. `XmCreateRowColumn()` is the RowColumn specific create function.

7.2.3 Layout

RowColumn provides a variety of resources that determine the type of layout performed. For example, resources control these attributes:

- Sizing. The size of the widget can be set explicitly (by resource settings), or the size can be set automatically according to requirements of the children and their specified layout.
- Orientation. RowColumn can be configured to lay out its children in a column fashion (vertical) or a row fashion (horizontal).
- Packing. The children can be packed together tightly (not in an organized grid of rows and columns); or, all children can be placed in identically sized boxes, thus producing a symmetrical-looking arrangement of the children. Another alternative allows the application to specify the exact x and y positions of the children.
- Spacing between children. The application can control the spacing between the rows and columns.
- Margin spacing. Resources set the spacing between the edges of the RowColumn widget and the children placed along the edge.

Sizing

When XmNresizeHeight and/or XmNresizeWidth are set to True, RowColumn will request new dimensions from its parent, if necessary. The resources should be set to False if the application wants to control the dimensions.

When the XmNpacking resource is set to XmPACK_NONE, the RowColumn widget expands, if necessary, to enclose its children.

Orientation

The orientation is set using the XmNorientation resource. There are two possible settings:

- XmVERTICAL (default). This specifies a "column-major" orientation. In a column major orientation, children are laid out in columns from top to bottom.
- XmHORIZONTAL. This specifies a "row-major" orientation. In a row major orientation, children are laid out in rows, from left to right.

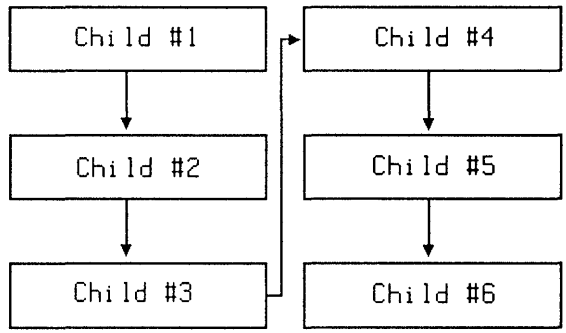


Figure 7-6. Column-Major Orientation (XmVERTICAL)

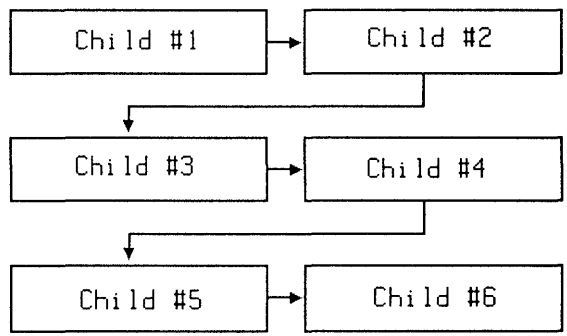


Figure 7-7. Row-Major Orientation (XmHORIZONTAL)

Packing

The XmNpacking resource determines how the items in a RowColumn widget are packed together. There are three possible settings:

- XmPACK_TIGHT (default). The layout depends on the orientation:
  - For Xm_VERTICAL orientation, items are placed one after another in a given column until there is no room left for another item. Wrapping then occurs to the next column and continues until all the children have been placed. The boxes in a given column are set to the same width, based on the widest box in that column. Thus, the items are stacked vertically but may be staggered horizontally.

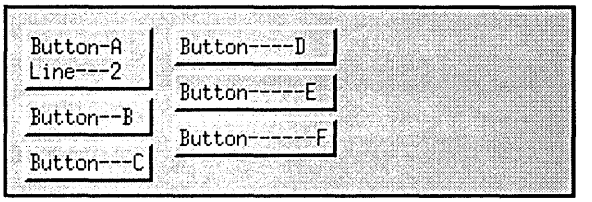


Figure 7-8. XmRowColumn Widget

  - For Xm_HORIZONTAL orientation, items are placed one after another in a given row until there in no room left for another item. Wrapping then occurs to the next row and continues until all the children have been placed. Boxes in a given row are set to the same height, based on the highest box in that row. Thus, the items are layered horizontally but may be staggered vertically.

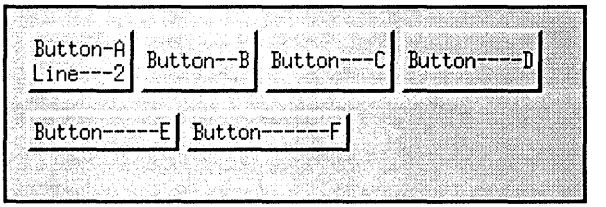


Figure 7-9. XmPACK TIGHT with XmHORIZONTAL Orientation

- XmPACK_COLUMN. In PACK_COLUMN packing, children are placed in identically sized boxes so that the layout becomes a grid. The height of the boxes is the height of the highest child; similarly, the width of the boxes is the width of the widest child.

  The XmNnumColumns resource specifies how many columns (for XmVERTICAL orientation) or rows (for XmHORIZONTAL orientation) are built. There is no automatic wrapping when a column or row is too long to fit.

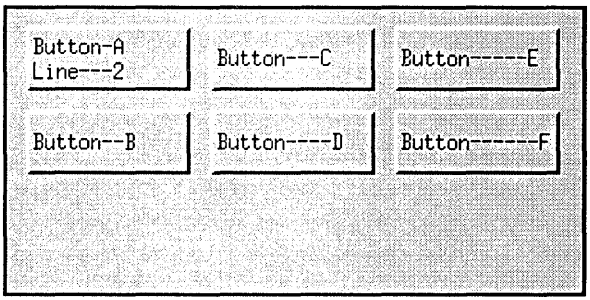


Figure 7-10. XmPACK_COLUMN With XmVERTICAL Orientation and XmNnumColumns = 3

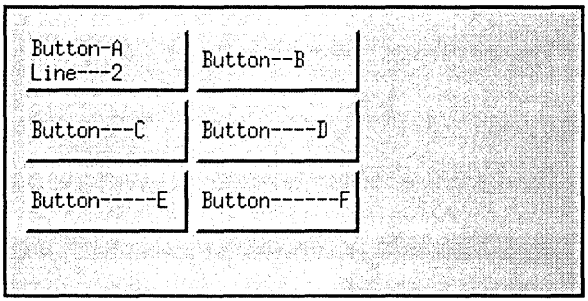


Figure 7-11. XmPACK_COLUMN With XmHORIZONTAL Orientation and XmNnumColumns = 3

- XmPACK_NONE. When there is no packing, children are positioned according to the x and y positions specified by their resources. If necessary, the RowColumn widget will attempt to become large enough to enclose all its children.

Spacing Between Children

The XmNspacing resource specifies the horizontal and vertical spacing, in pixels, between items within the RowColumn widget. The default is one pixel.

Margin Spacings

The XmNmarginHeight and XmNmarginWidth resources specify the size of the margins between the edge of the RowColumn widget and the children along the edge.

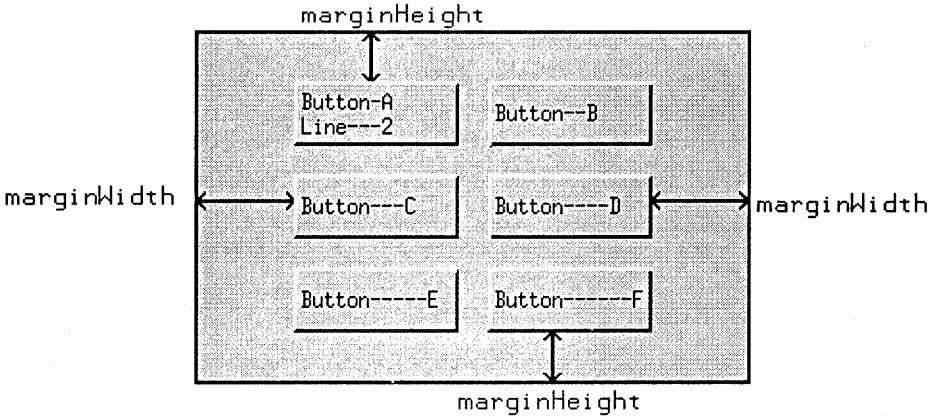


Figure 7-12. XmNmarginHeight and XmNmarginWidth

7.3 Text Widget

The Text widget can be used as a single line or a multiline text editor. You can interact with the Text widget programmatically or by user action. You can use the Text widget for single-line entry, form entry, or full-window editing.

7.3.1 Text Functions

There are a number of functions associated with the Text widget that are available to perform a variety of tasks. These functions are listed below, and each function has its own man page in the *HP OSF/Motif Programmer's Reference Manual.*

TABLE 7-2. Text Widget Functions

Function	Description
XtCreateWidget	Basic widget create function.
XmCreateText	Specific create function for XmText.
XmCreateScrolledText	Create function for ScrolledText.
XmTextClearSelection	Clears the primary selection.
XmTextGetSelection	Retrieves the value of the primary selection.
XmTextSetSelection	Sets the primary selection of text in the widget.
XmTextGetString	Accesses the string value of the Text widget.
XmTextSetString	Sets the string value of the Text widget.
XmTextGetMaxLength	Finds the maximum allowable length of the text string.
XmTextSetMaxLength	Sets the maximum allowable length of the text string.
XmTextGetEditable	Finds the value of the edit permission state.
XmTextSetEditable	Sets the value of the edit permission state.
XmTextReplace	Replaces part of a string.

The use of some of these functions will be described in the sections that follow. Actual code segments accompanied by illustrations show the results of programming actions.

7.3.2 Using the Text Widget In a Program

Figure 7-13 shows an example of a Text widget.

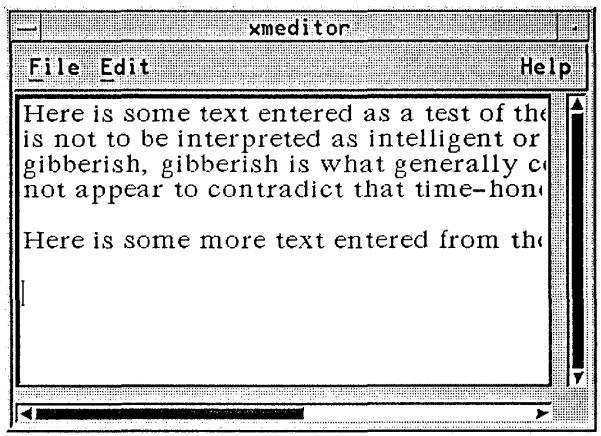


Figure 7-13. Text Widget

You have seen other examples in earlier chapters of this manual. The FileSelectionBox "Selection" window is a Text widget, as is the same window in the SelectionBox widget.

The program that generated the window shown in figure 7-13 is called `xmeditor` and it can be found in the directory `/usr/contrib/Xm`. This program demonstrates how you can use the Text widget in concert with other widgets. You can create a new file, edit an old file, cut and paste text, and so on.

The File Menu

If you move the pointer so that it covers the "File" button in the MenuBar and then click mouse button 1, a menu will appear as shown in figure 7-14.

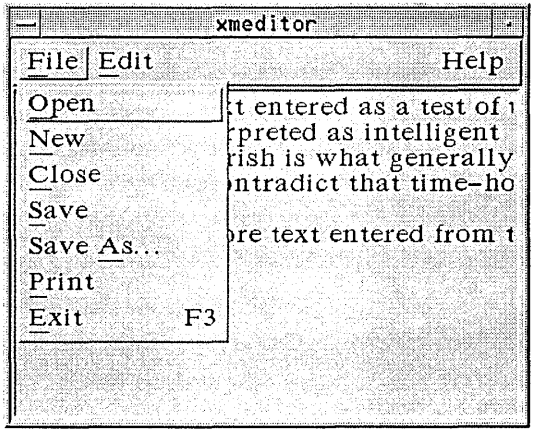


Figure 7-14. Text Demonstration File Menu

If you select "Open," a FileSelectionBox will appear. You can select a file from this as described in chapter 5. The other menu choices are self-explanatory.

The Edit Menu

The "Edit" menu allows you to use some of the "cut and paste" features of the OSF/Motif system within the Text widget. Move the pointer to "Edit" and click mouse button 1 to see the "Edit" menu.

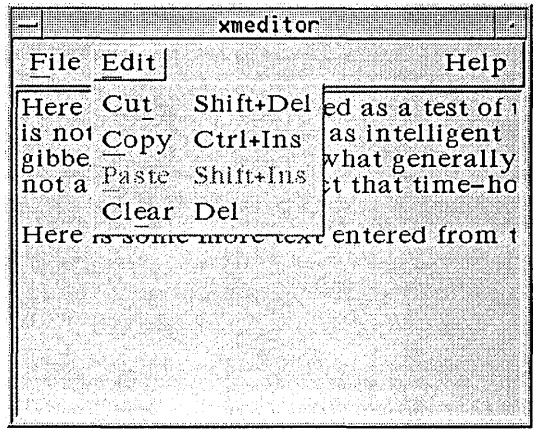


Figure 7-15. Text Demonstration Edit Menu

Accelerators and Mnemonics

Both the "Edit" and "File" menus contain **mnemonics**. Mnemonics are indicated by underlined characters in the menus. You can use them instead of the mouse button to make the selection you want. For example, to choose "File" using its mnemonic, move the pointer into the `xmeditor` window and press the Meta key and [f]. This will cause the Pulldown menu to appear. You can traverse the Pulldown menu by using the [↑] and [↓] keys, or use the mouse to move the pointer. You can select from the Pulldown menu by using its mnemonics as well. All you need to do is press the key that corresponds to the underlined letter in the menu. Don't worry about upper or lower case as these are not case-sensitive. For example, suppose you want to open a file. All you need to do is press [o]. You only need to use the Meta key when accessing a Pulldown menu from the MenuBar.

The "Edit" menu also has "accelerators" in addition to the mnemonics. Accelerators are keys or combinations of keys that perform a specific function within an application. For example, "Cut" has a mnemonic of "t" and an accelerator of [Shift] [Del]. Pressing either [t] or [Shift] [Del] execute the "Cut" function of the menu. Note that accelerators are not part of the menu and will execute regardless of whether or not the menu is visible.

Primary and Secondary Selections

The Text widget has two types of selections, primary and secondary. The primary selection is identified by highlighting the text in inverse video and the secondary selection is identified by underlining the text. You can cut out text that is the primary selection and paste it in at some other point in the Text window. You can either copy a secondary selection or copy and delete it. Both of these selections will be described in more detail in later sections of this chapter.

Cutting Text

To remove (or "cut") text, you move the insert cursor to the starting point of the text you wish to remove. You can then perform any of the following actions to select the text (primary selection) to be removed:

NOTE

To position the insert cursor, move the pointer into the text window and to the desired point on a text line, then click mouse button 1.

- If you want just a single word removed, position the insert cursor before the first letter of the word and then "double click" mouse button 1. The word is highlighted.
- If you want the entire line of text to be removed, "triple click" mouse button 1. The entire line of text is highlighted. Note that the entire line is highlighted regardless of the initial position of the insert cursor on the line.
- If you want all of the text to be removed, "quadruple click" mouse button 1. All of the text is highlighted. Note that the all of the text is highlighted regardless of the initial position of the insert cursor on the line.
- If you want more than one line (or more than one word) of text to be removed, you can drag the insert cursor to select the desired text. Begin by moving the pointer to the starting point in the text window and pressing and holding mouse button 1. This positions the insert cursor at that point. Now drag the insert cursor along the line (or lines) of text that you want to remove. When you reach the end of the text that you want removed, release mouse button 1 and that text is highlighted. See figure 7-16. Another way to do this is by pressing and holding Shift ← or Shift → until the text you want to delete is highlighted.
- If you want to add text to the current primary selection, move the pointer to the position in the text that corresponds to the last character you want to add, and press Shift mouse button 1. You can add text in front of or after the current primary selection.

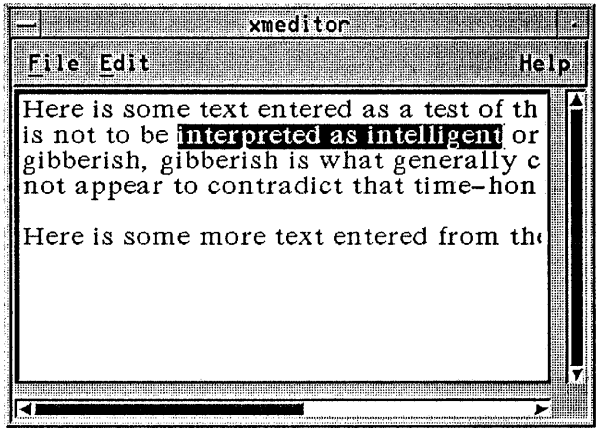


Figure 7-16. Selecting Text for Removal

Now that you have selected the text to be removed, move the pointer to the "Edit" button in the MenuBar and press mouse button 1. The menu shown in figure 7-15 appears. Move the pointer to "Cut" and release mouse button 1. The text you selected disappears.

