Programming With the HP X Widgets

Version 11

HP 9000 Series 300/800 Computers

HP Part Number 98794-90000

Hewlett-Packard Company
1000 NE Circle Boulevard, Corvallis, Oregon 97330-9988

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.

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.

July 1988...Release 1.

December 1988...Release 2.

May 1989...Update. This Update replaces the "Programming With the Xt Intrinsics" section.

June 1989...Release 3. This edition incorporates the May 1989 Update.

Contents

1	**Introduction**	1-1
1.1	Widget Classes	1-2
1.2	Terminology	1-2
1.3	Compiling Sample Programs	1-5
1.4	Available Documentation	1-6
2	**Widgets**	2-1
2.1	Display Widgets	2-4
2.2	Editing Widgets	2-4
2.3	User Selection Widgets	2-6
2.4	Layout Widgets	2-7
2.5	Menu Widgets	2-13
2.6	Miscellaneous Widgets	2-14
2.7	Utility Functions	2-15
3	**Using Widgets in Programs**	3-1
3.1	Including Header Files	3-4
3.2	Initializing the Toolkit	3-5
3.3	Setting Up Argument Lists for Widgets	3-6
3.4	Adding Callback Procedures	3-7
3.4.1	Writing a Callback Procedure	3-8
3.4.2	Adding Callbacks	3-9
3.4.3	Setting Widgets' Callback Resources	3-10
3.5	Creating the Widget	3-10
3.6	Making the Widget Visible	3-11
3.7	Linking in Libraries	3-12
3.8	Creating Defaults Files	3-12
3.8.1	Application Defaults Files	3-13
3.8.2	User Defaults Files	3-14
3.8.3	Defaults File Example	3-14
3.9	Using Color	3-14
3.9.1	Capabilities and Attributes	3-15
3.9.2	Using the Capabilities	3-16
3.9.3	Example Resource Values	3-17
3.10	Advanced Programming Techniques	3-18
3.10.1	Setting Argument Values	3-18
3.10.2	Manipulating Created Widgets	3-20
3.11	An Advanced Sample Program	3-21

3.11.1 Description 3-21
3.11.2 Widget Hierarchy 3-22
3.11.3 Setting Arguments 3-24
3.11.4 Writing the Callback Procedures 3-24
3.11.5 Source Code 3-25
3.12 Another Advanced Sample Program 3-31
3.12.1 Windows Used in xfonts 3-33
3.12.2 Widget Hierarchy 3-35
3.12.3 Source Code 3-37

4 **Menus** 4-1
4.1 Menu System Description 4-1
4.1.1 Menu Hierarchy 4-1
4.1.2 Menu Manager Views 4-2
4.1.3 Data Specification 4-4
4.2 Menu Components 4-5
4.2.1 Menu Manager 4-5
4.2.2 Menu Pane Widget 4-8
4.2.3 MenuButton Widget 4-8
4.2.4 MenuSep Widget 4-9
4.3 Creating a Menu 4-9
4.4 Using Menus 4-10
4.4.1 Callbacks 4-11
4.4.2 Keyboard Interface 4-11
4.5 Mixing Menu Accelerators and Traversal 4-11
4.6 A Sample Program 4-13

5 **Form Widgets** 5-1
5.1 Using the Form Widget 5-1
5.2 Summary 5-4

6 **Keyboard Interface** 6-1
6.1 Keyboard Input Processing 6-1
6.2 Keyboard Traversal 6-2
6.2.1 Visual Attributes 6-2
6.2.2 Application Control 6-2
6.3 Internal Implementation Requirements for Traversal 6-6
6.3.1 Primitive Widget Requirements 6-6
6.3.2 Manager Widget Requirements 6-6

7 **Writing New Widgets** 7-1
7.1 Widget Description 7-1
7.2 Constructing a Widget 7-2
7.2.1 The Private Header File 7-2
7.2.2 The Public Header File 7-6
7.2.3 The Source Code File 7-6
7.2.4 Source Code 7-13
7.2.5 Putting the New Widget Together 7-31
7.3 Widget Classing 7-33
7.3.1 Implementing Widget Classing 7-34
7.3.2 XwTasks 7-35
7.3.3 Using Resources 7-38
7.4 Summary 7-40

This page left blank intentionally.

Introduction 1

The HP X Widget system provides the base upon which you, the programmer, can build a wide variety of application environments. It is based on the X Toolkit Intrinsics, a set of functions and procedures that provide quick and easy access to the lower levels of the X Window system.

You can see from figure 1-1 that the HP X Widget system is layered on top of the X Toolkit Intrinsics, which in turn are layered on top of the X Window System, thus extending the basic abstractions provided by X. The HP X Widget system supports independent development of new or extended widgets.

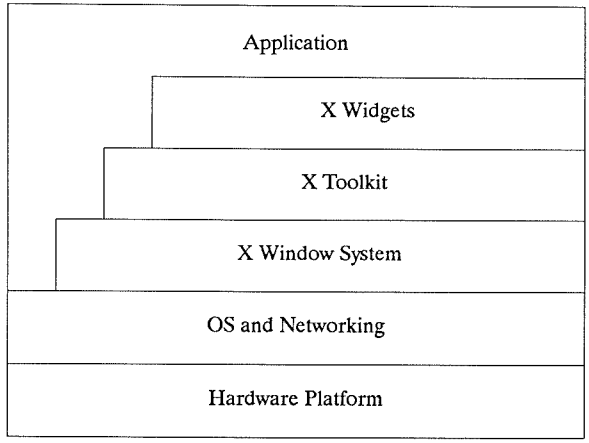


Figure 1-1. User Interface Development Model

The HP X Widget system 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 less lines of code using the HP X Widgets; however, they will require more memory than similar applications written without using the HP X Widgets.

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

1.1 Widget Classes

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.

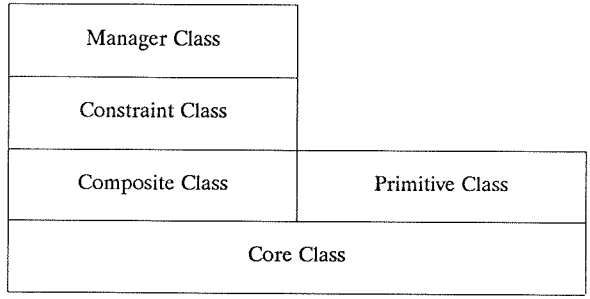


Figure 1-2. Widget Class Hierarchy

The basic class is the core class. It contains resources that are inherited by all other classes. Two classes are layered on top of the core class, the composite class and the primitive class. The primitive class has no other classes above it, but the composite class has two, the constraint class and the manager class. Each higher class can inherit some or all of the resources belonging to the lower class. For example, this means that 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 Reference Information section of this manual.