The source code to accomplish this is taken from `xmeditor.c`. First, in the procedure MenuCB, which is the callback from PushButtons in the Pulldown menu, the case called "MENU_CUT" is executed.

```
case MENU_CUT:
   {
  /* needed to get the event time */
  XmAnyCallbackStruct * cb =  (XmAnyCallbackStruct *) call_data;

  /* call routine to copy selection to clipboard */
  CopyFileToClipboard(cb->event->xbutton.time);

  /* call routine to delete primary selection */
  DeletePrimarySelection();
  }
  break;
```

The procedure `CopyFileToClipboard` copies the text you selected to the clipboard for safekeeping, then that text is deleted by the procedure `DeletePrimarySelection`. You can "paste" the text that was cut by using the method in the procedure `PasteFromClipboard`, which is discussed later in this chapter. Note that if you select "Clear" from the menu instead of "Cut," the text is not copied to the clipboard before being deleted. This means that using "Clear" does not allow a subsequent paste of the

deleted text.

The procedure CopyFileToClipboard uses several of the Cut and Paste functions described in chapter 8. Keep in mind that copying to the clipboard is just a temporary measure. You can then subsequently use other Cut and Paste functions to retrieve the data.

```
/*-------------------------------------------------------------
**  CopyFileToClipboard
**  Copy the present file to the clipboard.
*/
void CopyFileToClipboard(time)
Time time;
{
   char *selected_string = XmTextGetSelection (text);
   unsigned long item_id = 0;      /* clipboard item id */
   int data_id = 0;         /* clipboard data id */
   int status = 0;         /* clipboard status  */
   XmString clip_label;

   /* using the clipboard facilities, copy the selected text */
   /* to the clipboard */
   if (selected_string != NULL) {
  clip_label = XmStringCreateLtoR ("XM_EDITOR", charset);
  /* start copy to clipboard, and continue till
   a successful start copy is made */
  status = 0;
  while (status != ClipboardSuccess)
    status = XmClipboardStartCopy (XtDisplay(text),
        XtWindow(text), clip_label, time, text, NULL, &item_id);

   /* move the data to the clipboard, and
   continue till a successful copy is made */
  status = 0;
  while (status != ClipboardSuccess)
    status = XmClipboardCopy (XtDisplay(text), XtWindow(text),
      item_id, "STRING", selected_string,
     (long)strlen(selected_string)+1, 0, &data_id);

  /* end the copy to the clipboard and continue till
   a successful end copy is made */
  status = 0;
```

```
  while (status != ClipboardSuccess)
    status = XmClipboardEndCopy (XtDisplay(text), XtWindow(text),
       item_id);

  /* allow pasting when an item is successfully copied */
  /* to the clipboard */
  XtSetSensitive(paste_button, True);

    }
}
```

The function XmTextGetSelection is used to set the variable selected_string to point to the *primary selection* in the Text widget. The primary selection in this case is the three words that are highlighted in inverse video shown in figure 7-16. Note that the primary selection always appears in inverse video.

In the code segment shown above the actual copy-to-clipboard process involves three steps:

1. Prepare the clipboard for copying by executing XmClipboardStartCopy. This procedure must return a success before copying can occur. The variable clip_label is set to the string XM_EDITOR to identify the data item for possible use in a clipboard viewer. The variable item_id specifies is set to an arbitrary identification number assigned to this data item. This will be used in the calls to XmClipboardCopy and XmClipboardEndCopy.
2. Copy the data by using XmClipboardCopy. The copying done by this function is *not* actually to the clipboard but to a buffer. Copying to the clipboard occurs when the XmClipboardEndCopy function is executed.
3. The XmClipboardEndCopy function locks the clipboard to prevent access by other applications while performing the copy, copies the data to the clipboard, and then unlocks the clipboard.

Pasting Text

To paste the text you just removed, move the insert cursor to the position where you want the text to be inserted. Move the pointer to the "Edit" button in the MenuBar and press and hold mouse button 1. Move the pointer to "Paste" and release mouse button 1. The text you deleted is restored (pasted) into the new position. See figure 7-17.

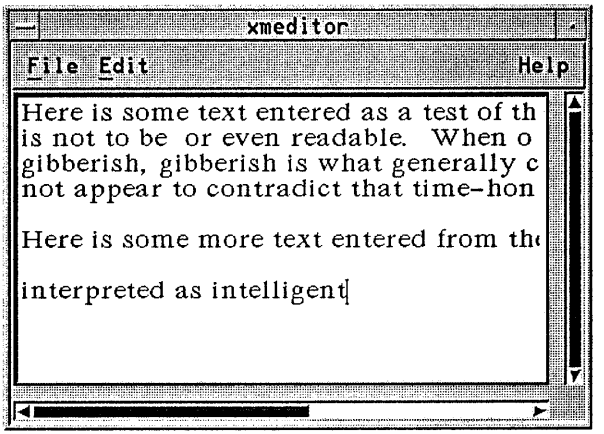


Figure 7-17. Pasting Text

The code to accomplish this begins with that part of the procedure MenuCB dealing with a paste operation.

```
case MENU_PASTE:
/* call the routine that pastes the
text at the cursor position */
PasteItemFromClipboard();
break;
```

When you select "Paste" from the menu, PasteItemFromClipboard is called.

```
/*-------------------------------------------------------------
** PasteItemFromClipboard
**  paste item from the clipboard to the current
**  cursor location
*/
void PasteItemFromClipboard()
{
   /* retrieve the current data from the clipboard
   and paste it at the current cursor position */

   int status = 0;              /* clipboard status        */
   char *buffer;                /* temporary text buffer   */
   int length;                  /* length of buffer        */
   int outlength = 0;           /* length of bytes copied  */
   int private_id = 0;          /* id of item on clipboard */
   XmTextPosition cursorPos;    /* text cursor position    */
   register int ac;             /* arg count               */
```

```
    Arg al[10];                 /* arg list                 */

    /* find the length of the paste item and
        continue till the length is found */
    while (status != ClipboardSuccess) {
    status = XmClipboardInquireLength(XtDisplay(text), XtWindow(text
           "STRING", &length);
    if (status == ClipboardNoData) {
    length = 0;
    break;
    }
    }

    if (length == 0) {
    fprintf(stderr, "Warning: paste failed, no items to paste.\n");
    return;
    }

    /* malloc to necessary space */
    buffer = XtMalloc(length);

    status = XmClipboardRetrieve (XtDisplay(text), XtWindow(text),
         "STRING", buffer, length, &outlength, &private_id);

    /* Dialogs need to be added to indicate errors in pasting */
    if (status != ClipboardSuccess) {
    fprintf(stderr, "Warning: paste failed, status = %d\n", status);
    return;
    }

    /* get cursor position for pasting */
    XtSetArg(al[0], XmNcursorPosition, &cursorPos);
    XtGetValues(text, al, 1);

    /* add new text */
    XmTextReplace(text, cursorPos, cursorPos, buffer);
}
```

The length of the data stored under the format name "STRING" is found by using the procedure `XmClipboardInquireLength`. If this procedure is successful, then space is allocated for that length by `XtMalloc` and `XmClipboardRetrieve` is called to retrieve the text from the clipboard. Then the current cursor position in the Text widget is found through `XtGetValues` and the text is actually pasted by the function `XmTextReplace`.

Cutting and Pasting Using Secondary Selection

The cut and paste procedure discussed above using the primary selection requires two steps. You can accomplish the same thing in one step by using the secondary selection feature of the Text widget. Position the insert cursor at the starting point that you wish the text be copied to. Move the Pointer to the first character of the text you want to delete and copy, press and hold CTRL mouse button 3, and drag the pointer until you reach the end of the text you want to delete and copy. Release the CTRL key and mouse button 3 and the text is deleted from its original position and copied to its new position.

Copying Text

To copy text, you move the insert cursor to the starting point of the text you want to copy and select the text in the same manner as for cutting text. When you have the text selected, move the pointer to the "Edit" button in the MenuBar and press and hold mouse button 1. When the menu appears, drag the pointer to "Copy" and release mouse button 1. You won't see any change in the `xmeditor` window, but the text has been copied to the clipboard. In this case, the case called `MENU_COPY` is executed.

```
case MENU_COPY:
    {
    /* needed to get the event time */
    XmAnyCallbackStruct * cb = (XmAnyCallbackStruct *) call_data;

    /* call routine to copy selection to clipboard */
    CopyFileToClipboard(cb->event->xbutton.time);

    }
    break;
```

The only difference between this case and `MENU_CUT` is that the delete procedure `DeletePrimarySelection` is not called. The data is copied to the clipboard as before and can be pasted anywhere in the text.

Copying Text Using Secondary Selection

You can also copy text using the secondary selection method. This is a feature of the Text widget rather than the `xmeditor` program. For example, perform the following steps to copy the words "Here is some more text" from the last line of the visible text to a new line.

1. Move the insert cursor at the point you want the copied text to appear. In this case, move the pointer to the start of the second line after the last visible line in the Text window and click mouse button 1. The insert cursor should then appear.

2. Move the pointer (not the insert cursor) to the first position in the last visible line of text. Press and hold mouse button 3 and drag the pointer along the line of text. The text is underlined as you drag the pointer. Be careful not to move the pointer up or down as that will cause other areas of the text to be selected and underlined. You

can continue to select as long as you hold mouse button 3 down. When you are satisfied with the text you have selected, release mouse button 3 and the copy is done automatically.

Figures 7-18 and 7-19 show the secondary selection and the results of the copy action. Note that the insert cursor is positioned at the end of the newly copied text.

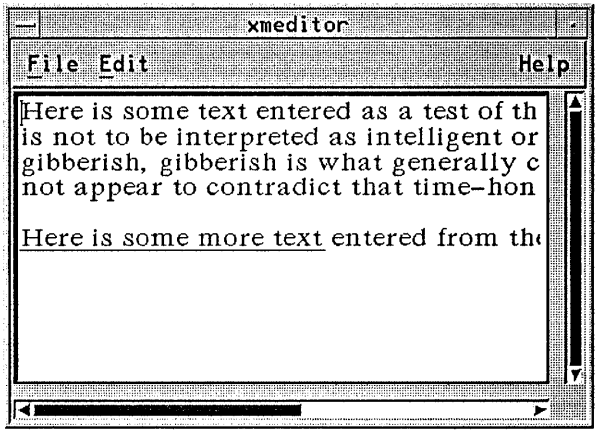


Figure 7-18. Secondary Selection

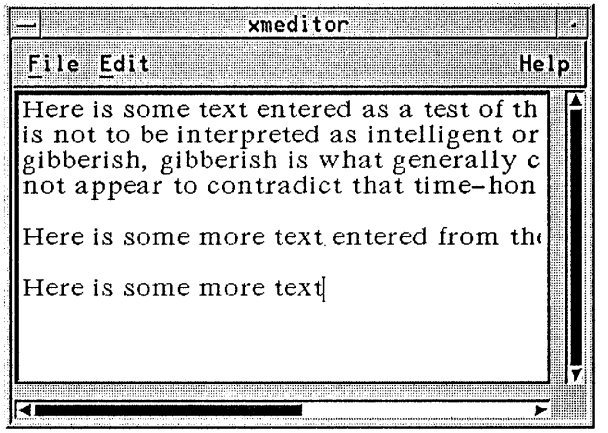


Figure 7-19. Secondary Selection Copy Result

Additional Functionality 8

The OSF/Motif library provides additional functionality that you can use to perform certain tasks. The following list provides an overview of this functionality. Subsequent sections of this chapter describe each topic in detail.

- Compound Strings. A compound string is designed to allow any message or text to be displayed without having to resort to "hard-coding" certain attributes that are language dependent. The three main attributes involved are direction, character set, and text. The OSF/Motif library provides a number of functions that allow you to create and manipulate compound strings.
- Cut and Paste functions. The OSF/Motif library has a "clipboard" that is used to hold data that is being transferred between applications. It also provides a set of "Cut and Paste" functions that allow you to modify the type and value of the data.
- Dynamic Resource Defaulting. This feature incorporates a processing function into a widget's resource definitions. This function is used to calculate a default resource value when the widget is created, thus overriding any static default value.
- Keyboard "Grabbing." Grabbing refers to an action in which an application or window isolates an input device (the keyboard in this case) from other applications or windows, thus preventing the other applications or windows from using the device.
- Localization. You can use localized defaults files in the OSF/Motif system by specifying the location of these files within certain environment variables. `XtInitialize` determines the proper path to the localized defaults variables.
- Pixmap Naming and Caching. The OSF/Motif library provides functions that allow you to associate any image with a unique name. With this association, the functions can generate pixmaps through references to a .Xdefaults file and through an argument list for all widgets with pixmap resources.
- Resolution Independence. Resolution independence is a feature that allows your application to create and display images that are the same physical size regardless of the resolution of the display.
- Interacting with the OSF/Motif Window Manager. There are a number of functions you can use when interacting with the OSF/Motif Window Manager. A sample program is available that demonstrates how to use some of these functions.

- OSF/Motif Version and Window Manager Presence. There are functions that provide information on the current version of the OSF/Motif system and whether or not the OSF/Motif Window manager is running.

8.1 Compound Strings

A compound string is designed to allow any message or text to be displayed without having to resort to "hard-coding" certain attributes that are language dependent. The three main attributes involved are character set, direction, and the text of the message. For example, suppose you have a message to display in English and Arabic. This can easily be done with a compound string because you specify not only the text of the message, but character-set and direction as well.

- Character set. The mapping between the string of bytes that make up the text and the font in which the text will be displayed.
- Direction. The relationship between the logical order (keystroke entry) and the display order of the characters in a string. In English, the display order is left to right; as characters are typed, they are displayed from left to right. In Arabic, direction is right to left; as characters are typed, they are displayed from right to left.
- Text. This is simply the text of the message or string you wish to display.

In addition, you specify a font list from which to select the fonts used to display the message. The font list contains character set references that are matched with an X font. To display a compound string, the widget uses the character set specified in the compound string and searches the font list for a matching character set. Note that the font list is a widget resource.

The OSF/Motif library provides a set of compound string functions that enable the creation and manipulation of compound strings and font lists. This chapter discusses the compound string functions you can use to

- Create compound strings.
- Compare and manipulate compound strings.
- Create a font list and font list entries.

8.1.1 Components of a Compound String

A compound string is a stream of data that is made up of "tag-length-value" (TLV) segments. Each TLV segment represents an attribute of the compound string. Within a TLV segment, the fields are

- Tag. A one-byte field that identifies the type of Value that follows.
- Length. A two-byte field that specifies the length of the data in the Value field.
- Value. The value of the segment. The length of this field is the number of bytes specified in the Length field.

For example, for a TLV segment that identifies the character set to be used, the Tag field identifies the segment as a character-set segment, the Length field sets the length of the segment, and the Value field contains the character-set identifier.

A compound string *always* begins with a Tag field set to `0x7f` and a Length field set to the length of the segments that follow. Subsequent TLV segments then define the remainder of the compound string. The Value field of each of the TLV segments contains the information about the attributes of the compound string. These attributes are described below.

- A character set identifier. This is a sequence of bytes that identifies the desired character set. This information is used by the OSF/Motif library to match a font with a compound string segment. All text between two character set identifiers are interpreted to be in the first set. It is an error for a text component to precede the first character set identifier.

  The character set identifier has **persistence**, that is, any specified character set is used for all subsequent text segments until a new character set is encountered.

  There are times when an application must create a string without knowing what character sets will be available at the time the string is to be displayed. The OSF/Motif library provides a special character set identifier that will match any available font. This universal character set is specified by the `XmSTRING_DEFAULT_CHARSET identifier`. If this identifier is used as the character set when a compound string is created, it will match the first font in the fontlist used to display the string, regardless of the character set associated with that particular font. By using the universal character set, an application can construct its strings so that they will be displayed in any font desired by the user, no matter what character set is associated with that font.

  The universal character set can also be associated with a font in a font list. When used with a font, that font will match any string, no matter what the character set of the string. An application can thus construct a multiple-font fontlist and specify a default font to be used when no other font is matched.

- A Direction. This can have three values: left-to-right, right-to-left, and revert. Like the character set identifier, it has persistence. The default direction is left-to-right; that is, text components preceding the first direction component will be assigned a direction of left-to-right.
- Text. This is the actual character data. There are no semantics for any bytes. Specifically this means that characters like \n do not have any meaning. As a convenience there is a function, "XmStringCreateLtoR," which does impose this single semantic.
- A separator. This is a tag with no value. It is simply a marker that allows an array of compound string segments to be presented as a single entity.

The following is a set of useful definitions for compound strings:

```
typedef unsigned char   XmStringDirection  /* an enumerated type */
```

The set of possible values for this type are

```
XmSTRING_DIRECTION_L_TO_R
XmSTRING_DIRECTION_R_TO_L

typedef char * XmStringCharSet /* octet chars, null terminated */

typedef char * XmString  /* opaque to users */

typedef unsigned char XmStringComponentType /* component tag types
```

The set of currently possible values for this type are defined as follows:

```
#define XmSTRING_COMPONENT_UNKNOWN 0
#define XmSTRING_COMPONENT_CHARSET 1
#define XmSTRING_COMPONENT_TEXT  2
#define XmSTRING_COMPONENT_DIRECTION 3
#define XmSTRING_COMPONENT_SEPARATOR 4
#define XmSTRING_COMPONENT_END  126  /* no more components */
#define XmSTRING_COMPONENT_RESERVED 127   /* 0-127 reserved for OSI
#define XmSTRING_DEFAULT_CHARSET (-1) * The universal character set
```

8.1.2 Compound String Functions

There are a number of functions associated with compound strings that you can use. The tables on the following pages list these functions along with a brief description of what each can do. Subsequent sections describe the functions in more detail. There is a short sample program at the end of the list of functions that shows how to use some of them.

TABLE 8-1. Compound String Functions

Function Name	Description
XmFontListAdd	Add an entry to an existing fontlist.
XmFontListCreate	Create a new fontlist.
XmFontListFree	Recover memory used by a fontlist.
XmStringBaseline	Returns the number of pixels between the top of the character box and the baseline of the first line of text in the specified compound string.
XmStringByteCompare	Returns True or False as the result of a byte-by-byte comparison of two specified compound strings.
XmStringCompare	Returns True if two compound strings have the same text components, directions, and separators.
XmStringConcat	Appends one compound string to another.
XmStringCopy	Returns a copy of the specified compound string.
XmStringCreate	Creates a compound string.
XmStringCreateLtoR	Creates a compound string in a left-to-right direction.
XmStringDirectionCreate	Creates a compound string with just one component, the specified direction.
XmStringDraw	Draws a compound string in an X window.
XmStringDrawImage	This function is identical to XmStringDraw except that it also paints both the foreground and background bits of each character.
XmStringDrawUnderline	Identical to XmStringDraw except that if the specified substring is matched in the main string, then the substring is underlined.
XmStringEmpty	Returns True if all text segments are empty.
XmStringExtent	Determines the height and width (in pixels) of the smallest rectangle that will enclose the specified compound string.
XmStringFree	Frees the memory used by a compound string.
XmStringFreeContext	Frees a previously established context.
XmStringGetLtoR	Returns True if a segment can be found in the input compound string that matches the specified character set identifier.
XmStringGetNextComponent	Returns the next component.

TABLE 8-1. Compound String Functions (Continued)

Function Name	Description
XmStringGetNextSegment	Returns the bytes in the next segment.
XmStringHeight	Returns the height (in pixels) of the sum of all the line heights of the specified compound string.
XmStringInitContext	Specifies a context used to read the contents of a compound string segment by segment.
XmStringLength	Returns the length of the specified compound string.
XmStringLineCount	Returns the number of lines of text in the specified compound string.
XmStringNConcat	Appends a specified number of bytes from one compound string to another.
XmStringNCopy	Returns a copy of a specified portion of a compound string.
XmStringPeekNextComponent	Returns the type of the next component.
XmStringSegmentCreate	Creates a compound string segment.
XmStringSeparatorCreate	Creates a compound string with only a separator.
XmStringWidth	Returns the width (in pixels) of the longest sequence of text components in the specified compound string.

XmFontListAdd

This function adds an entry to an existing font list.

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

oldlist Specifies a pointer to the font list to which an entry will be added.

font Specifies a pointer to the font structure to be added to the list.

charset Specifies the character set identifier for the font being added to the list. This can be XmSTRING_DEFAULT_CHARSET.

XmFontListAdd creates a new font list consisting of the contents of *oldlist* and the new

font list element being added. Note that this function de-allocates the *oldlist* after extracting the required information; *oldlist* should not be referenced thereafter. The code segment below shows you how this function is used.

```
XmFontList fontlist1, fontlist2;

XmFontStruct *font1, *font2;

fontlist1 = XmFontListCreate(font1, "chset1");
fontlist2 = XmFontListAdd(fontlist1, font2, "chset2");
```

The variables "chset1" and "chset2" are set in an apps-default file. This is shown in the sample program at the end of this section.

XmFontListCreate

This function creates a new font list. See the code segment above for an example of how to use this function.

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

font Specifies a pointer to a font structure for which the new font list is generated.

charset Specifies the character set identifier for the font. This can be XmSTRING_DEFAULT_CHARSET.

The function XmFontListCreate creates a new font list with a single element specified by the provided font and character set. It also allocates the space for the font list.

XmFontListFree

This function frees the memory used by a font list.

```
void XmFontListFree (list)
  XmFontList list;
```

list Specifies the font list to be freed.

XmStringBaseline

This function returns the number of pixels between the top of the character box and the baseline of the first line of text in the specified compound string.

```
Dimension XmStringBaseline (fontlist, string)
 XmFontList fontlist;
 XmString string;
```

XmStringByteCompare

This function determines whether or not two compound strings are identical.

```
Boolean XmStringByteCompare (s1, s2)
  XmString s1, s2;
```

This function returns True if the comparison shows the two specified compound strings to be identical and False if they are not.

It is important to note that when a compound string is placed in a widget it is converted into an internal form to allow faster processing. Part of the conversion process strips out unnecessary or redundant information. The result is that if an application subsequently executes a call to `XtGetValues` to retrieve a compound string from a widget (specifically XmLabel and all of its subclasses), no guarantee can be made that the compound string returned will be the same byte-for-byte as the original string in the widget.

XmStringCompare

This function determines whether or not two compound strings are "semantically" (but not necessarily byte-for-byte) equivalent.

```
Boolean XmStringCompare (s1, s2)
  XmString s1, s2;
```

This function returns True if the two compound strings are semantically equivalent and False otherwise. "Semantically equivalent" means that the strings have the same text components, directions, and separators. If character sets are specified, they must be equal as well.

XmStringConcat

This function appends a copy of one compound string to another compound string.

```
XmString XmStringConcat (s1, s2)
  XmString s1, s2;
```

`XmStringConcat` appends *s2* to the end of *s1* and returns the resulting compound string. The original strings are preserved. The space for the resulting compound string is

allocated within the function. After using this function, you should free this space by calling XtFree.

XmStringCopy

This function creates a copy of a compound string.

```
XmString XmStringCopy (s1)
  XmString s1;
```

This function returns a copy of *s1*. The space for the resulting compound string is allocated within the function. The application is responsible for managing the the allocated space. The memory can be recovered by calling XtFree.

XmStringCreate

This function creates a compound string.

```
XmString XmStringCreate (text, charset)
  char   *text;
  XmStringCharSet charset;
```

text Specifies a pointer to a null terminated string.

charset Specifies the character set identifier to be associated with the given text. This can be XmSTRING_DEFAULT_CHARSET.

This function creates a compound string with two components: a character set and text.

XmStringCreateLtoR

This function creates a compound string with a default direction of left-to-right.

```
XmString XmStringCreateLtoR (text, charset)
  char   *text;
  XmStringCharSet charset;
```

text Specifies a pointer to a null terminated string.

charset Specifies the character set identifier to be associated with the given text. This can be XmSTRING_DEFAULT_CHARSET.

This function is similar to XmStringCreate except that it scans the text for newline characters in the text. When one is found, the text up to that point is put into a segment followed by a separator component. No final separator component is appended to the end of the compound string. The direction is defaulted to left-to-right. Finally, note that this function assumes that the encoding is single octet rather than double or quadruple octet

per character of text.

XmStringDirectionCreate

This function creates a compound string with a single component, a direction with the specified value.

```
XmString XmStringDirectionCreate (direction)
  XmStringDirection direction;
```

direction Specifies the value of the directional component.

XmStringDraw

This function is used to draw a compound string in an X window.

```
void XmStringDraw (d, w, fontlist, string, gc, x, y, width, alignment, layout_direction, clip);
  Display *d;
  Window w;
  XmFontList fontlist;
  XmString string;
  GC gc;
  Position x,y;
  Dimension width;
  Byte alignment;
  Byte layout_direction;
  XRectangle *clip;
```

The *x* and *y* parameters identify the top left coordinate of the rectangle that contains the displayed compound string. The *layout_direction* parameter controls the direction in which the segments of the compound string are laid out. It also is used to determine the meaning of the alignment parameter. The *clip* parameter allows the application to restrict the area into which the compound string will be drawn. If it is NULL, no clipping is done.

XmStringDrawImage

This function is identical to `XmStringDraw` except that it paints both the foreground and background bits of each character (equivalent to `XDrawImageString`).

```
void XmStringDrawImage (d, w, fontlist, string, gc, x, y, width,
  alignment, layout_direction, clip);
   Display *d;
   Window w;
   XmFontList fontlist;
   XmString string;
   GC gc;
   Position x,y;
```

```
Dimension width;
Byte alignment;
Byte layout_direction;
XRectangle *clip;
```

XmStringDrawUnderline

This function is equivalent to XmStringDraw with the addition that if the substring identified by *underline* can be matched in *string*, then the substring will be underlined. Once a match has occurred no further matches or underlining will be done.

```
void XmStringDrawUnderline (disp, d, fontlist, string, gc, x, y, width,
 alignment, layout_direction, clip, underline);
  Display *disp;
  Drawable d;
  XmFontList fontlist;
  XmString string;
  GC gc;
  Position x,y;
  Dimension width;
  Byte alignment;
  Byte layout_direction;
  XRectangle *clip;
  XmString underline;
```

XmStringEmpty

This function determines whether or not a compound string is empty.

```
Boolean XmStringEmpty (s1)
  XmString s1;
```

This function returns True or False depending on whether or not any non-zero text components exist in the provided compound string. It returns True if all text segments are empty or if the specified string parameter is NULL, and False otherwise.

XmStringExtent

This function determines the width and height (in pixels) of the smallest rectangle that will enclose the specified compound string.

```
void XmStringExtent (fontlist, string, width, height)
  XmFontList fontlist;
  XmString string;
  Dimension *width, *height;
```

XmStringFree

This function frees the memory used by a compound string.

```
void XmStringFree (string)
  XmString string;
```

string Specifies the compound string to be freed.

XmStringFreeContext

This function instructs the intrinsics that the context is no longer needed and will not be used without reinitialization.

```
void XmStringFreeContext (context)
  XmStringContext context;
```

XmStringGetLtoR

This function returns True if a segment can be found in the input compound string that matches the given character set identifier.

```
Boolean XmStringGetLtoR (string, charset, text)
  XmString string;
  XmStringCharSet charset;
  char **text;
```

On return, *text* will have a null-terminated octet sequence containing the matched segment.

XmStringGetNextComponent

This function returns the type and value of the next component in the compound string identified by the specified context.

```
XmStringComponentType XmStringGetNextComponent (context,
  text, charset, direction, unknown_tag, unknown_length, unknown_value)
  XmStringContext *context;
  char **text;
  XmStringCharSet *charset;
  XmStringDirection *direction;
  XmStringComponentType *unknown_tag;
  short *unknown_length
  char **unknown_value;
```

This is a low-level component fetch function. Components are returned one at a time. Only some output parameters will be valid on return, and this can be determined by examining the return status. In the case of text, charset, or direction components, only one output parameter is valid. If the return status indicates that an unknown component was encountered, the tag, length and value are returned. This function will allocate the space necessary to hold returned values; freeing this space is the caller's responsibility.

XmStringGetNextSegment

This function returns the bytes in the next segment of the specified compound string.

```
Boolean XmStringGetNextSegment (context, text, charset, direction, separator)
  XmStringContext *context;
  char **text;
  XmStringCharSet *charset;
  XmStringDirection *direction;
  Boolean *separator;
```

The text, character set, and direction of the fetched segment are returned. The separator parameter indicates whether or not the next component of the compound string is a separator. True or False is returned to indicate whether or not a valid segment was successfully parsed.

XmStringHeight

This function returns the height in pixels of the sum of all the line heights of the given compound string.

```
Dimension XmStringHeight (fontlist, string)
  XmFontList fontlist;
  XmString string;
```

Separator components delimit lines.

XmStringInitContext

This function establishes the context for a subsequent segment-by-segment read of the specified compound string.

```
Boolean XmStringInitContext (context, string)
  XmStringContext *context;
  XmString string;
```

In order to allow applications to read the contents of a compound string segment-by-segment some "context" needs to be maintained. This function establishes the context for such a read. A True or False value is returned to indicate whether or not the input string was able to be parsed.

XmStringLength

This function obtains the length of a compound string.

```
int XmStringLength (s1)
  XmString s1;
```

This function returns the number of bytes in s1, including the string header (0x7f) and all tags, direction indicators, and separators. Zero is returned if the compound string has an

invalid structure.

XmStringLineCount

This function returns the number of lines of text in the specified compound string.

```
int XmStringLineCount (string)
  XmString string;
```

XmStringNConcat

This function appends a specified number of bytes from one compound string to another.

```
XmString XmStringNConcat (s1, s2, num_bytes)
  XmString s1, s2;
  int num_bytes;
```

XmStringNConcat appends *num_bytes* bytes from *s2* to *s1*, including tags, directional indicators, and separators. It then returns the resulting compound string. If *num_bytes* is less than the length of *s2*, the resulting string will not be a valid compound string the original strings are preserved. The space for the resulting compound string is allocated within the function. The application is responsible for managing the allocated space. The memory can be recovered by calling XtFree.

XmStringNCopy

This function copies a specified portion of a given compound string.

```
XmStringNCopy (s1, num_bytes)
 XmString s1;
 int num_bytes;
```

This function creates a copy of *s1* which contains *num_bytes* bytes from *s1*, including tags, directional indicators, and separators. It then returns the resulting copy. If *num_bytes* is less than the length of s1, the resulting string will not be a valid compound string and the original string is preserved. For this reason, you should normally use XmStringCopy. The space for the resulting compound string is allocated within the function. The application is responsible for managing the allocated space. The memory can be recovered by calling XtFree.

XmStringPeekNextComponent

This function examines the next component that would be fetched by XmStringGetNextComponent and returns the component type.

```
XmStringComponentType XmStringPeekNextComponent (context)
  XmStringContext *context;
```

XmStringSegmentCreate

This is a high-level function that assembles a compound string consisting of a character set identifier, a direction component, a text component, and, optionally, a separator component. If *separator* is False, then the compound string does not have a separator component at the end. If it is True, the compound string has a separator component immediately following the text component.

```
XmString XmStringSegmentCreate (text, charset, direction, separator)
   char *text;
   XmStringCharSet charset;
   XmStringDirection direction;
   Boolean separator;
```

text Specifies a pointer to a null-terminated string.

charset Specifies the character set identifier to be associated with the text. This can be XmSTRING_DEFAULT_CHARSET.

direction Specifies the direction of the text.

separator Specifies if a separator should be added to the compound string segment being constructed.

XmStringSeparatorCreate

This function creates a compound string with a single component, a separator.

```
XmString XmStringSeparatorCreate (separator)
   Boolean separator;
```

XmStringWidth

This function returns the width in pixels of the longest sequence of text components in the provided compound string. Separator components are used to delimit sequences of text components.

```
Dimension XmStringWidth (fontlist, string)
  XmFontList fontlist;
  XmString string;
```

8.1.3 A Sample Program

The sample program shown below illustrates how you can use the compound string functions in applications. This program creates a PushButton, the label for which is "Hello World." Each word in the label appears in a different font as specified in the apps-defaults file for the program.

Program Listing

```
#include <X11/Xlib.h>
#include <Xt/Intrinsic.h>
#include <Xt/Shell.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>

Widget toplevel,pbutton;
Arg myArgs[10];

void main(argc, argv)
  unsigned int argc;
  char **argv;
{
  int i;
  XmString s1,s2,string,ButtonText;
  char *word1="Hello ",*word2="World";
  char *appname = "lab0";
  char *appclass = "LAB0";

  toplevel = XtInitialize(appname, appclass, NULL, NULL, &argc, ar
  s1=XmStringSegmentCreate(word1, "chset1", XmSTRING_DIRECTION_L_T
                           False);
  s2=XmStringSegmentCreate(word2, "chset2", XmSTRING_DIRECTION_L_T
                           True);
  string=XmStringConcat(s1,s2);

  i=0;
  XtSetArg(myArgs[i], XmNlabelString, string); i++;
  pbutton = XmCreatePushButton(toplevel,"x01",myArgs,i);
  XtManageChild(pbutton);
  XtRealizeWidget(toplevel);
  XtMainLoop();
```

Defaults File

```
!
!   Apps default file LAB0 for compound string function sample program
!
!
*foreground: Yellow
*background: SlateBlue
*FontList: hp8.8x16b=chset1, hp8.8x16=chset2
```

Note that fontlists can be specified in a defaults file by setting the fontlist resource to a string of the form `fontname = character set`. The widget will build a fontlist consisting of the specified font and character set. If the character set is omitted, the character set will default to XmSTRING_DEFAULT_CHARSET.