1.2 Terminology

This section defines selected words and terms used in this manual.

- **Accelerator.** A keyboard key or keys used to cause some action to occur. For example, the Shift Menu keys might be used to post a menu instead of a mouse button action.
- **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 Widget.** A composite manager widget 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 VPanedWindow 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.
- **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.
- **Double Click.** A method of selection in which a mouse button is pressed and released twice in rapid succession.
- **Drag.** A method of menu selection in which a 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 select the action to be taken.
- **Grab.** A procedure by which a window will act upon a key or button event that occurs for it or any of its descendants. 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.
- **Manager Widget 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.
- **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 double click on one of the mouse buttons.
- **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. See *Programming With the Xt Intrinsics* for more information.
- **Primitive Widget Class.** 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. For example, if a program included Panel, Form, RowCol, and three instances of PushButton widgets, its widget tree would be as shown in figure 1-3.

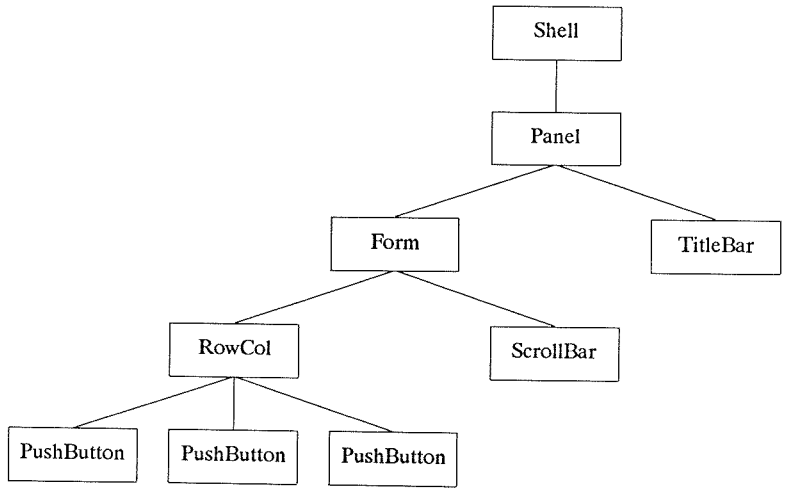


Figure 1-3. A Widget Tree

The shell widget returned by `XtInitialize` or `XtCreateApplicationShell` is the root of the widget tree. Widgets with no children of any kind are leaves of the tree.

1.3 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/Xw`. 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/Xw` to your work directory. *Do not* attempt to compile the program in the `/usr/contrib/Xw` directory.
2. Compile the program by executing the following command:

   ```
   Make <programname>
   ```

3. If there is a defaults file commented into the beginning of the source code, move that defaults file to the directory `/usr/lib/X11/app-defaults` before you run the program.

1.4 Available Documentation

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

TABLE 1-1. Documentation Map

Task	Document Title	Document Part No.
Configure the X Window System	*Configuring X on the Series 300*	98594-90025
Learn how to start the X Window System	*Using the X Window System*	98594-90040
Customize the X Window Environment	*Using the X Window System*	98594-90040
Incorporate widgets into applications	*Programming With the HP X Widgets and the Xt Intrinsics*	98794-90000
Write new widgets	*Programming With the HP X Widgets and the Xt Intrinsics*	98794-90000
Learn about Fortran bindings	*Programming With the Xrlib User Interface Toolbox*	5090-0004
Learn about National Language I/O System	*Programming With the Xrlib User Interface Toolbox*	5090-0004
Write graphics programs	*Programming With Xlib*	98794-90010

Other sources of information are listed below:

- A "readme" file named `x11windows` found in the directory `/etc/newconfig/Update_info`. This file contains last-minute information about the X Window system.
- *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.

This page left blank intentionally.

Widgets 2

The HP X Widgets library contains a variety of widgets, each designed for a different task. A widget is a single part of a group of components that comprise a predefined set. Widgets are used 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.

A widget instance is composed of a data structure that contains 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 is 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 `XtCreateWidget` or `XtCreateManagedWidget`. Refer to chapter 3, "Using Widgets in Programs," for specific examples of creating widgets.

There are also some functions to aid you in using the widgets. This chapter provides an overview of what is available. The man pages at the back of this manual contain details for each of the widgets. Figure 2-1 shows how widgets might be combined in an application.