Strings can be specified by setting the string resource to the desired text. For example, `oklabel.labelString: OK` sets a label's text to "OK." Specifying a string in a defaults file is the same as creating the string using XmStringCreateLtoR with the default character set XmSTRING_DEFAULT_CHARSET.

8.2 Cut and Paste Functions

The OSF/Motif clipboard is used to hold data that is to be transferred between applications. The OSF/Motif library provides the functions necessary to modify the type and value of the data that is to be transferred via the clipboard. These functions are known as "cut and paste" functions. An application can interface to the OSF/Motif clipboard through calls to the cut and paste functions.

A brief description of each function is listed in table 8-2. Detailed information on each function is presented in the sections that follow.

TABLE 8-2. Cut and Paste Functions

Function Name	Description
XmClipboardStartCopy	Set up storage and data structures for clipboard copying.
XmClipboardCopy	Copy a data item to the clipboard.
XmClipboardCopyByName	Copy a data item passed by name.
XmClipboardCancelCopy	Cancel a copy to clipboard.
XmClipboardUndoCopy	Delete the last item placed on the clipboard.
XmClipboardEndCopy	Ends copy to clipboard.
XmClipboardInquireCount	Return the number of data item formats.
XmClipboardInquireFormat	Return a specified format name.
XmClipboardInquireLength	Return the length of the stored data.
XmClipboardInquirePendingItems	Return a list of data id/private id pairs.
XmClipboardStartRetrieve	Start copy from clipboard.
XmClipboardRetrieve	Retrieve a data item from the clipboard.
XmClipboardEndRetrieve	Ends copy from clipboard.
XmClipboardLock	Lock the clipboard.
XmClipboardRegisterFormat	Registers a new format.
XmClipboardUnlock	Unlock the clipboard.
XmClipboardWithdrawFormat	Indicates the application no longer wants to supply a data item.

8.2.1 Clipboard Copy Functions

XmClipboardStartCopy

This function sets up storage and data structures to receive clipboard data.

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

display Specifies a pointer to the Display structure that was returned in a previous call to XOpenDisplay or XtInitialize.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

clip_label Specifies the label to be associated with the data item. This argument is used to identify the data item, for example, in a clipboard viewer. An example of a label is the name of the application that places the data in the clipboard.

timestamp The time of the event that triggered the copy.

widget Specifies the ID of the widget that will receive messages requesting data previously passed by name. This argument must be present in order to pass data by name. Any valid widget ID in your application can be used for this purpose and all the message handling is taken care of by the cut and paste functions.

callback Specifies the address of the callback function that is called when the clipboard needs data that was originally passed by name. This is also the callback to receive the DELETE message for items that were originally passed by name. This argument must be present in order to pass data by name.

item_id Specifies the number assigned to this data item. The application uses this number in calls to `XmClipboardCopy`, `XmClipboardEndCopy`, and `XmClipboardCancelCopy`.

The `XmClipboardStartCopy` function sets up storage and data structures to receive clipboard data. An application calls `XmClipboardStartCopy` during a cut or copy operation. The data item that these structures receive then becomes the next-paste item in the clipboard.

Copying a large piece of data to the clipboard can take time. It is possible that, once copied, no application will ever request that data. The OSF/Motif library provides a mechanism so that an application does not need to actually pass data to the clipboard until the data has been requested by some application. Instead, the application passes format and length information in `XmClipboardCopy` to the clipboard functions, along with a widget ID and a callback function address that is passed in `XmClipboardStartCopy`.. The widget ID is needed for communications between the clipboard functions in the application that owns the data and the clipboard functions in the application that requests the data. Your callback functions are responsible for copying the actual data to the clipboard (via `XmClipboardCopyByName`). The callback function is also called if the data item is removed from the clipboard, and the actual data is therefore no longer needed.

For more information on passing data by name, see XmClipboardCopy, and XmClipboardCopyByName.

The *widget* and *callback* arguments must be present in order to pass data by name. The callback format is as follows:

```
function name(widget, data_id, private, reason)
      Widget *widget;
      int *data_id;
      int *private;
      int *reason;
```

widget	Specifies the ID of the widget passed to XmClipboardStartCopy.
data_id	Specifies the identifying number returned by XmClipboardCopy, which identifies the pass-by-name data.
private	Specifies the private information passed to XmClipboardCopy.
reason	Specifies the reason, which is either XmCRCLIPBOARD_DATA_DELETE or XmCRCLIPBOARD_DATA_REQUEST.

This function can return one of the following status return constants:

ClipboardSuccess	The function is successful.
ClipboardLocked	The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardCopy

This function copies a data item to temporary clipboard storage.

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

display Specifies a pointer to the Display structure that was returned in a previous call to XOpenDisplay or XtInitialize.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to XmClipboardStartCopy.

format_name Specifies the name of the format in which the data item is stored on the clipboard. Format is referred to as "target" in the *Inter-Client Communication Conventions* manual.

buffer Specifies the buffer from which the clipboard copies the data.

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

private_id Specifies the private data that the application wants to store with the data item.

data_id Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This argument is required only for data that is passed by name.

The XmClipboardCopy function copies a data item to temporary clipboard storage. The data item is moved from temporary storage to the clipboard data structure when a call to XmClipboardEndCopy is made. Additional calls to XmClipboardCopy before a call to XmClipboardEndCopy add additional data item formats to the same data item or append data to an existing format.

If the *buffer* argument is NULL, the data is considered passed by name. If data passed by name is later needed by another application, the application that owns the data receives a callback with a request for the data. The application that owns the data must then transfer the data to the clipboard with the XmClipboardCopyByName function. When a data item that was passed by name is deleted from the clipboard, the application that owns the data receives a callback that states that the data is no longer needed.

For information on the callback function, see the callback argument description for XmClipboardStartCopy.

This function can return one of the following status return constants:

ClipboardSuccess The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardCopybyName

This function copies a data item to the clipboard.

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

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.
data_id	Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This number was assigned by `XmClipboardCopy` to the data item.
buffer	Specifies the buffer from which the clipboard copies the data.
length	Specifies the number of bytes in the data item.
private_id	Specifies the private data that the application wants to store with the data item.

The `XmClipboardCopyByName` function copies the actual data for a data item that was previously passed by name to the clipboard. Data is considered to be passed by name when a call to `XmClipboardCopy` is made with the *buffer* parameter is set to NULL. Additional calls to `XmClipboardCopyByName` append new data to the existing data. The clipboard should be locked before making such calls by using `XmClipboardLock` to insure the integrity of the clipboard data.

This function can return one of the following status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardCancelCopy

This function cancels any copy to the clipboard that is in progress and frees any temporary storage in use.

```
void XmClipboardCancelCopy(display, window, item_id)
     Display *display;
     Window window;
     long item_id;
```

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.
item_id	Specifies the number assigned to this data item. This number was returned by a previous call to `XmClipboardStartCopy`.

The `XmClipboardCancelCopy` function cancels any copy-to-clipboard that is in progress and frees any temporary storage in use. When a copy is performed, `XmClipboardStartCopy` allocates temporary storage for the clipboard data. `XmClipboardCopy` places the data in the temporary storage. `XmClipboardEndCopy` copies the data to the clipboard data structure and frees the temporary data storage.

`XmClipboardCancelCopy` also frees up temporary storage. If `XmClipboardCancelCopy` is called, then `XmClipboardEndCopy` does not have to be called. A call to `XmClipboardCancelCopy` is valid only after a call to `XmClipboardStartCopy` and before a call to `XmClipboardEndCopy`.

XmClipboardUndoCopy

This function deletes the last item placed on the clipboard.

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

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

The `XmClipboardUndoCopy` function deletes the last item placed on the clipboard if the item was placed there by an application with the passed *display* and *window* arguments. Any data item deleted from the clipboard by the original call to `XmClipboardCopy` is restored. If the *display* or *window* IDs do not match the last copied item, no action is taken and this function has no effect.

This function can return one of the following status return constants:

`ClipboardSuccess` The function is successful.

`ClipboardLocked` The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardEndCopy

This function has several uses: to lock the clipboard, to place data in the clipboard data structure, and to unlock the clipboard.

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

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window

ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

item_id Specifies the number assigned to this data item. This number was returned by a previous call to `XmClipboardStartCopy`.

The `XmClipboardEndCopy` function locks the clipboard from access by other applications, places data in the clipboard data structure, and unlocks the clipboard. Data items copied to the clipboard by `XmClipboardCopy` are not actually entered in the clipboard data structure until the call to `XmClipboardEndCopy`. It also frees the temporary storage that was allocated by `XmClipboardStartCopy`.

This function can return one of the following status return constants:

`ClipboardSuccess` The function is successful.

`ClipboardLocked` The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

8.2.2 Clipboard Inquire Functions

XmClipboardInquireCount

This function returns the maximum length for all data item formats.

```
int XmClipboardInquireCount (display, window, count,
max_format_name_length)
      Display *display;
      Window window;
      int *count;
      int *max_format_name_length;
```

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

count — Returns the number of data item formats available for the next-paste item in the clipboard. If no formats are available, this argument equals zero. The count includes the formats that were passed by name.

max_format_name_length — Specifies the maximum length of all format names for the next-paste item in the clipboard.

The `XmClipboardInquireCount` function returns the number of data item formats available for the next-paste item in the clipboard. This function also returns the maximum name length for all formats in which the next-paste item is stored.

This function can return one of the following status return constants:

`ClipboardSuccess` — The function is successful.

`ClipboardLocked` — The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

`ClipBoardNoData` — The function could not find data on the clipboard that corresponds to the format requested. This could occur because the clipboard is empty, there is no data on the clipboard in the format requested, or the data requested was passed by name and is no longer available.

XmClipboardInquireFormat

This function obtains the format name for the next paste data item in the clipboard.

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

display — Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

index Specifies which of the ordered format names to be obtained. If this index *i* is greater than the number of formats for the data item, `XmClipboardInquireFormat` returns a zero in the *copied_len* argument.

format_name_buf Specifies the buffer that receives the format name.

buffer_len Specifies the number of bytes in the format name buffer.

copied_len Specifies the number of bytes in the string copied to the buffer. If this argument equals zero, there is no *nth* format for the next-paste item.

The `XmClipboardInquireFormat` function returns a specified format name for the next-paste item in the clipboard. If the name must be truncated, the function returns a warning status. This function can return one of the following status return constants:

`ClipboardSuccess` The function is successful.

`ClipboardLocked` The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

`ClipboardTruncate` The data returned is truncated because the user did not provide a buffer that was large enough to hold the data.

`ClipBoardNoData` The function could not find data on the clipboard that corresponds to the format requested. This could occur because the clipboard is empty, there is no data on the clipboard in the format requested, or the data requested was passed by name and is no longer available.

XmClipboardInquireLength

This function obtains the length of the data stored under a specified format.

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

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

format_name Specifies the name of the format for the next-paste item.

length Specifies the length of the next data item in the specified format. This argument equals zero if no data is found for the specified format, or if there is no item on the clipboard.

The `XmClipboardInquireLength` function returns the length of the data stored under a specified format name for the clipboard data item. This is accomplished by passing a pointer to the length in the *length* parameter in the function.

If no data is found for the specified format, or if there is no item on the clipboard, `XmClipboardInquireLength` returns a value of zero.

This function can return one of the following status return constants:

`ClipboardSuccess` The function is successful.

`ClipboardLocked` The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

`ClipBoardNoData`	The function could not find data on the clipboard that corresponds to the format requested. This could occur because the clipboard is empty, there is no data on the clipboard in the format requested, or the data requested was passed by name and is no longer available.

XmClipboardInquirePendingItems

This function obtains a format name's list of data id or private id pairs.

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

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.
format_name	Specifies a string that contains the name of the format for which the list of data id/private id pairs is to be obtained.
item_list	Specifies the address of the array of data id/private id pairs for the specified format name. This argument is a type `XmClipboardPendingList`. The application is responsible for freeing the memory provided by this function for storing the list.
item_count	Specifies the number of items returned in the list. If there is no data for the specified format name, or if there is no item on the clipboard, this argument equals zero.

The `XmClipboardInquirePendingItems` function returns a list of data id/private id pairs for a specified format name. For the purposes of this function, a data item is considered pending if the application originally passed it by name, the application has not yet copied the data, and the item has not been deleted from the clipboard.

The application is responsible for freeing the memory provided by this function to store the list.

This function is used by an application when exiting to determine if the data that it passed by name should be sent to the clipboard.

This function can return one of the following status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

8.2.3 Clipboard Retrieve Functions

XmClipboardStartRetrieve

This function begins copying data incrementally from the clipboard.

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

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls.
timestamp	The time of the event that triggered the copy.

This routine tells the cut and paste routines that the application is ready to start copying an item from the clipboard. The clipboard will be locked by this routine and will stay locked until `XmClipboardEndRetrieve` is called. Between an `XmClipboardStartRetrieve` and an `XmClipboardEndRetrieve`, multiple calls to `XmClipboardRetrieve` with the same format name will result in data being incrementally copied from the clipboard until the data in that format has all been copied. The return value `ClipboardTruncate` from calls to `XmClipboardRetrieve` indicates that more data remains to be copied in the given format. It is recommended that any calls to the "inquire" functions that the application needs to make to effect the copy from the clipboard be made between the call to `XmClipboardStartRetrieve` and the first call to `XmClipboardRetrieve`. That way, the application does not need to call `XmClipboardLock` and `XmClipboardUnlock`. Applications do not need to use `XmClipboardStartRetrieve` and `XmClipboardEndRetrieve`, in which case `XmClipboardRetrieve` works as it did before.

XmClipboardRetrieve

This function obtains the current next paste data item from clipboard storage.

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

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.
format_name	Specifies the name of a format in which the data is stored on the clipboard.
buffer	Specifies the buffer to which the application wants the clipboard to copy the data.
length	Specifies the length of the application buffer.
num_bytes	Specifies the number of bytes of data copied into the application buffer.
private_id	Specifies the private data stored with the data item by the application that placed the data item on the clipboard. If the application did not store private data with the data item, this argument returns zero.

The `XmClipboardRetrieve` function retrieves the current data item from clipboard storage.

`XmClipboardRetrieve` returns a warning under the following circumstances:

- The data needs to be truncated because the buffer length is too short.
- The clipboard is locked.
- There is no data on the clipboard.

This function returns one of the following status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.
`ClipboardTruncate`	The data returned is truncated because the user did not provide a buffer that was large enough to hold the data.
`ClipboardNoData`	The function could not find data on the clipboard corresponding to the format requested. This could occur because (1) the clipboard is empty; (2) there is data on the clipboard but not in the requested format; and (3) the data in the requested format was passed by name and is no longer available.

XmClipboardEndRetrieve

This function suspends copying data incrementally from the clipboard.

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

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

`XmClipboardEndRetrieve` tells the cut and paste routines the application is through copying an item to the clipboard. Until `XmClipboardEndRetrieve` is called, data items can be retrieved incrementally from the clipboard by calling `XmClipboardRetrieve`. If the application calls `XmClipboardStartRetrieve`, it has to call `XmClipboardEndRetrieve`. If data is not being copied incrementally, `XmClipboardStartRetrieve` and `XmClipboardEndRetrieve` do not need to be called.

8.2.4 Miscellaneous Clipboard Functions

XmClipboardLock

This function locks the clipboard from access by other applications.

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

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls.

The `XmClipboardLock` function locks the clipboard from access by another application until you call `XmClipboardUnlock`. All clipboard functions lock and unlock the clipboard to prevent simultaneous access. The `XmClipboardLock` and `XmClipboardUnlock` functions allow the application to keep the clipboard data from changing between calls to the inquire functions and other clipboard functions. The application does not need to lock the clipboard between calls to `XmClipboardStartCopy` and `XmClipboardEndCopy`, but it should do so before multiple calls to `XmClipboardCopyByName`.

If the clipboard is already locked by another application, `XmClipboardLock` returns an error status.

Multiple calls to `XmClipboardLock` by the same application increase the lock level.

This function can return one of the following status return constants:

`ClipboardSuccess` The function is successful.

`ClipboardLocked` The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardRegisterFormat

This function registers a new format.

```
int ClipboardRegisterFormat (display, format_name, format_length)
  Display   *display;
  char      *format_name;
  unsigned long format_length;
```

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

format_name Specifies the string name for the new format.

format_length Specifies the format length in bits (8, 16, or 32).

Each format stored on the clipboard should have a length associated with it and known to the cut and paste routines. All of the formats specified by the *Inter-Client Communication Conventions* manual are pre-registered (formats are referred to as "targets" in the ICCCM). Any other format that the application wants to use must be registered via this routine. Failure to register the length of the data will result in applications not being compatible across platforms having different byte swapping orders.

XmClipboardUnlock

This function unlocks the clipboard.

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

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay` or `XtInitialize`.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.

remove_all_locks Specifies a boolean value that, when True, indicates that all nested locks should be removed. If False, indicates that only one level of lock should be removed.

The `XmClipboardUnlock` function unlocks the clipboard, enabling it to be accessed by other applications.