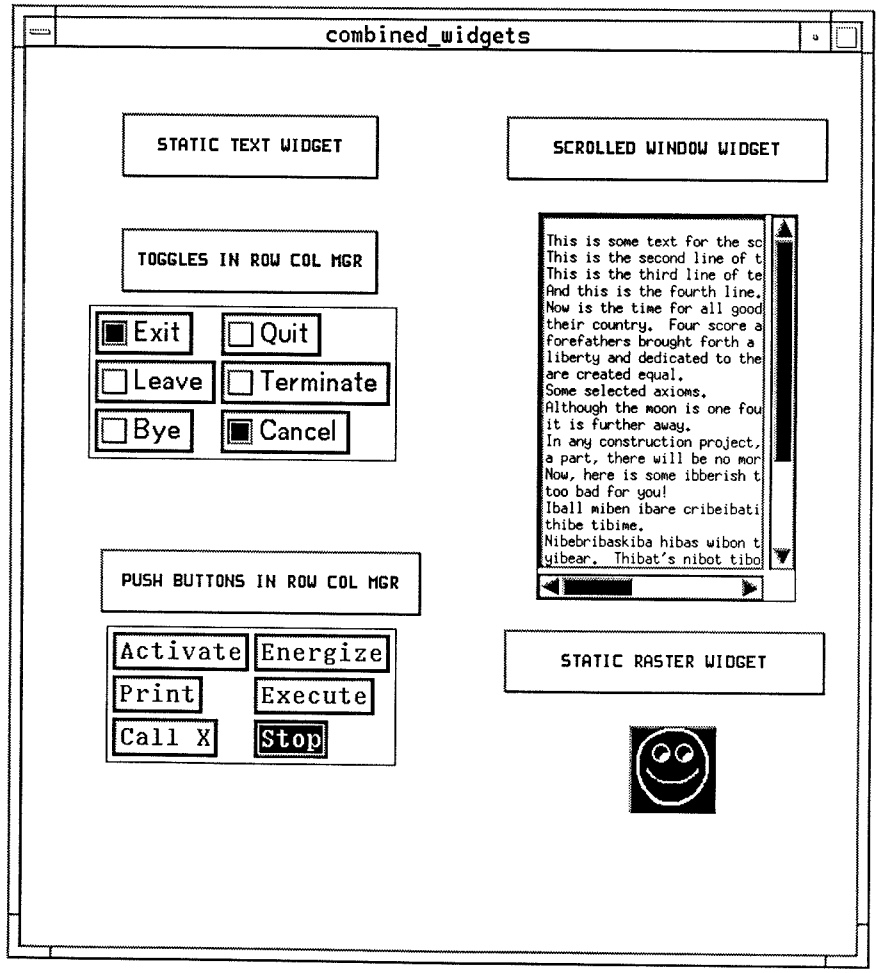


Figure 2-1. Widget Application Screen

Several types of widgets are shown in figure 2-1. The large box that contains the rest of the widgets is a BulletinBoard widget. In the upper left-hand corner is a simple StaticText widget. Below that is another StaticText widget, and below that is a set of Toggle widgets that are managed by a RowCol widget. Note that two of the Toggles, "Exit" and "Cancel" are "selected" as indicated by the black square. Below the Toggles is another StaticText widget, and below that a set of PushButton widgets that are also managed by a RowCol widget. The "Stop" PushButton is highlighted to show that it is selected. At the upper right is another StaticText widget, and below that is a ScrolledWindow widget with some text in it. Note the ScrollBars at the right and bottom of the ScrolledWindow widget.

Finally, there is yet another StaticText widget, and below that a StaticRaster widget. This example does not show all of the available widgets. It does show how you can combine several types of widgets within the same window.

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

TABLE 2-1. Categories of Widgets

Class Name	Widget Class
Display Widgets	
StaticRaster	`XwstaticRasterWidgetClass`
StaticText	`XwstaticTextWidgetClass`
Editing Widgets	
ImageEdit	`XwimageEditWidgetClass`
TextEdit	`XwtextEditWidgetClass`
Selection Widgets	
PushButton	`XwpushButtonWidgetClass`
Toggle	`XwtoggleWidgetClass`
Layout Widgets	
BulletinBoard	`XwbulletinWidgetClass`
Form	`XwformWidgetClass`
List	`XwlistWidgetClass`
Panel	`XwpanelWidgetClass`
RowCol	`XwrowColWidgetClass`
ScrolledWindow	`XwscrolledWindowWidgetClass`
VPanedWindow	`XwvPanedWidgetClass`
Menu Widgets	
Cascade	`XwcascadeWidgetClass`
MenuButton	`XwmenuButtonWidgetClass`
MenuSep	`XwmenuSepWidgetClass`
PopupMgr	`XwpopupMgrWidgetClass`
Pulldown	`XwpulldownWidgetClass`
Miscellaneous Widgets	
Arrow	`XwarrowWidgetClass`
Frame	`XwframeWidgetClass`
Sash	`XwsashWidgetClass`
ScrollBar	`XwscrollBarWidgetClass`
TitleBar	`XwtitleBarWidgetClass`
Valuator	`XwvaluatorWidgetClass`

2.1 Display Widgets

Display widgets normally do not provide any interaction. They only display data on the screen.

StaticRaster (`XwstaticRasterWidgetClass`)
This widget will display a picture (raster image). You can indicate selection of the widget by moving the mouse cursor over the widget and clicking mouse button 1. The default window size will fit the dimensions of the raster image. If you specify a larger one, the image is centered. For a smaller window, the image is clipped along the right and bottom sides, as required.

Figure 2-2. StaticRaster Widget

StaticText (`XwstaticTextWidgetClass`)
Use this widget to display short messages to users. The widget will create a window for the text. If you supply your own window dimensions, you can cause the text to be centered in a larger window, or wrapped in a smaller one.

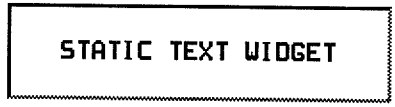


Figure 2-3. StaticText Widget

2.2 Editing Widgets

The widgets in this category allow the user to modify the data being displayed.

ImageEdit (`XwimageEditWidgetClass`)

The ImageEdit widget allows an image to be displayed in an enlarged version so that it can be edited on a pixel-by-pixel basis. To edit the image, move the mouse pointer to the desired point and press mouse button 1. The pixel under the pointer will change to the foreground color. If the button is held down while the pointer is moved, all pixels that are touched will change to the foreground color. Repeating this procedure will cause the pixels to revert to their original color.

TextEdit (`XwtextEditWidgetClass`)

The TextEdit widget provides a single or multi-line 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, multi-page document viewing, and full-screen editing.

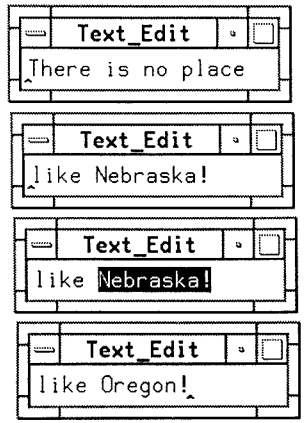


Figure 2-4. TextEdit Widgets

Figure 2-4 shows four TextEdit widgets with a single line. The line is too long to show all the text at once, but you can move the pointer (by using the arrow keys) to display the remainder of the text. The second widget shows the result of moving the pointer to the right. The pointer represents the "insert character" point. The third widget shows the word "Nebraska" highlighted. This was accomplished by moving the pointer to any character in the word and "double-clicking" mouse button 1. You can highlight any part of the text by positioning the pointer at the first character to be highlighted, pressing and holding mouse button 1, and "dragging" the mouse to move the pointer to the last character you want to highlight. The highlighted text can be deleted by pressing CTRL w. You can "paste" the deleted text anywhere in the text by moving the pointer to the desired location and pressing CTRL y. You can type in new text for the text that was deleted. This is shown in the fourth widget of figure 2-4. The program used to generate these widgets can be found in `/usr/lib/contrib/Xw/Text_Edit.c`. See section

1.3 in chapter 1 for instructions to compile this program.