If multiple calls to `XmClipboardLock` have occurred, then the same number of calls to `XmClipboardUnlock` is necessary to unlock the clipboard, unless the *remove_all_locks* argument is True.

The application should lock the clipboard before making multiple calls to XmClipboardCopyByName and should unlock the clipboard after completion.

This function can return one of the following status return constants:

ClipboardSuccess	The function is successful.
ClipboardLocked	The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

XmClipboardWithdrawFormat

This function indicates that the application is no longer willing to supply a data item to the clipboard.

```
int XmClipboardWithdrawFormat(display, window, data_id)
     Display *display;
     Window window;
     int data_id;
```

display	Specifies a pointer to the Display structure that was returned in a previous call to XOpenDisplay or XtInitialize.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each of the clipboard functions that it calls. Note that this window *must* be associated with a widget.
data_id	Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This was assigned to the item when it was originally passed by XmClipboardCopy.

The XmClipboardWithdrawFormat function indicates that the application will no longer supply a data item to the clipboard that the application had previously passed by name.

This function can return one of the following status return constants:

ClipboardSuccess	The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function again with the same parameters until the lock goes away. This gives the application the opportunity to ask if the user wants to keep trying or to give up on the operation.

8.3 Dynamic Resource Defaulting

Dynamic resource defaulting is a mechanism that incorporates a processing function into a widget's resource definitions. The widget can use this mechanism to calculate a default resource value when it (the widget) is created, instead of having the resource default be static. The widget set uses this capability to determine much of its visual resource defaults at run time. This allows the widget to make more sensible choices for color and pixmap defaults.

All of the color resources and pixmap resources that represent visual data are dynamically defaulted. This includes the resources `XmNforeground`, `XmNbackground`, `XmNbackgroundPixmap`, `XmNtopShadowColor`, `XmNtopShadowPixmap`, `XmNbottomShadowColor`, `XmNbottomShadowPixmap`, `XmNhighlightColor`, and `XmNhighlightPixmap`.

Color and pixmap data are set as shown below.

- Set to black and white if a monochrome system is used.
- If a color system is used:
  - A "default" color scheme, *or*
  - A color scheme based on the background resource, `XmNbackground`.

Part of the design for the widget set and window manager includes an algorithmic approach for generating color schemes. This means that by specifying the background color, the foreground and two shadowing colors are calculated.

8.4 Key/Keyboard Grabbing

The OSF/Motif library provides several routines for redirecting keyboard events within a widget hierarchy. These routines are front-ends to the Xlib routines that provide key/keyboard grabbing.

The client should use these routines rather than the raw Xlib routines so that the intrinsics can be aware of grabs and process them in correct priority relative to the intrinsics focus and modal cascade management. Grabs will have priority over the focus redirection but will be overridden by the modal cascade in the case of contention.

8.4.1 Passive Grabs

XtGrabKey and XtUngrabKey allow the client to redirect the specified keyevent (as described by keysym modifiers) to the root widget of a hierarchy. The root widget is the widget parameter to the XtGrabKey call and all keyevents that would have been dispatched to other subwindows will get dispatched to it. Note that the use of owner_events = True is not meaningful for XtGrabKey since all widgets in the hierarchy belong to the grabbing client. In addition, the pointer_mode and keyboard_mode fields are currently forced to GrabModeAsync in order to avoid deadlock conditions.

void XtGrabKey (*widget, keysym, modifiers, owner_events, pointer_mode, keyboard_mode*)
  Widget *widget;*
  Keycode *keycode;*
  unsigned *int modifiers;*
  Boolean *owner_events;*
  int *pointer_mode;*
  int *keyboard_mode;*

XtGrabkey establishes a passive grab on the specified keys, such that when the specified key/modifier combination is pressed, the keyboard is grabbed. If owner events is false, then all key events will be dispatched to the grab widget.

void XtUngrabKey (*widget, keysym, modifiers*)
  Widget *widget;*
  Keycode *keycode;*
  unsigned int *modifiers;*

XtUngrabKey cancels the passive grab on the key combination on the specified widget. A modifier of AnyModifier is equivalent to issuing the request for all possible modifier combinations. A keysym of AnyKey is equivalent to issuing the request for all possible non-modifier key codes. This call has no effect on an active grab.

8.4.2 Active Grabs

```
int XtGrabKeyboard (widget, owner_events, pointer_mode, keyboard_mode,time)
    Widget widget;
    Boolean owner_events;
    int                 pointer_mode;
    int keyboard_mode;
    Time time;
```

XtGrabKeyboard actively grabs control of the main keyboard. If the grab is successful, it returns the constant GrabSuccess. Further key events are reported to the grab widget.

```
void  XtUngrabKeyboard (widget, time)
  Widget widget;
  Time time;
```

XtUngrabKeyboard releases any active grab on the keyboard.

8.5 Localization

The R3 version of the Xt Intrinsics do not support localization of resource files. While the X Consortium is planning to support localization in the R4 version of the Xt Intrinsics, the OSF/Motif Intrinsics have interim support that is compatible with both X/Open and the OSF/Motif Resource Manager (MRM).

You can specify a language by using the LANG environment variable (see the man page *Environ*(5) for information on this variable). Elements of this variable are then used to establish a path to the proper resource files. The following substitutions are used in building the path:

- %N is replaced by class_name of the application.
- %L is replaced by the value of LANG environment variable.
- %l is replaced by the language part of LANG environment variable.
- %t is replaced by the territory part of LANG environment variable.
- %c is replaced by the codeset part of LANG environment variable.
- %% is replaced by %.

If the LANG environment variable is not defined, or if one of its parts is missing, then a % element that references it is replaced by NULL.

The paths contain a series of elements separated by colons. Each element denotes a file name, and the file names are looked up left to right until one of them succeeds. Before

doing the lookup, substitutions are performed.

NOTE

We are using the X/Open convention of collapsing multiple adjoining slashes in a filename into one slash.

The `XtInitalize` function loads the resource database by merging in resources from these sources:

- Application-specific *class* resource file on the local host.
- Application-specific *user* resource file on the local host.
- Resource property on the server or user preference resource file on the local host.
- Per-host user environment resource file on the local host.
- The application command line (`argv`).

To load the application-specific class resource file, `XtInitialize` performs the appropriate substitutions on this path:

`/usr/lib/X11/%L/app-defaults/%N:/usr/lib/X11/app-defaults/%N`

If the LANG environment variable is not defined (or the first path lookup using LANG fails), then the lookup will default to the current non-language specific location (`/usr/lib/X11/app_defaults/%N`).

To load the user's application resource file, `XtInitialize` performs the following steps:

1. Use XAPPLRESLANGPATH to look up the file.
2. If that fails, or if XAPPLPRESLANGPATH is not defined, and if XAPPLRESDIR is defined, use the following as the path:

   XAPPLRESDIR%L/%N:XAPPLRESDIR/%N

   else use:

   $HOME/%L/%N:$HOME/%N

Note that if the XAPPLRESLANGPATH lookup is not successful and LANG is not defined, the lookup is then equivalent to that used by the R3 specification of `XtInitialize` (actually described under `XtDisplayInitialize`).

8.6 Pixmap Caching Functions

The pixmap caching functions provide the application and widget writer with a means of associating an image with a name. Given this association, these functions can generate pixmaps through references to a .Xdefaults file (by name) and through an argument list (by pixmap), for all widgets that have pixmap resources. A cache of all pixmaps is automatically maintained. This improves performance and decreases server data space when requesting identical pixmaps.

The pixmap caching provides four functions by which the application or widget writer can install images, uninstall images, create pixmaps, and destroy pixmaps.

```
Boolean XmInstallImage (image, image_name)
    XImage   *image;
    char     *image_name;
```

image Points to the image structure to be installed. The installation process does not make a local copy of the image, therefore the application should not destroy the image until it is installed from the caching functions.

image_name Specifes a string that the application uses to name the image. After installation, this name can be used in a `.Xdefaults` file for referencing the image. A local copy of the name is created by the image caching functions.

`XmInstallImage` is used to give to the caching routines an image that can later be used to generate a pixmap. Part of the installation process is to extend the resource converter used to reference these images. The resource converter can access the image name so that the image can be referenced in a .Xdefaults file. Since an image can be referenced by a widget through its pixmap resources, it is up to the application to ensure that the image is installed before the widget is created. *image* is a pointer to the image structure to be installed. The installation process does not make a local copy of the image. Therefore, the application should not destroy the image until it is uninstalled from the caching functions. *image_name* is a string the application uses to name the image. After installation this name can be used in a .Xdefaults for referencing the image. A local copy of the name is created by the image caching functions.

The image caching functions provide a set of eight preinstalled images. These names can be used within a .Xdefaults file for generating pixmaps for the resource they are provided for.

TABLE 8-3. Preinstalled Images

Image Name	Description
background	A tile of solid background
25_foreground	A tile of 25% foreground, 75% background
50_foreground	A tile of 50% foreground, 50% background
75_foreground	A tile of 75% foreground, 25% background
horizontal	A tile of horizontal lines of the two colors
vertical	A tile of vertical lines of the two colors
slant_right	A tile of slanting lines of the two colors
slant_left	A tile of slanting lines of the two colors

```
Boolean XmUninstallImage (image)
            XmImage  *image;
```

image Points to the image structure given to the `XmInstallImage` routine.

XmUninstallImage provides the mechanism by which an image can be removed from the caching routines. *image* is a pointer to the image given to the `XmInstallImage()` routine.

An application or widget makes a call to extract a pixmap when the images have been installed or to access a set of the predefined images. When an application or widget is finished with a pixmap, it can call a function to destroy the pixmap. These functions are defined as follows.

```
Pixmap XmGetPixmap (screen, image_name, foreground, background)
    Screen  *screen;
    char    *image_name;
    Pixel   foreground;
    Pixel   background;
```

screen Specifies the display screen on which the pixmap is to be drawn and is used to ensure that the pixmap matches the visual required for the screen.

image_name Specifies the name of the image to be used to generate the pixmap.

foreground Combines the image with the *foreground* color to create the pixmap if the image referenced is a bit-per-pixel image.

background Combines the image with the *background* color to create the pixmap if the image referenced is a bit-per-pixel image.

`XmGetPixmap` uses the parameter data to perform a lookup in the pixmap cache to see if a pixmap has already been generated that matches the data for the specified screen. If one

is found, a reference count is incremented and the pixmap is returned. If one is not found, the image corresponding to *image_name* is used to generate a pixmap which is then cached and returned. *screen* contains the display screen on which the pixmap is to be drawn and is used to ensure the pixmap matches the visual required for the screen. *image_name* is the name of the image to be used to generate the pixmap. If a "bit-per-pixel" image is being accessed, *foreground* and `background` are combined with the image to create the pixmap.

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

screen Specifies the display screen for which the pixmap was requested.

pixmap Specifes the pixmap to be destroyed.

`XmDestroyPixmap` is used to remove pixmaps that are no longer needed. A pixmap will only be completely freed when there is no further references to it.

8.7 Resolution Independence

The OSF/Motif library provides a mechanism built into the widgets called "resolution independence." Resolution independence allows applications to create and display images that are the same physical size regardless of the resolution of the display. This frees the application developer from the task of ensuring that an application can be used on a wide range of systems.

The resolution independence mechanism provides resource data to the widgets in various unit types, including millimeters, inches, points, and font units. All widget resources connected with size, position, thickness, padding, and spacing can be set using the above unit types. The application or user of the application can provide resolution independent data through .Xdefaults and app-defaults files, command line arguments, or arg lists.

8.7.1 The Resolution Independence Mechanism

The units a widget uses is defined by the resource XmNunitType, which can be found in the base classes of XmPrimitive, XmGadget, and XmManager. Since all widgets are built from these base classes, it follows that all widgets support resolution independence. There are five values XmNunitType can have:

- `XmPIXELS` - All values provided to the widget are treated as normal pixel values. This is the default value for the resource.
- `Xm100TH_MILLIMETERS` - All values provided to the widget are treated as 1/100 of a millimeter.

- Xm1000TH_INCHES - All values provided to the widget are treated as 1/1000 of an inch.
- Xm100TH_POINTS - All values provided to the widget are treated as 1/100 of a point. A point is a unit typically used in text processing applications and is defined as 1/72 of an inch.
- Xm100TH_FONT_UNITS - All values provided to the widget are treated as 1/100 of a font unit. The value to be used for the font unit is determined in one of two ways. The resource XmNfont can be used in a defaults file or on the command line. The standard command line options of `-fn` and `-font` can also be used. The font unit value is taken as the QUAD_WIDTH property of the font. The function `XmSetFontUnits` allows applications to specify the font unit values. This function is defined later in this section.

There are two reasons for the unit types to be fractional.

- It allows all calculations to be done in integer representation. This ensures maximum performance for type conversions.
- There is no way to supply a floating point number to a widget through an argument list. This is because the value field in the argument list is of type `char *`. When a floating point value is forced into a `char *` variable, the fractional part is truncated.

When a widget is created and its unit type is something other than pixels, it converts the data specified by the application or the user into pixel values, taking into account the resolution of the screen. These converted pixel values are then placed into the internal data space of the widget, and the widget operates as it would normally. The same process occurs when the application issues an `XtSetValues()` to a widget. The new values are converted from unit type to pixels and placed back into the widget.

When the application issues an `XtGetValues()` to a widget, the pixel values are taken out of the widget, converted back to the unit type, and inserted into the argument list to be returned.

The conversion and storing of unit type values to pixel values can cause some rounding errors. Therefore, when an application issues an `XtGetValues()`, it should not expect exactly the same data to be returned as was originally specified. This rounding error will only occur once and will not get progressively worse. For example, if a widget's width is set to 1000/1000 inches (1 inch), `XtGetValues()` may return 993/1000 inches. If this value is then used to set the width of a second widget and the application calls `XtGetValues()` on the second widget, 993/1000 inches will be returned.

8.7.2 Setting the Font Units

Applications may want to specify resolution independent data based on a global font size. The widget set provides an external function to use to initialize the font unit values. This function needs to be called before any widgets with resolution independent data are created.

```
XmSetFontUnits (display, font_unit_value)
Display * display;
int font_unit_value;
```

The parameters for this function are described below.

display Defines the display for which this font unit value is to be applied.

font_unit_value Specifies the value to be used in the conversion calculations. The font unit value is normally taken as the QUAD_WIDTH property of the font. However, the application can specify any integer value.

8.7.3 Converting Between Unit Types

The widgets use a general conversion function to convert between pixels and other unit types. This function can convert values between any of the defined unit types, and is available to the application for its use.

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

XmConvertUnits uses the parameter data to convert the value and return it as the return value from the function. The parameters for this function are as follows:

widget Specifies the widget for which the data is to be converted.

orientation Specifies whether the converter should use the horizontal screen resolution or vertical screen resolution when performing the conversions. *orientation* can have values of XmHORIZONTAL or XmVERTICAL.

from_unit_type Specifies the current unit type of the supplied value.

from_value Specifies the value to be converted.

to_unit_type Specifies the unit type into which the value should be converted.

8.8 Interacting With the OSF/Motif Window Manager

This section explains the procedures an application can use to interact with the WM_PROTOCOLS and system menu facilities provided by the OSF/Motif Window Manager (MWM). You should be familiar with the concepts presented in the Inter-Client Communications Conventions manual (ICCCM).

Many of the proposed Inter-Client Communications Conventions (ICCC) are not currently supported by the X Consortium libraries (Xlib and Xt), because the the conventions were not official when the R3 version of the X Window System was introduced. The X Consortium has indicated its intent to add support for the ICCC in the next X11 release (R4). The OSF/Motif library supports a minimal set of inter-client (mainly application-to-window manager) communication services and provides them as a layer above existing intrinsics facilities.

8.8.1 Protocol Management

The Protocol management functions are a set of general purpose routines for interacting with properties that contain atom arrays, client messages, and associated callbacks. They are used to support the existing entries for the WM_PROTOCOLS and _MOTIF_WM_MESSAGES properties. See *Programming With the Xt Intrinsics* for more information.

NOTE

In the following discussion, the names of atoms or properties are in upper case and are obtained by "interning" the strings with the server. Use `XmInternAtom` to convert these strings to a 32-bit tag. The following code segment shows how to obtain the _MOTIF_WM_MESSAGES atom.

```
Atom motif_wm_messages;

  motif_wm_messages = XmInternAtom(display,
    "_MOTIF_WM_MESSAGES", true);
```

Alternatively, you could use _XA_MOTIF_WM_MESSAGES (defined in <X11/MwmUtil.h>) as the second argument to `XmInternAtom`.

A protocol is a 32-bit tag that is used by clients to communicate with the window manager. This tag is either an X Atom or an arbitrary long integer variable whose value is shared by the parties to the protocol communication. The client indicates interest in certain

communications protocols by adding these tags to a tag array that is the value of a special property on its top-level window. For pre-defined ICCC protocols, this property is WM_PROTOCOLS. For the OSF/Motif Window Manager, this property is _MOTIF_WM_MESSAGES. The window manager sends a protocol message (when appropriate) in the form of a client message event with the *message_type* field of the ClientMessage structure set to the property and the *data.l[0]* field set to the protocol. The client can associate a callback list with the protocol that is invoked when the client message event is received.