2.3 User Selection Widgets

Use these widgets to allow users to specify choices. Each of these widgets corresponds to a single choice. By combining several of these widgets with a manager widget (discussed later in this section), you can build forms and menus to provide a variety of input choices.

PushButton (`XwpushButtonWidgetClass`)

This widget consists of a text label surrounded by a button border. Normally, 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 colors will invert. When the mouse button is released, the button colors will revert to the original color scheme. PushButtons are used to invoke actions, such as run, cancel, etc.

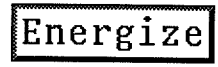


Figure 2-5. PushButton

Toggle (`XwtoggleWidgetClass`)

This widget consists of a rectangle with a small box and a label contained inside the rectangle. Normally, you select the Toggle by placing the mouse cursor inside the rectangle and pressing mouse button 1. The interior of the box (not the rectangle) will then be filled with the selection color. The selection color is the foreground color by default, but this can be changed using a default file. Toggles are normally used for binary (on-off) state applications.

Figure 2-6. Toggle

2.4 Layout Widgets

These widgets make it easier to combine several widgets into one combination widget. They can be used as a "backdrop" on which other widgets can be placed. The layout widget is the *parent* widget. The widgets being combined are called *child* widgets.

BulletinBoard (`XwbulletinWidgetClass`)

This is a general layout widget that will accept any number of children. The BulletinBoard widget will automatically make the BulletinBoard the appropriate size to hold the child widgets. You, the programmer, must specify the x and y coordinates of each child widget. You can also control the BulletinBoard size.

Form (`XwformWidgetClass`)

The Form widget is a constraint-based manager that provides a layout language used to establish spatial relationships between its children. It manipulates these relationships when any of the following occur:

- The Form is realized.
- New children are added to the Form.
- The children are resized, unmanaged, remanaged, or destroyed.

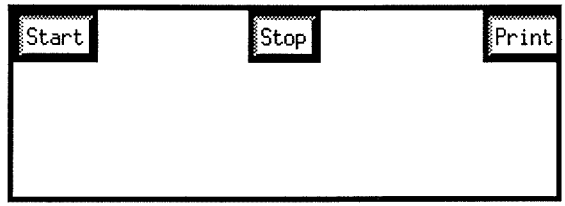


Figure 2-7. Form Widget

For example, you can specify one widget to be always next to another regardless of the size of the form window. Chapter 5, "Forms," provides a detailed description of this layout widget.

List (`XwlistWidgetClass`)

The List widget allows a set of widgets to be presented in a row-column fashion. There can be any number of columns, and by default each column will be wide enough to display the widest item in the column. The entire list window can be scrolled either horizontally or vertically, but the columns cannot be scrolled individually. The List widget also provides the capability to select one or more elements of the list, and the application can then perform some action as a result. Also, the List widget provides automatic scrolling of its window when there are too many children to display. The major differences between the List widget and the RowCol widget are listed below.

- The List widget allows you to make multiple selections from the displayed list. This is accomplished by setting the resource XtNselectionMethod to XwMULTIPLE. When in this mode, if you position the cursor on an item within the list and then press the mouse button that is bound to "Define Select" (mouse button 1 is the default), that item is selected and is highlighted. As you drag the mouse with the button held down, the original choice remains highlighted, and any other items that the cursor touches are also selected and highlighted. The RowCol does not have this feature.
- The List widget will automatically display its items within a ScrolledWindow when the list becomes too large for the given window. The RowCol widget does not do this automatically, although you could place the RowCol as a child of a ScrolledWindow and accomplish the same thing. The point is, the List widget performs this function automatically.
- The List widget, unless otherwise specified, will display all elements in a column with the same length and width. The RowCol widget will display its elements in varying sizes.
- The List widget normally will only have primitive widgets as its children. The List widget overrides any primitive semantics (behavior functions), such as highlighting on select, and uses its own. Thus, if you use a set of PushButtons within a list, the PushButton highlight-on-select feature will not work.

Figure 2-8 shows a List widget with multiple selection invoked. The items highlighted are those that were selected. Note the vertical and horizontal ScrollBars, indicating that more items are contained in the list.

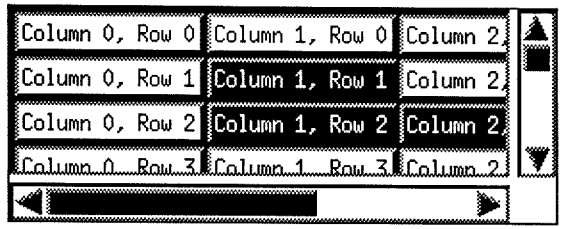


Figure 2-8. List Widget

Panel (`XwpanelWidgetClass`)

The Panel widget is used for windows with titles and menus. It may have three children, each of which is always laid out in areas known as title, menu, and workspace. Panel always lays the children out so that the child in the title area is placed at the top of the window, the child in the menu area is placed next, and the child in the workspace area is placed at the bottom. Figure 2-9 shows a Panel widget with the three areas. The title area consists of a "quit" pushbutton and four other pushbuttons used to add or remove child widgets from either the menu area or the workspace area. The menu area consists of an optional number of pushbuttons, the exact number determined by how many are created from the buttons in the title area. The workspace area is similar, but in this case it is a RowCol widget specifying only one column.

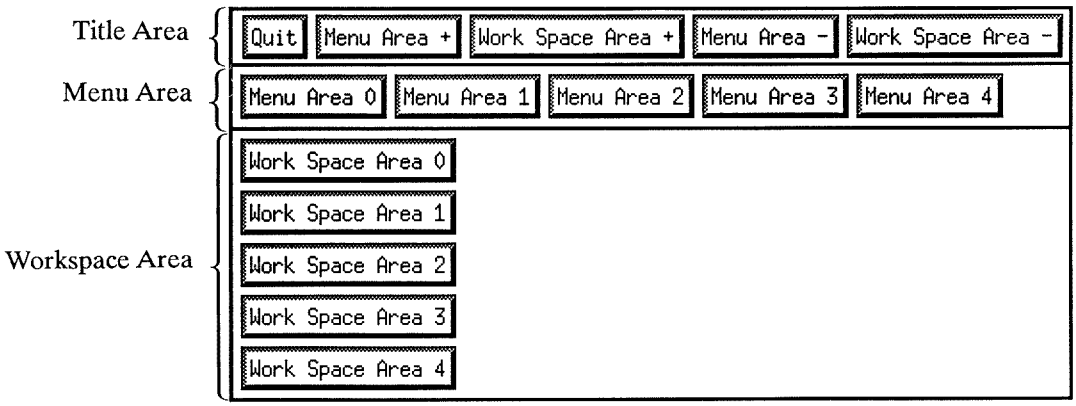


Figure 2-9. Panel Widget

When the Panel is resized, the menu area is resized but the title and work space areas are not. Figure 2-10 shows the Panel widget of figure 2-9 after resizing. Note how the widgets in the title and work space areas have been clipped, while the widgets in the menu area have been reorganized to accommodate the smaller size window.

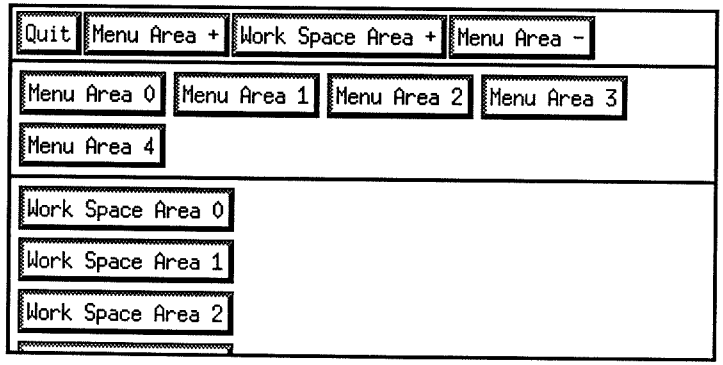


Figure 2-10. Panel Widget After Resizing

RowCol (`XwrowColWidgetClass`)

This widget controls the layout of its children in a grid of rows and columns. There are three types of row-column layouts that you can use.

- Requested columns. You specify the number of columns (the default is one) to be used in laying out the data. The children that make up the data are laid out in rows. All the columns are as wide as the widest element in the column, and all elements are left justified. Row height is determined by the largest element in the row, and all elements are centered in the row.
- Maximum columns. The RowCol widget calculates the maximum number of columns that can fit within the manager and lays the data out accordingly. You specify the manager size by setting values for the inherited resources `XtNHeight` and `XtNWidth`.
- Maximum unaligned. In this mode, no column alignment is used. Each item is placed to the right of the previous item until a row is full. The width of each row is simply the width of the RowCol widget itself. When a row is full, the next item is placed at the left edge of the manager in the next row down.

You can select any or all of the children in accordance with the children's selection processes. Alternatively, you can specify that only one widget can be selected at any time (selection policy `one_of_many`). In that case, if you select a second widget, the manager will unselect the child that was selected earlier.

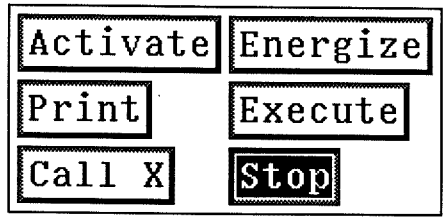


Figure 2-11. Row Column Manager

ScrolledWindow (`XwscrolledWindowWidgetClass`)

This widget manages one and only one child widget. If the manager is sized so that the entire child is not visible, it will display vertical and horizontal scroll bars to allow the entire child to be displayed. Figure 2-12 shows a ScrolledWindow with some text in it. Note that the horizontal scroll bar is positioned to the left, indicating that horizontal scrolling is possible.

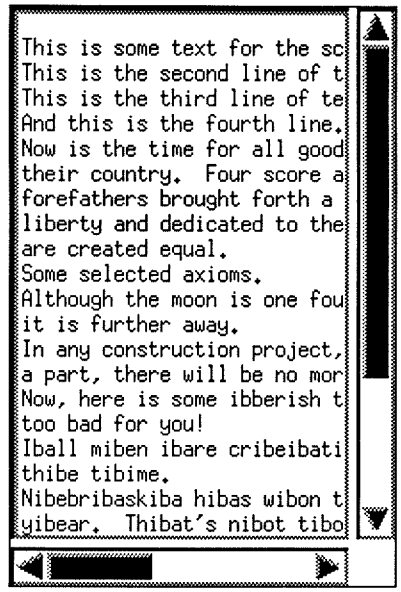


Figure 2-12. ScrolledWindow

Figure 2-13 shows the same window after scrolling to the right. Compare the positions of the horizontal scroll bar and the text with those of figure 2-12.

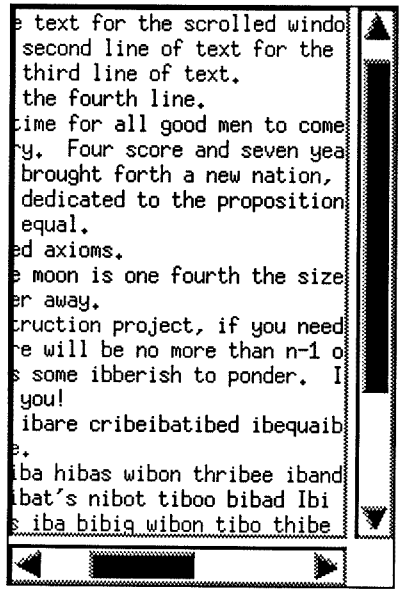


Figure 2-13. ScrolledWindow

VPanedWindow (`XwvPanedWidgetClass`)

This widget arranges its children vertically. Each child is placed in its own pane. A Sash (usually a small square box) appears at the bottom of each pane except the lowest pane. This allows the pane to be resized vertically. Figure 2-14 shows an example of a VPanedWindow widget with a number of children. Figure 2-15 shows the same widget after a pane has been resized.