Each shell can have one protocol manager per property associated with it and the protocol manager can have multiple protocols registered. Each protocol is identified by its tag (WM_SAVE_YOURSELF, for example). The protocols can have any number of client callbacks associated with them, in addition to pre-hook and post-hook callbacks (usually registered by the widget set) for each protocol.

- Tracks the state of the protocols whether or not they are active.
- Tracks the state of the Shell and creates and updates the protocol property accordingly.
- Processes the client messages received and invokes the appropriate callbacks.

_MOTIF_WM_MESSAGES

The client uses the _MOTIF_WM_MESSAGES property to indicate to MWM which messages (sent by MWM when an f.send_msg function is invoked from the MWM system menu) it is currently handling. A client can add f.send_msg entries to the menu by using the .mwmrc file or by using the XmNmwmMenu resource of VendorShell. This resource is a string that is parsed by MWM to determine what to display in the system menu and and how to react to an item's selection. When the action associated with an item is f.send_msg, MWM sends the client a message if the specified protocol is active. The protocol is the integer argument to the f.send_msg action. A menu protocol is active if the protocol is in the _MOTIF_WM_MESSAGES property and the _MOTIF_WM_MESSAGES atom is in the WM_PROTOCOLS property. Otherwise, the protocol is inactive and the menu label will be "grayed out."

WM_PROTOCOLS

There is a corresponding macro provided for each of the general protocol manager routines to simplify their use. The only difference between them is that the general routines are passed a protocol property in all calls while the macros *always* force this property to WM_PROTOCOLS. These macros are useful if you want to interact with ICCC protocols such as WM_DELETE_WINDOW or WM_SAVE_YOURSELF.

Note that if you are using the protocol manager for the system menu, the property should be the atom corresponding to _MOTIF_WM_MESSAGES.

8.8.2 Protocol Manager Functions

The following sections list the Protocol Manager functions. There is a sample program, `xmprotocol`, in the directory `/usr/contrib/Xm` that adds or deactivates entries to the system menu. You can use the methods presented in that program to get an idea of how to use the functions discussed in this section.

Note that the statement

```
#include <X11/Protocols.h>
```

must be present in any program using these functions.

The functions that have the letters "WM" are the macros referred to earlier. Each function has a corresponding macro. For example, `XmAddProtocols` has a corresponding macro `XmAddWMProtocols`. The macro simply calls `XmAddProtocols` with the *property* parameter set to XA_WM_PROTOCOL.

Add and Remove Functions

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

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

This routine adds the protocols to the Protocol manager corresponding to the specified property and allocate the internal tables. The protocols are initialized to active.

`void XmRemoveWMProtocols` (*shell, protocols, num_protocols*)
`Widget` *shell;*
`Atom` **protocol;*
`Cardinal` *num_protocols;*

`void XmRemoveProtocols` (*shell, property, protocols, num_protocols*)
`Widget` *shell;*
`Atom` *property;*
`Atom` **protocols;*
`Cardinal` *num_protocols;*

This routine removes the protocols from the Protocol manager and deallocates the internal

tables. It also updates the handlers and the property if any of the protocols are active and the shell referenced in the *shell* parameter is realized.

Protocol State

It is sometimes useful to allow a protocol's state information (callback lists for example) to persist, even though the client may choose to temporarily resign from the interaction. The main use of this capability is to gray out `f.send_msg` labels in the system menu. This is supported by allowing a protocol to be in one of two states, active or inactive. If the protocol is active and the shell is realized, then the property contains the protocol atom. If the protocol is inactive, then the atom is not present in the property.

```
void XmActivateWMProtocol (shell, protocol)
Widget shell;
Atom protocol;
```

```
void XmActivateProtocol (shell, property, protocol)
Widget shell;
Atom property;
Atom protocol;
```

If the protocol is inactive, this routine updates the handlers and adds the protocol to the property if the shell is realized.

```
void XmDeactivateWMProtocol (shell, protocol)
Widget shell;
Atom protocol;
```

```
void XmDeactivateProtocol (shell, property, protocol)
Atom property;
Cardinal num_protocols;
```

If the protocol is inactive, this routine updates the handlers and the property if the shell is realized.

Protocol Callbacks

When a client message associated with a protocol is received by the protocol manager, it checks to see if the protocol is active. If it is, then any callbacks associated with the protocol are called. There are three callback lists that can be associated with a protocol. One is for client use and is accessed by `XmAddProtocolCallbacks` and `XmRemoveProtocolCallbacks`. The other two (the pre- and post- hook callbacks) are intended for toolkit use and are accessed by the `XmSetProtocolHooks` routine. The hook routines are called before and after the client callbacks (if any) are called. The protocol callbacks have a reason field of XmCR_PROTOCOLS and a type of XmAnyCallbackStruct.

```
void XmAddWMProtocolCallbacks (shell, protocol, callback, closure)
Widget shell;
Atom protocol;
XtCallbackProc callback;
caddr_t closure;
```

```
void XmAddProtocolCallbacks (shell, property, protocol, callback, closure)
Widget shell;
Atom property;
Atom protocol;
XtCallbackProc callback;
caddr_t closure;
```

This routine checks to see if the protocol is registered and if not, it calls XmAddProtocols. It then adds the callbacks to the internal list. These callbacks are called when the corresponding client message is recieved.

```
void XmRemoveWMProtocolCallbacks (shell, protocol, callback, closure)
Widget shell;
Atom protocol;
XtCallbackProc callback;
caddr_t closure;
```

```
void XmRemoveProtocolCallbacks (shell, property, protocol, callback, closure)
Widget shell;
Atom property;
Atom protocol;
XtCallbackProc callback;
caddr_t closure;
```

This routine removes the callback from the internal list.

```
void XmSetWMProtocolHooks (shell, protocol, prehook, pre_closure, posthook, post_closure)
Widget shell;
XtCallbackProc prehook, posthook;
caddr_t pre_closure, post_closure;
```

```
void XmSetProtocolHooks (shell, protocol, property, prehook, pre_closure, posthook, post_closure)
Widget shell;
Atom property;
XtCallbackProc prehook, posthook;
caddr_t pre_closure, post_closure;
```

This routine is used by toolkit widgets that want to have "before and after" actions executed when a protocol message is recieved from the window manager. Since there is no

guaranteed ordering in execution of event handlers or callback lists, this allows the shell to control the flow while leaving the protocol manager structures private. Call_data will contain the same pointer as that passed to the client callbacks.

8.8.3 Atom Management

The atom management routines mirror the Xlib interfaces for atom management, but provide client side caching. When (and where) caching is provided in Xlib, the routines will become pseudonyms for the Xlib routines. Note that the statement

```
#include <X11/AtomMgr.h>
```

must be present in any program using these functions.

```
Atom XmInternAtom (display, name, only_if_exists)
  Display *display;
  String name;
  Boolean only_if_exists;

String XmGetAtomName (display, atom)
  Display *display;
  Atom    atom;
```

8.9 OSF/Motif Version Number

The OSF/Motif library provides a macro, `XmVersion`, that returns the current OSF/Motif version. Essentially, the macro multiplies the version number of the library by 1000 and adds the revision number. For example, in the first release of OSF/Motif Version 1.0, the macro would return 1000.

Additionally, a global variable, `XmUseVersion` is provided. The value of this variable is set to reflect the value returned by `XmVersion` as soon as the first widget is created (the setting takes place during the class initialization procedure of the widget). In the future, an application may be able to set this variable to specify the kind of behavior the widget library should provide.

8.10 OSF/Motif Window Manager Presence

Very often the need arises to be able to determine whether the OSF/Motif Window Manager (MWM) is running on a given display. The OSF/Motif function `XmIsMotifWMRunning` can provide this information.

```
#include <X11/Shell.h>

Boolean XmIsMotifWMRunning (shell)

 Widget shell;
```

shell specifies the shell whose screen should be tested for MWM's presence. The function returns True if MWM is running, False if it is not.

Keyboard Interface 9

The keyboard interface allows the user to interact with an application using the keyboard in place of, or as a supplement to, the mouse. This capability is necessary in a variety of situations, such as mouseless systems or applications that don't want to force the user to switch back and forth between the keyboard and mouse.

The keyboard interface involves two major components:

- Keyboard focus and traversal from widget to widget.
- Keyboard input processing to an individual widget.

9.1 Keyboard Focus Models

Traversal provides the means of moving the "keyboard focus" within an application. The keyboard focus indicates which widget is currently active. When a particular widget has keyboard focus, all keyboard input directed at the application goes to that widget, regardless of the location of the pointer.

The OSF/Motif system supports two focus models:

- The "pointer driven" focus model. In the "pointer driven" model, a widget receives keyboard input only when the cursor is positioned within the widget's bounding rectangle; moving the cursor out of the widget causes it to lose focus.
- The "click-to-type" focus model. In the "click-to-type" model, when the window manager passes the focus to the topmost shell widget, the topmost shell widget redirects the focus to one of its descendents. The user can move the focus to another descendent of the topmost shell widget either by pressing the arrow or tab keys, or by clicking mouse button 1 in a widget. Clicking mouse button 1 in a widget may cause that widget to ask for and receive the input focus. When a descendent has focus, it continues to receive all keyboard input until either of the following occur:
  - The user requests that the focus be moved to another descendent of the topmost shell widget.
  - The window manager takes the focus away from the topmost shell widget.

An application sets the desired model by means of the XmNkeyboardFocusPolicy resource, which is exported by the VendorShell widget class. The specified focus model is active for the complete widget hierarchy built from the topmost shell widget.

The functionality described in the rest of this chapter applies only to the "click-to-type" focus model. The OSF/Motif menu system provides its own type of keyboard traversal. This is explained in chapter 6, "Menus."

Only Primitive widgets and gadgets can have the keyboard focus, since these are the only widgets with which the user interacts; other widgets are merely containers. In this discussion, gadgets are considered comparable to Primitive widgets.

Each Primitive widget has a boolean resource, XmNtraversalOn, that specifies whether or not the widget will accept the focus. The default is False, which denies focus. The resource must be set to True in order for the widget to accept the focus.

When a widget has accepted the keyboard focus, a highlight is drawn around the widget.

9.2 Grouping Widgets Into Tab Groups

The OSF/Motif system uses the concept of "tab groups" to group Primitive widgets. Any Manager or Primitive widget can be defined as a tab group. If a Manager widget is in a tab group, its Primitive children are part of the tab group.

Two functions manage the addition and deletion of tab groups for an application.

`XmAddTabGroup` adds the specified tab group to the list of tab groups associated with a particular widget hierarchy.

`XmAddTabGroup(`*tab_group*`)`
`Widget` *tab_group*;

tab_group Specifies the Manager or Primitive widget that defines a tab group.

`XmRemoveTabGroup` removes the tab group from the list of tab groups associated with a particular widget hierarchy.

`XmRemoveTabGroup(`*tab_group*`)`
`Widget` *tab_group*;

tab_group Specifies the Manager or Primitive widget that defines a tab group.

9.3 Traversal Within and Between Tab Groups

Traversal involves two types of focus changes – changing the focus to a different widget within a particular tab group and changing the focus to another tab group.

Movement among the Primitive widgets within a tab group is controlled by the order in which the widgets were created. The following keys change the focus to another widget in the same tab group:

- The down arrow key moves the focus to the next widget for which the XmNtraversalOn resource has been set to True. When the focus reaches the end of the tab group, it wraps to the beginning. The right arrow key has the same effect unless its behavior is defined by the particular widget. For example, a text widget configured for single-line edit defines the behavior of the right arrow key; therefore, that key does not change the focus.
- The up arrow key moves the focus to the previous widget. When the focus reaches the beginning of the tab group, it wraps to the end. The left arrow key has the same effect unless its behavior is defined by the particular widget.
- The home key moves the focus to the first Primitive widget in the tab group.

Movement between tab groups is controlled by the order in which the application has registered the tab groups.

- The tab key or function key 6 (f6) moves the focus to the first widget in the next tab group. When the focus reaches the end of the tab group list, the focus wraps to the beginning of the list.
- Shift Tab moves the focus to the first widget in the previous tab group. When the focus reaches the beginning of the tab group list, it wraps to the end of the list.

Clicking mouse button 1 within certain widgets (typically, text widgets) moves the focus to the indicated widget. The focus remains there until either the widget hierarchy loses the focus, or until the user moves the focus to another widget. A widget must have its XmNtraversalOn resource set to True in order to get focus this way.

Certain widgets must be placed within their own tab group; that is, the widget cannot be included in a tab group containing other widgets.

- Each List widget and the ScrollBar widget must be registered as its own tab group, since they define special behavior for the arrow keys.
- Each multiline text widget must be registered as its own tab group, since it defines special behavior for both the arrow keys and the Tab keys. f6 must be used to switch to another tab group. Single-line text widgets do *not* have this requirement.

- The Option menu widget must be registered as its own tab group because it consists of two internal Primitive widgets.

9.4 Keyboard Input Processing to a Widget

Keyboard input into a widget that has focus is handled by definitions of the widget's default translations for keyboard input. Refer to the *HP OSF/Motif Programmer's Reference Manual* for the default translations.

Debugging 10

No matter how careful you are, sooner or later you will need to debug a program. Because of the underlying Xt Intrinsics functions, this can be somewhat confusing. This chapter discusses some basic tools to use when debugging OSF/Motif programs.

10.1 Debugging Tools

The HP-UX system provides several very useful debugging tools to help you debug programs. These tools are adb, cdb, fdb, pdb, and xdb. The HP 9000 Series 800 computers use xdb, while all of them can be used on the HP 9000 Series 300 computers. The cdb, fdb, and pdb debuggers are for C, Fortran, and Pascal, respectively. The commands used in these debuggers are very similar. There are man pages for adb, cdb, and xdb in section 1 of the *HP-UX Reference Vol 1: Sections 1, 1M, and 9*. There are also descriptions of adb and cdb in the *Programming Environment HP-UX Concepts and Tutorials* manual. Discussion and examples of debugging procedures in this chapter will use cdb. The procedures shown can be followed for the other debuggers, although the exact commands may differ somewhat. You should refer to the man page for the debugger you intend to use.

The next sections describe what to do when certain faults or error conditions occur. The examples given should provide you with an understanding of the basic procedures involved in debugging.

10.2 Segmentation Fault

In many cases the OSF/Motif system will provide an error message or a warning that will provide a clue as to the cause of the problem. Unfortunately, that does not always happen. For example, a program can abort with only the message "Segmentation Fault (core dumped)." This is not very informative, and in this case you must resort to debugging. We will discuss two methods to accomplish this.

10.2.1 C Debugger

The C Debugger (cdb for short) is the most suitable tool to use in this case, however it is somewhat complex. If you are not familiar with cdb, see the section on it in the *Programming Environment HP-UX Concepts and Tutorials* manual.