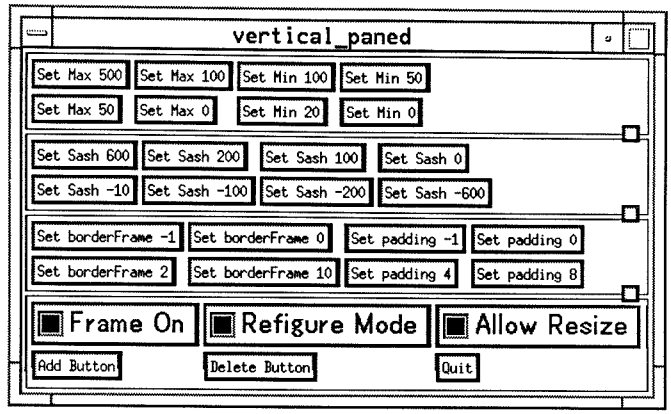


Figure 2-14. VPanedWindow Widget

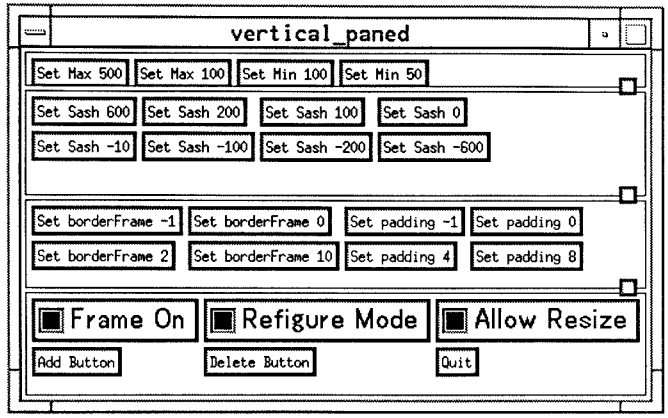


Figure 2-15. VPanedWindow Widget After Pane Resizing

2.5 Menu Widgets

Menu widgets combine to form menus that are then attached to another widget. Refer to chapter 4, "Menus," or the individual widget man pages for more information.

Cascade (XwcascadeWidgetClass)

The Cascade widget is a composite widget that may be used by an application to create a set of cascading menus. This widget always displays its managed children in a single column, and it always attempts to size itself to the smallest size possible, as determined by its children.

MenuButton (XwmenuButtonWidgetClass)

The MenuButton is commonly used with MenuMgr and MenuPane widgets to build a menu system. It consists of a single label, a mark, and a cascade indicator. It can be used by itself as a button widget.

MenuSep (XwmenuSepWidgetClass)

The MenuSep widget is a Primitive widget used to separate items or groups of items in a menu. Several different types of lines (solid, dashed, double, etc.) are available.

PopupMgr (XwpopupMgrWidgetClass)

The PopupMgr widget is a Composite widget used by an application to manage a collection of menu panes that form a Popup menu.

Pulldown (XwpulldownWidgetClass)

The Pulldown menu manager widget is a Composite widget used by an application to manage a collection of MenuPanes that form a Pulldown menu.

2.6 Miscellaneous Widgets

These widgets are helpful in designing special windows for your application. They form a class hierarchy, and each class in the hierarchy is available, as are some special routines.

Arrow (XwarrowWidgetClass)

The arrow widget draws an arrow inside a box. This arrow is used in combination with the Valuator widget to create a ScrollBar widget. See figure 2-16.

Frame (XwframeWidgetClass)

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

Scroll Bar (`XwscrollBarWidgetClass`)
The ScrollBar widget combines the Valuator and Arrow widgets to draw either a horizontal or vertical ScrollBar. The ScrollBar widget is shown in figure 2-16.

Figure 2-16. Scrollbar Widget

Title Bar (`XwtitleBarWidgetClass`)
The TitleBar widget will display StaticText and other widgets. By setting priorities appropriately, you can control how this widget responds to being resized by the user.

Valuator (`XwvaluatorWidgetClass`)
The Valuator widget combines a slider within a box. This combination is used to denote the viewable portion of a widget whose contents are too large to fit in the available window.

2.7 Utility Functions

This section describes utility functions that are currently available.

CreateTile (`XwCreateTile(screen,background,foreground,tile)`)
This utility is used to create background tiling in primitive widgets. Nine tile patterns are available, as shown in figure 2-17. The program that produced this figure can be found in */usr/contrib/Xw/tiletest.c*. Instructions for compiling this program can be found in section 1.3 of chapter 1.

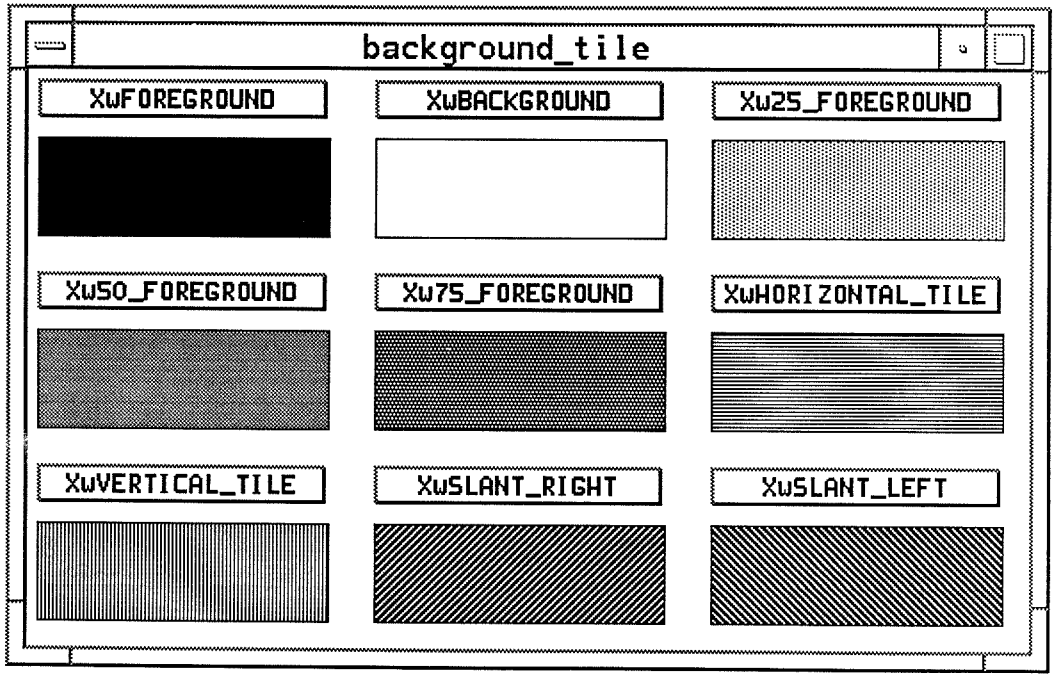


Figure 2-17. Available Background Tiles

The next chapter will show you how to use many of these widgets in an application.

Using Widgets in Programs 3

This chapter explains how to write applications that use the HP X Widgets. If you are interested in writing your own widgets, read chapter 7 "Writing Widgets" in this manual.

Application programs use the HP X Widgets by calling functions of the Xt Intrinsics. Relevant functions are explained. For more details on the Xt routines, refer to *Programming With the Xt Intrinsics* in this binder.

Writing widget programs involves nine steps:

TABLE 3-1. Steps in Writing Widget Programs

Step	Description	Related Functions

1	Include required header files	`#include <X11/StringDefs.h>`
		`#include <X11/Intrinsic.h>`
		`#include <Xw/Xw.h>`
		`#include <Xw/`*widget*`.h>`

2	Initialize Xt Intrinsics	`XtInitialize(...)`
3	Add additional top-level windows	`XtCreateApplicationShell(...)` `or XtCreatePopupShell(...)`
	Do steps 4 through 6 for each widget.	
4	Set up argument lists for widget	`XtSetArg(...)`
5	Add callback routines	
6	Create widget	`XtCreateManagedWidget(...)`
7	Realize widgets and loop	`XtRealizeWidget(`*parent*`)` `XtMainLoop()`
8	Link relevant libraries	`cc +Nd2000 +Ns2000 -o`*application* `\` *application*`.c -lXw -lXt -lX11`
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 3.12. The sample code segments of each section build a simple widget program (called `demo1`) that implements a PushButton widget. The program, its defaults

file, and a picture of its output are listed below.

Program Listing demo1.c

```
/**---------------------------------------------------------------------
***
***     file:           demo1.c
***
***     project:        X Widgets example programs
***
***     description:    This program creates a PushButton widget.
***
***
***     Copyright (c) 1988, Hewlett-Packard Company.
***     All rights are reserved.
***
***-------------------------------------------------------------------*/


/*  include files  */
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/PButton.h>


/*--------------------------------------------------------------
**      SelectCB        - callback for button
*/
void SelectCB (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 ("PushButton selected.\n");
        exit (0);
}


/*--------------------------------------------------------------
**      main            - main logic for demo1 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", DEmo1, NULL, NULL, &argc, argv);
```

```
/*  set up arglist  */
        n = 0;
        XtSetArg (args[n], XtNlabel, "Push Here");  n++;
        XtSetArg (args[n], XtNwidth, 250);  n++;
        XtSetArg (args[n], XtNheight, 150);  n++;
/*  create button  */
        button = XtCreateManagedWidget ("button", XwpushButtonWidgetClass,
                        toplevel, args, n);
/*  add callback  */
        XtAddCallback (button, XtNselect, SelectCB, NULL);
/*  realize widgets  */
        XtRealizeWidget (toplevel);
/*  process events  */
        XtMainLoop ();
}
```

Defaults File DEmo1 Listing

```
#   DEmo1 app-defaults file for demo1.c
#   Place this file in /usr/lib/X11/app-defaults/DEmo1
#
#   general appearance and behavior defaults
*topShadowTile:                 foreground
*bottomShadowTile:              foreground
*topShadowColor:                light blue
*bottomShadowColor:             navy blue
*foreground:                    midnight blue
*background:                    sky blue
*allowShellResize:              true
*allowResize:                   true
*invertOnSelect:                false
*borderWidth:                   0
#
#   specific defaults for this program
*font:                          hp8.8x16b
```

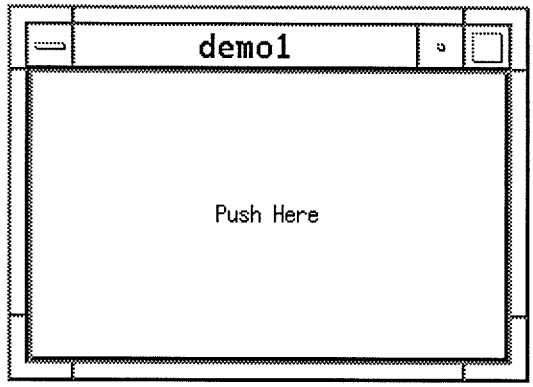


Figure 3-1. Sample Program Screen Display

Section 3.9 describes use of color in screen design and section 3.10 introduces some advanced programming techniques. Sections 3.11 and 3.12 present more involved sample programs.

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. Also, the programs in this chapter were run on a system using the hpwm window manager. If you are using another window manager, the results you see may be different from the figures shown in this chapter.

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 HP X Widgets.

3.1 Including Header Files

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

Usually this section in your program will look like this:

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/widget.h>
```

For each widget you are using in your program, replace *widget* with the name of the widget. 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/Xw`. For the PushButton widget in the sample program `demo1`, the header file name is `PButton.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 needs.

The file `StringDefs.h` contains definitions for resources used by the Xt Intrinsics. `Intrinsic.h` defines the rest of the Xt structures and variables. Variables common to all HP X Widgets are defined in `Xw.h`.

The callback procedure `SelectCB` is explained in section 3.4.

3.2 Initializing the Toolkit

You *must* initialize the before making any other calls to Xt Intrinsics functions. The function `XtInitialize` establishes the connection to the display server, parses the command line that invoked the application, 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. The options and their formats are described in chapter 3 "Running Your X Clients" of *Using the X Window System*. `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 `demo1` is:

```
toplevel = XtInitialize("main", DEmo1, NULL, NULL, &argc, argv);
```

This line names the application shell "main," the application class "DEmo1," passes no additional options, and passes the command line that invoked the application. The first two parameters are used later in setting up defaults files. They 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 `demo1` 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 two letters. For example, the sample program "demo1" would have a class name of "DEmo1."

options Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to `XrmParseCommand`. For further information, see *Programming With Xlib*.

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.

3.3 Setting Up 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 resources for a given widget are shown in the man page for the widget. The list of resources acceptable for a widget comprises not only resources unique to the widget, but also those resources inherited from widgets higher in the widget class hierarchy. Refer to chapter 7, "Writing New Widgets," for a discussion of widget class hierarchies.

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 greater than the number of elements used. The first 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 push button.