Take a look at the following program:

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/X.h>
#include <X11/Intrinsic.h>
#include <X11/Core.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/List.h>

Widget  toplevel;
Widget outer_box;

Arg myArgs[20];

void  SelectProc(w,closure,call_data)
  Widget w;
  caddr_t  closure, call_data;
{
  printf("Single Selection.0);
}

void main(argc, argv)
  unsigned int argc;
  char **argv;
{
  int i;

  toplevel = XtInitialize(argv[0], "DBtest", NULL, 0, &argc, argv)

  i = 0;

  XtSetArg(myArgs[i], XmNselectedItemCount, (XtArgVal) 10); i++;
  XtSetArg(myArgs[i], XmNvisibleItemCount, (XtArgVal) 5); i++;

  XtAddCallback(outer_box, XmNsingleSelectionCallback, SelectProc)
```

```
  outer_box = XmCreateScrolledList(toplevel, "ListWidget",
    myArgs, i);

  XtManageChild(outer_box);

  XtRealizeWidget(toplevel);

  XtMainLoop();
}
```

This is a simple program that is supposed to create a ScrolledList widget. Unfortunately, there is a bug within the program that will cause a "segmentation fault." The debugging process using cdb is described below.

First, the program must be compiled with the "-g" option to the "cc" command. The C Debugger will not work unless this is done. When you have the program properly compiled (we'll assume you've named it `showdebug`), run the program under cdb by using the command `cdb showdebug`. You'll see a window like that shown in figure 10-1.

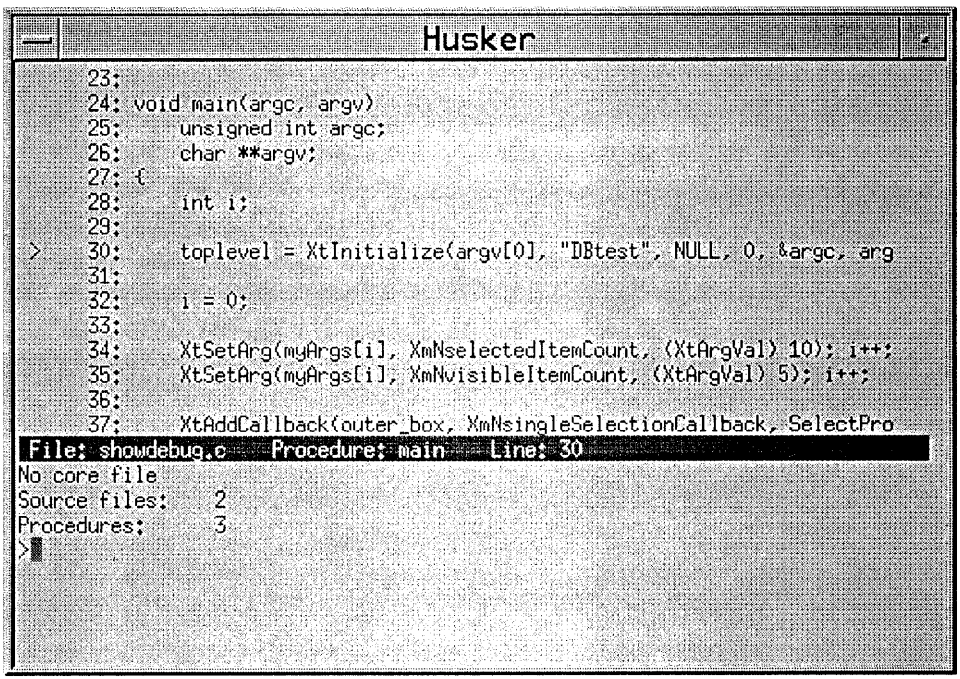


Figure 10-1. Running a Program With cdb

At this point the program is not actually running. The upper part of the window has a ">" pointing to the first executable instruction in the program. The lower window is the window in which you'll enter commands. To actually start the program, simply type "r" (for run) and press Return. You'll see the message "segmentation violation (no ignore)..." in the lower window. This is shown in figure 10-2.

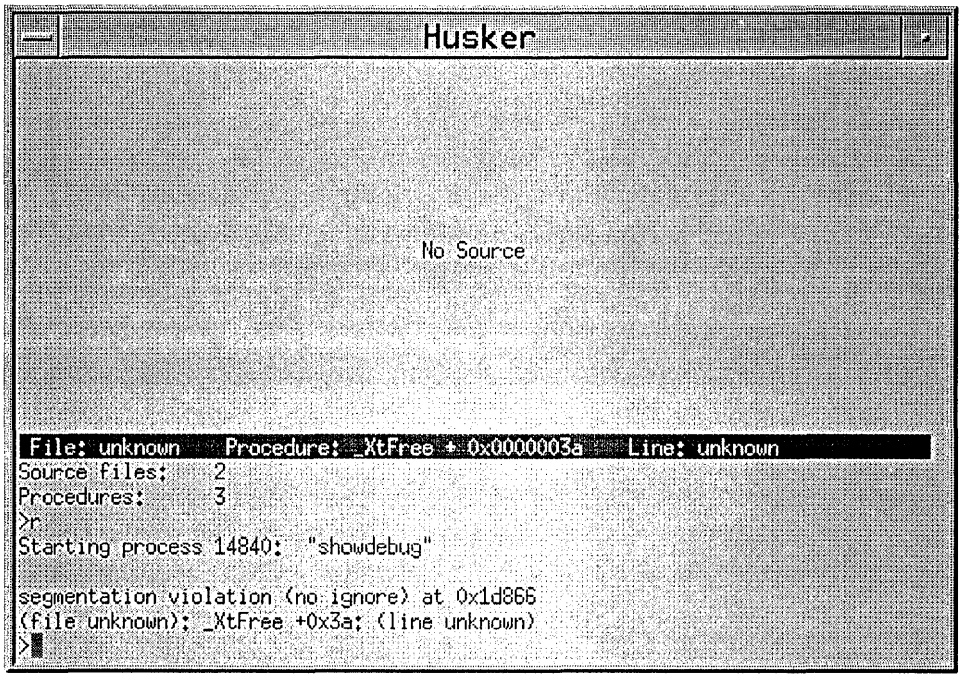


Figure 10-2. Results of cdb Trace

Now type "t" (for trace) and press Return. What you see in the lower window is a trace of events that occurred, as shown in figure 10-3. This tells you that program `main` called `XtAddCallback` at line 37 and `XtAddCallback` called `XtFree` which then called itself. Notice that the upper window simply says "no source." Now type "3E" in the lower window (the "3E" stands for "enter the third function listed") and press Return. The source code for `main` appears in the upper window with a ">" pointing to line 37, the offending line, as shown in figure 10-3.

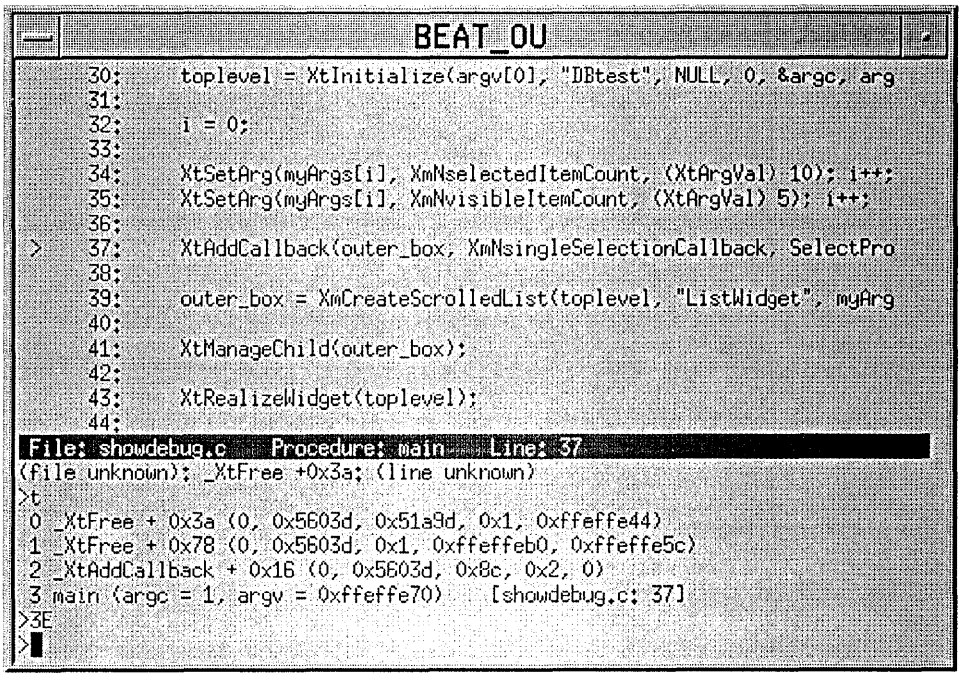


Figure 10-3. Entering a Function From cdb

You can see that the problem came about as a result of the call to `XtAddCallback` at line 37 of program `main`. But what is there about this line that causes the abort? In figure 10-3, notice the values for the parameters of `XtAddCallback`. The value of the first parameter is 0, yet line 37 of `main` shows that this parameter is supposed to be the value in outer_box, a widget id. If you type "*outer_box" in the lower window, you will see that its value is indeed 0, meaning that it did not really exist. Why doesn't it exist? If you look back at the program, you'll see that line 37 of `main`, which is the call to `XtAddCallback`, is placed *before* the line which creates the widget `outer_box`. In effect, the program is trying to add a callback to a widget at a time when that widget does not exist. The cure is to simply move the line that calls `XtAddCallback` to a point after the widget `outer_box` has been created.

This example should give you an idea of the power of cdb. You should become familiar with cdb (or one of the other debuggers), as it is an indispensable tool for debugging widget programs.

10.2.2 A Trace Routine

You can include a built-in trace routine in your program to help isolate bugs. This routine will not allow you to perform the sophisticated maneuvers that you can with cdb, but it can point you in the right direction. A later section shows how to use cdb "breakpoints" that halt the program execution at points you specify.

The trace routine consists of a procedure to get command line options and, if the proper option is set, turn tracing on. If tracing is on, certain actions, such as printing variable values, is taken at various points in the program. The following program is our `showdebug` program from the previous section, but now it includes the trace features. We'll call this program `tracebug`.

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/X.h>
#include <X11/Intrinsic.h>
#include <X11/Core.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/List.h>

#define TRACE(string, var) if (trace) {printf(string, var);}

Widget  toplevel;
Widget outer_box;

Arg myArgs[20];

static void  GetOptions(argc, argv, trace)
  int          argc;
  char       **argv;
  Boolean     *trace;
{
  register int i;

  for (i = 1; i < argc; i++) {
    if (argv[i][0] == '-' && argv[i][1] == 'v')
      *trace = True;
    else {
      fprintf(stderr, "\nusage:  %s [-v]\n", "tracebug.c");
      fprintf(stderr, "The -v option enables trace output.\n\n");
      exit(1);
    }
```

```
    }
}

void  SelectProc(w,closure,call_data)
  Widget w;
  caddr_t  closure, call_data;
{

  printf("Single Selection.\n");

}

void main(argc, argv)
  unsigned int argc;
  char **argv;
{
  int i;
  Boolean trace = False;

  toplevel = XtInitialize(argv[0], "DBtest", NULL, 0, &argc, argv);

  GetOptions(argc, argv, &trace);

  TRACE("First trace: toplevel = %d \n.", toplevel);
  i = 0;
  XtSetArg(myArgs[i], XmNselectedItemCount, (XtArgVal) 10); i++;
  XtSetArg(myArgs[i], XmNvisibleItemCount, (XtArgVal) 5); i++;
  TRACE("Second trace: i = %d \n.", i);
  XtAddCallback(outer_box, XmNsingleSelectionCallback,
    SelectProc);
  outer_box = XmCreateScrolledList(toplevel, "ListWidget",
    myArgs, i);
  TRACE("Created outer_box: outer_box = %d.\n", outer_box);
  XtManageChild(outer_box);

  XtRealizeWidget(toplevel);

  XtMainLoop();

}
```

The key to the trace process is the function `GetOptions`. This function parses the command line that executed the program. If an option "-v" is present, tracing is turned on;

otherwise, it is off. Note that the macro TRACE is defined to have two arguments and provides the means to print messages and variable values. There are three TRACE statements in the program. These can be placed anywhere you need some information. You could just as easily put in "printf" statements that would accomplish the same thing, but they would always be executed.

It is not necessary to compile the program with the "-g" option, although to do so will cause no harm. Run the program with the "-v" option set to turn tracing on. The command to do this is `tracebug -v`. The program prints the information specified at each TRACE occurrence, up to the time it aborts. As you can see from figure 10-4, this process is not as informative as cdb.

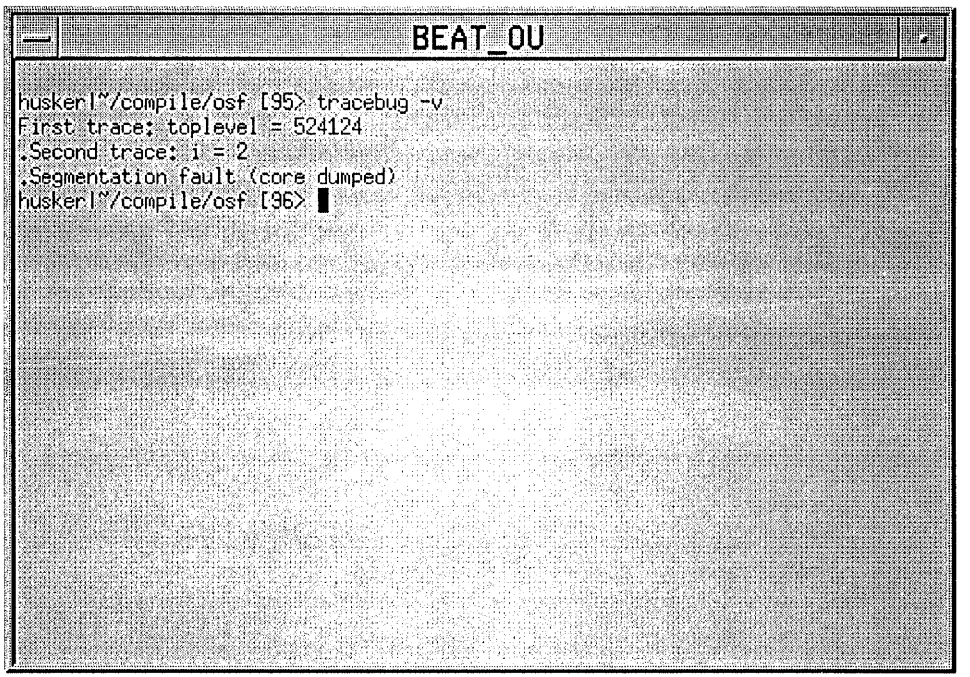


Figure 10-4. Trace Process Output

You know that the Segmentation Fault occurred somewhere between the last TRACE to be printed and the first one *not* to be printed (which, in this case, is the *only* one not to be printed). To further isolate the problem, you would have to insert more TRACE statements and recompile the program.

Notice that the first TRACE statement prints the value of the widget toplevel. Since toplevel is defined to be type Widget, it is actually a pointer to a structure. Printing this value does not really tell us much, except that it is not NULL. Be wary of printing

information from variables that are pointers.

One word of caution when using this process: Be sure that the call to `GetOptions` occurs *after* the initialization takes place. We used `XtInitialize` in this program and the call to `GetOptions` occurs after it is called. The reason for this is that very often some command line parameters used by `XtInitialize` are present and would not be recognized by `GetOptions` and could cause an early exit from the program via the "usage" path of `GetOptions`.

10.3 Other Errors

In some cases you will see some kind of error or warning message that attempts to convey the cause of the problem. In other cases you may not see any error message at all, yet the program does not run.

10.3.1 An X Toolkit Error

Sometimes the system will detect an error and output a message to that effect before aborting the program. For example, the program in this section is supposed to create a window that displays four ArrowButton widgets. Unfortunately, the program has several bugs, the first of which causes the error message

```
X Toolkit Error: XtCreateWidget requires non-NULL parent
```

to appear, after which the program terminates.

The program source code is shown below.

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/RowColumn.h>
#include <Xm/ArrowB.h>
#include <Xm/Frame.h>

#define TRACE(string, var) if (trace) {printf(string, var);}

static void  GetOptions(argc, argv, trace)
  int          argc;
  char       **argv;
  Boolean     *trace;
```

```
{
  register int i;

  for (i = 1; i < argc; i++) {
    if (argv[i][0] == '-' && argv[i][1] == 'v')
      *trace = True;
    else {
      fprintf(stderr, "0sage:  %s [-v]0, "debugdemo.c");
      fprintf(stderr, "The -v option enables trace output.0);
      exit(1);
     }
  }
}

void main (argc,argv)
unsigned intargc;
char     **argv;
{
   Widgetapp_shell;/*  ApplicationShell */
   Widgetframe1;/*  Frame*/
   Widgetframe2;/*  Frame*/
   Widgetrow_column;/*  RowColumn */
   Widgetab1;/*  ArrowButton*/
   Widgetab2;/*  ArrowButton*/
   Widgetab3;/*  ArrowButton*/
   Widgetab4;/*  ArrowButton*/

   Display*display;

   Arg   al[10];/*  arg list*/
   register intac;/*  arg count*/
   register inti;/*  counter*/
   Boolean  trace = False;


   /* Initialize toolkit and open display.
   */
   XtToolkitInitialize ();
   display = XtOpenDisplay (NULL, NULL, argv[0], "XMdemos",
         NULL, 0, &argc, argv);
   if (!display)
   {
      XtWarning ("debugdemo: can't open display, exiting...");
```

```
    exit (0);
}


/* Create ApplicationShell.
*/
app_shell = XtAppCreateShell (argv[0], "XMdemos",
      applicationShellWidgetClass, display, NULL, 0);

GetOptions(argc, argv, &trace);

/* Create RowColumn and ArrowButtons
*/
ac = 0;
XtSetArg (al[ac], XmNshadowThickness, 3);  ac++;
frame1 = XmCreateFrame (app_shell, "frame", al, ac);
XtManageChild (frame1);

ac = 0;
XtSetArg (al[ac], XmNpacking, XmPACK_COLUMN);  ac++;
XtSetArg (al[ac], XmNnumColumns, 2);  ac++;
XtSetArg (al[ac], XmNspacing, 20);  ac++;
XtSetArg (al[ac], XmNmarginHeight, 20);  ac++;
XtSetArg (al[ac], XmNmarginWidth, 20);  ac++;
row_column = XmCreateRowColumn (frame2, "row_column", al, ac);
XtManageChild (row_column);

ac = 0;
XtSetArg (al[ac], XmNarrowDirection, XmARROW_UP);  ac++;
ab1 = XmCreateArrowButton (row_column, "ab1", al, ac);

XtManageChild (ab1);

ac = 0;
XtSetArg (al[ac], XmNarrowDirection, XmARROW_DOWN);  ac++;
ab2 = XmCreateArrowButton (row_column, "ab1", al, ac);

XtManageChild (ab2);

ac = 0;
XtSetArg (al[ac], XmNarrowDirection, XmARROW_RIGHT); ac++;
ab3 = XmCreateArrowButton (row_column, "ab1", al, ac);
```

```
    XtManageChild (ab3);

    ac = 0;
    XtSetArg (al[ac], XmNarrowDirection, XmARROW_LEFT);  ac++;
    ab4 = XmCreateArrowButton (row_column, "ab1", al, ac);

    XtManageChild (ab4);

    XtMainLoop();
}
```

When you have the program properly compiled (we'll assume you've named this one debugdemo), run it and you'll see that it does indeed come to an abnormal termination with the X Toolkit error message shown earlier. What exactly does the error message mean? First of all, there is no call to XtCreateWidget in the program, so where does it come from? Recall from earlier chapters that convenience functions such as XmCreateFrame, XmCreateRowColumn, and XmCreateArrowButton create unmanaged widgets. They do so by calling XtCreateWidget, so our problem must be involved with creating a widget, but which one? You can use cdb or the trace routine to find out.

Using cdb

You can set "breakpoints" anywhere in the program from cdb. A breakpoint is simply a point you specify at which execution of the program is temporarily halted. You resume execution of the program at any time. While the program is halted you can examine the value of program variables or execute certain breakpoint commands. To set breakpoints in the program, you must first run the program under cdb by using the command

```
cdb debugdemo
```

You'll see a window like that shown in figure 10-5.

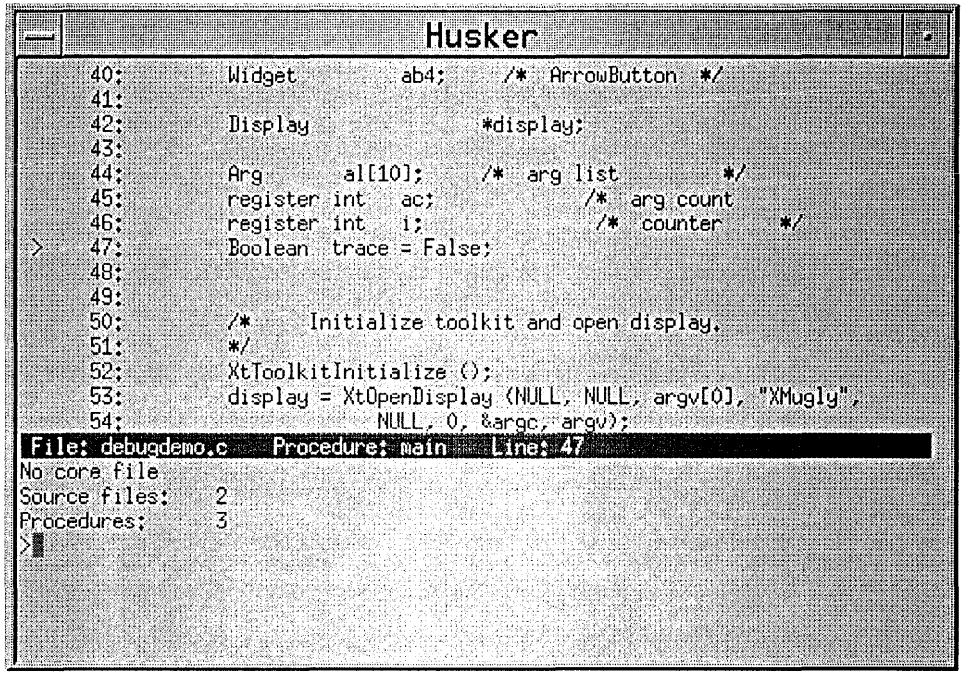


Figure 10-5. Running debugdemo With cdb

As you know from previous experience with cdb, the program debugdemo is not yet running because you have not entered "r" in the lower window. Before you do so, you must decide where to set the breakpoints. You know that the problem causing the X Toolkit error is associated with the creation of some widget, so placing breakpoints at those lines seems like a reasonable start. Notice that in the upper window of figure 10-5 there is an indicator (a ">") to the left of line 47. Since you want a breakpoint set at a line that creates a widget, you need to move the indicator. Enter "w" (for "window") and press Return to move the indicator in the upper window. The indicator is now at line 61, which is still not at a line that creates a widget. Enter "w" and press Return again. You should see the window in figure 10-6.

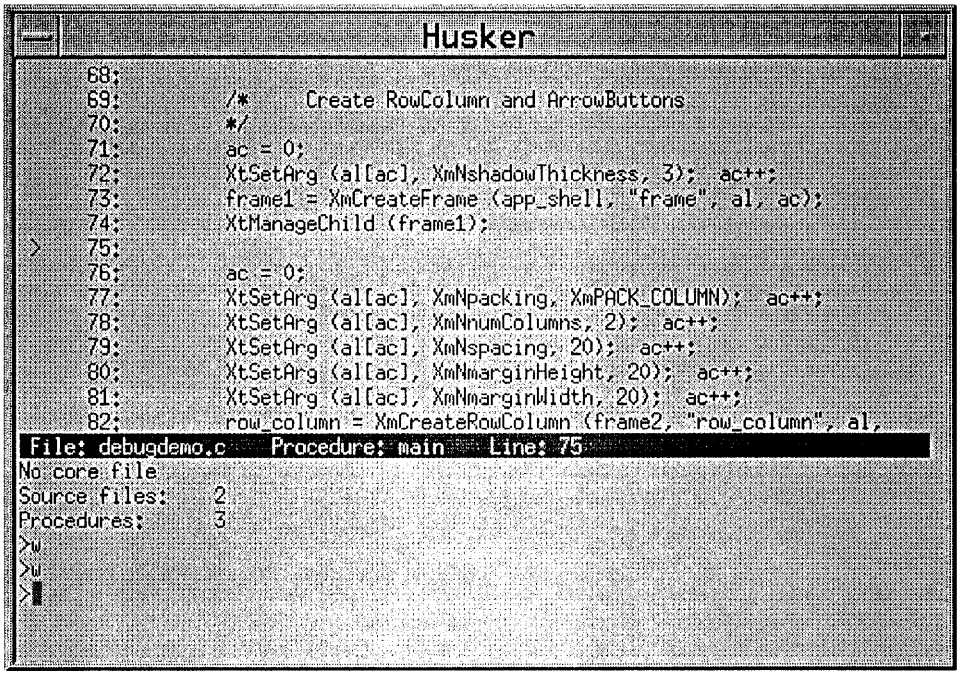


Figure 10-6. Using the cdb Window Command

Line 74 is immediately after a statement that creates a widget, so to set a breakpoint there, move the indicator to that line by entering "74" and pressing Return. You could have specified "74" earlier instead of using the "w" command if you had known the line number. Now with the indicator on line 74, enter "b" and press Return. You should see the window shown in figure 10-7.

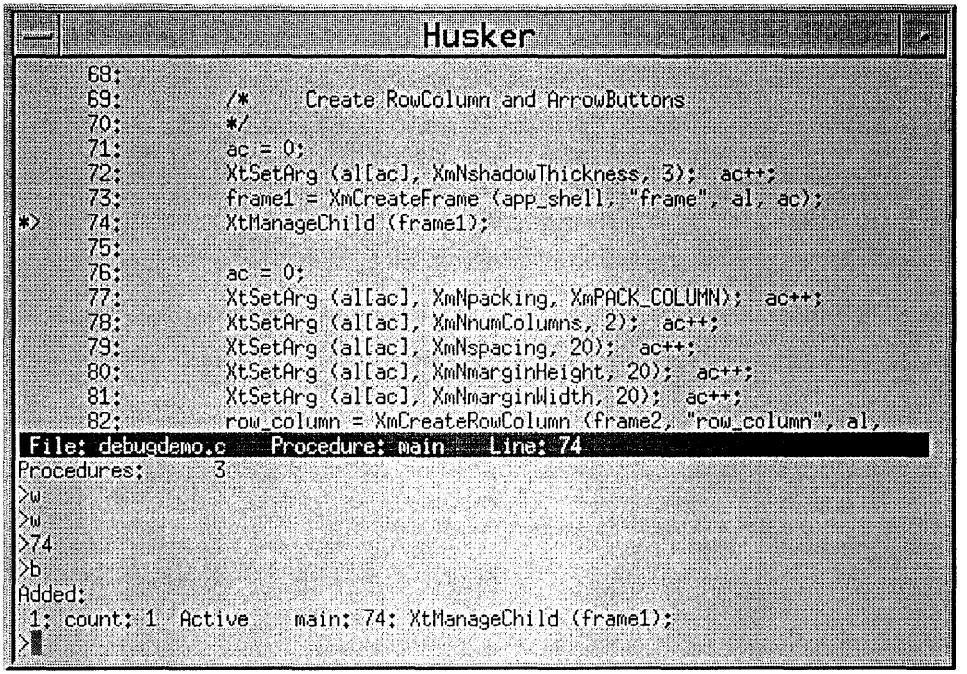


Figure 10-7. Setting a Breakpoint in cdb

An asterisk appears next to the indicator, indicating that this line is a breakpoint. Using this procedure, set breakpoints at the line following each widget creation. One thing to remember about a breakpoint is that it must be placed on a line that contains an executable statement. When you have completed setting the breakpoints, run debugdemo by entering "r" and pressing Return. The program should stop at the first breakpoint (line 74). If it does, the widget `frame1` was created without error, thus eliminating it as the source of the problem. Continue the program by entering "c" and pressing Return. The error message appears instead of the program halting at the second breakpoint. This indicates that the problem is occurring when the RowColumn widget is being created at line 83. The error message mentions that "XtCreateWidget requires non-NULL parent," implying that the parent specified in the `XtCreateRowColumn` function call is NULL. If you examine line 83, you'll see that the parent specified is a widget called `frame2`. Since `frame2` is about to be a parent, it should have been created previously. Looking back to line 74, the widget created there is called `frame1`. Since no other widgets have been created, `frame2` appears to be a typographical error and should be `frame1` instead. Make this change and recompile the program.

Using the Trace Routine

Notice that we've included the trace routine in the program. You could insert TRACE statements at the same places in the program that you set breakpoints. The TRACE statements could be something like that shown below:

```
TRACE("First trace: frame1 = %d \n", frame1);
```

Subsequent statements should have the appropriate widget name instead of `frame1`. By running the program with the `-v` option, you enable the trace action. The trace printouts that result provide you the clues needed to solve the problem.

10.3.2 No Error Message

After you recompile the program after changing `frame2` to `frame1`, run it again. This time nothing seems to happen. You see no error messages, yet no window is displayed on the screen. The program seems to be "hung." By setting breakpoints or placing TRACE statements in the program, you can follow its progress. For example, if you place breakpoints after each widget creation statement and the run the program, you'll see that it runs fine until some point after the creation of the fourth ArrowButton, `ab4`. There are only two statements following the creation of `ab4`, `XtManageChild (ab4)` and `XtMainLoop()`. Didn't chapter 3 mention something about "making the widget visible?" Right! We inadvertently left out the call to `XtRealizeWidget`. Add the statement

```
XtRealizeWidget();
```

to the program just before the call to `XtMainLoop` and recompile the program. Then run it and you should see the window shown in figure 10-8.

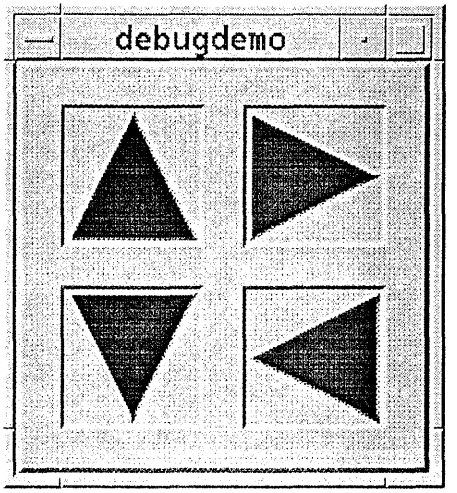


Figure 10-8. Program debugdemo Window

We've explained a few of the potential problems that you might encounter. Obviously, we cannot treat every possible error condition. If you follow the ideas explained here, you should have little difficulty in debugging.

Glossary 11

accelerator

A keyboard key or keys used to cause some action to occur. For example, the Shift Menu keys could be used to post a menu instead of a mouse button action.

atom

A 32-bit number that represents a string value.

callback

A procedure that is called if and when certain specified conditions are met. This is accomplished by specifying the procedure in a callback list. Individual widgets can define callback lists as required.

child widget

A child widget is a subwidget of a composite widget. The composite widget is referred to as the *parent* of the child widget. The parent controls where the child is placed and when it is mapped. If the parent is destroyed, the child is automatically destroyed.

class

The general group that a widget belongs to.

Composite widget class

This class provides the resources and functionality that allows subclass widgets to manage the layout and children.

Composite Manager

A composite manager is a manager widget with special knowledge about the handling of one or more particular widgets. For example, a TitleBar and ScrollBar can be registered with a Panel widget, and the Panel widget will position the TitleBar and ScrollBar widgets correctly. Normally, a Manager widget has no knowledge about its children.

constraint

Resources that certain manager widgets can impose on their children are called *Constraint* resources. For example, if a PanedWindow widget wants its children to be a certain size, it can specify the size by using the resources `XtNmin` and `XtNmax`. The man pages will specify those manager widgets that have Constraint resources.

convenience dialog

A widget or collection of widgets created by a Dialog convenience function.

convenience function

A convenience function is a function that creates certain combinations of widgets, including the necessary Shell widget.

Core

Core is the basic class from which all widgets are built. It acts as a superclass for other widget classes and provides resources that are required by all widgets.

Dialog

A collection of widgets, including a DialogShell, a BulletinBoard (or a subclass of BulletinBoard or some other container widget), plus various children of BulletinBoard such as Label, PushButton, and Text widgets. Dialogs are used as an interface between the application and its user.

double click

A method of selection in which a mouse button is pressed and released twice in rapid succession.

drag

A type of interaction in which a mouse button is pressed and held. The mouse is moved so that the pointer is "dragged" to the desired point and the button is then released to complete the action.

grab

A procedure by which a window will act upon a key or button event that occurs for it or any of its descendents. This precludes the necessity of setting up translations for all windows.

instantiate

To represent an abstraction by a concrete instance. To instantiate a widget means that a widget class creates an instance of that class.

intern

The procedure used to define an atom.

manager class

A class that provides the resources and functionality to implement certain features, such as keyboard interface and traversal mechanism. It is built from core, composite, and constraint classes.

meta class

A meta class is a set of structures and functionality that a widget uses to export that functionality to subclass widgets. Each instance of a widget subclass will have the features common to that widget class and will export these features to child widgets of that class. Included in this class are Core, Composite, Constraint,

Primitive, Button, Manager, MenuMgr, and MenuPane. A meta class widget is never instantiated.

modal Dialog
A Dialog that interrupts the work session to solicit input from the user.

modeless Dialog
A Dialog that solicits input from the user but does *not* interrupt the work session.

persistence
Persistence means that a specified character set is used for all subsequent text segments in a compound string until a new character set is encountered.

popup
A type of widget that appears as the result of some user action (usually clicking a mouse button) and then disappears when the action is completed.

post
The action required to make a Popup or Pulldown menu appear. This action is normally a click or button press on one of the mouse buttons.

property
Public information (that is, information that is available to any client) associated with a window.

protocol
A mutually agreed-upon mechanism for communicating between clients to accomplish certain actions.

scroll region
The rectangular portion of a ScrollBar that contains the two arrows and the slider.

subclass
A class of widgets that inherits resources from a higher class.

translations
Action procedures that are invoked for an event or sequence of events.

Primitive
The Primitive class provides the resources and functionality for the low-level widgets that are managed by the manager class. Primitive class widgets cannot have normal child widgets but they can have popup child widgets.

widget
A widget is a graphic device capable of receiving input from the keyboard and the mouse and communicating with an application or another widget by means of a callback. Every widget is a member of only one class and always has a window associated with it.

widget instance

The creation of a widget so that it is seen on the display. Note that some widgets (meta class, for example) cannot be instantiated.

widget tree

A widget tree is a hierarchy of widgets within a specific program. Examples of widget trees can be found in chapter 3. The shell widget is the root of the widget tree. Widgets with no children of any kind are leaves of the tree.

Index

A

B

C

D

E

F

G

Q

R

S

T

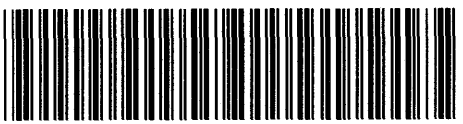
U

V

W

X

HP Part Number
98794-90005
Microfiche No. 98794-99005
Printed in U.S.A. E0989



For Internal Use Only