```
Arg args[10];
XtSetArg(args[0], XtNwidth, 250);
XtSetArg(args[1], XtNheight, 150);
XtSetArg(args[2], XtNlabel, "Push Here");
```

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 demo1.

```
Arg args[10];
Cardinal n=0;
XtSetArg(args[n], XtNwidth, 250); n++;
XtSetArg(args[n], XtNheight, 150); n++;
XtSetArg(args[n], XtNlabel,  "Push Here"; 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.

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 Adding Callback Procedures

Callback routines are one of the key features of the Xt Intrinsics. They allow you, the application programmer, 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 (refer to "Event Handling" in *Programming With the Xt Intrinsics* for information on processing other events). 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.

3.4.1 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 `demo1`, the callback procedure prints a message to the standard output device (usually the terminal window from which the application was invoked) and ends the program using the HP-UX system `exit` call.

```
void selectCB(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")
   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. Refer to the XWVALUATOR man page for an example of how this is used.

The general syntax of a callback procedure is described below:

```
typedef void (*XtCallbackProc)();

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.

3.4.2 Adding Callbacks

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

The sample program `demo1` creates a callback by calling the procedure `XtAddCallback`.

```
XtAddCallback (button, XtNselect, SelectCB, 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 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.

3.4.3 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 (man page XWPUSHBUTTON) used in the sample program `demo1` 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 destroyed (argument `XtNdestroyCallback`).
- Callback(s) invoked when the PushButton widget is selected (argument `XtNselect`). This is the callback you use in `demo1`.
- Callback(s) invoked when the PushButton widget is released (argument `XtNrelease`).

The translation table for this widget has been set such that a select action occurs whenever the pointer is within the widget and the user presses mouse button 1. A select action then causes the widget to invoke each of the callback routines on the callback list pointed to by its `XtNselect` argument. These routines are invoked in the order in which they appear in the callback list. In the case of the sample program `demo1`, only the routine `SelectCB` is executed.

3.5 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 `demo1`.

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

This call names the newly created widget "button" and defines it to be a PushButton widget (from the class `XwpushButtonWidgetClass`). The class "PushButton" or the name "button" can be used in defaults files (discussed in section 3.8) to refer to this widget. Its 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.

Widgets form a hierarchical structure called a widget tree. 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.

3.6 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 `demo1` 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 HP X 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 in Libraries

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

- `libXw.a` which contains the HP X Widgets.
- `libXt.a` which contains the Xt Intrinsics.
- `libX11.a` which contains the underlying Xlib library.

See section 1.3 of chapter 1 for information on compiling this and the other programs in this chapter.

NOTE

The Makefile found in the directory `/usr/contrib/Xw` contains some "N" parameters for Series 300 users. These parameters expand the compiler's internal tables to accommodate the large number of declarations made by the various components of the X Window System. If you are writing particularly large programs, you may need to expand the table sizes further.

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, the application programmer, 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. Section 3.11, "An Advanced Sample Program," shows how colors are specified inside a program.
- 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.
- A file (called `.Xdefaults`) in each user's home directory can supply default values to all applications started by the user.

All files are of the same format. Chapter 4 "User-Level Customization" and chapter 5 "System-Level Customization" in the manual *Using the X Window System* contain a detailed discussion of defaults files. There you will learn about the format of the files, specifying a different location for them, and additional user 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 `demo1`) will cause the Xt Intrinsics to look for the file `/usr/lib/X11/app-defaults/DEmo1` for default information.

```
toplevel = XtInitialize("main", "DEmo1", NULL, NULL, &argc, argv);
```

The sample defaults file below 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 `demo1` 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. It will first look for the system defaults and initialize the background color to `white`. (These defaults are compiled into the widgets.)
2. Then it will look for the *application* defaults file `/usr/lib/X11/app-defaults/DEmo1` and set the color to black.
3. Next the Xt Intrinsics will look for the *user* defaults file `.Xdefaults` and set the background color to blue.
4. If the program explicitly sets the background argument (`XtNbackground`), it will override the defaults.

3.9 Using Color

The X 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 Capabilities and Attributes

The X Widget's visual capabilities are based on specialized border and background drawing. The border drawing consists of a band around the widget two pixels wide that 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 3-2 illustrates the drawing areas.

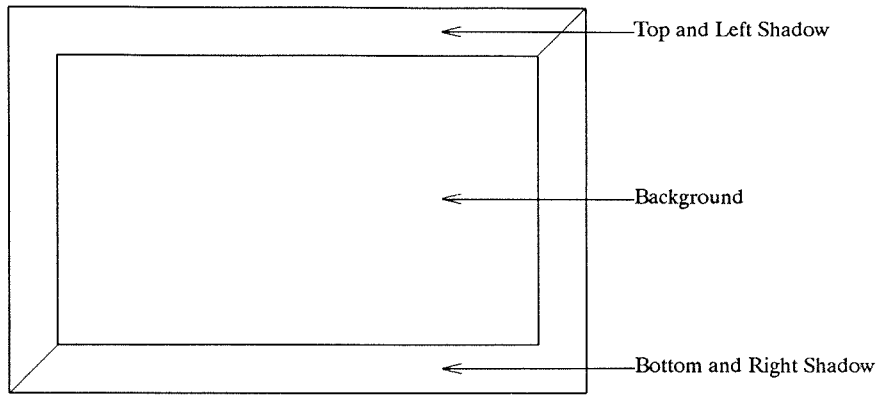


Figure 3-2. Widget Drawing Areas

Each area is drawn using a pixmap built from a combination of two colors and a bitmap tile. Refer to the `XwCreateTile(3X)` man page and figure 2-14 for a description of the set of tiles available. The top and left shadow pixmap is created using these widget resources:

- `XtNtopShadowColor`
- `XtNforeground`
- `XtNtopShadowTile`

The background pixmap is created using these widget resources:

- `XtNbackground`
- `XtNforeground`

- XtNbackgroundTile

The bottom and right shadow pixmap is created using these widget resources:

- XtNbottomShadowColor
- XtNforeground
- XtNbottomShadowTile

The primary color (top shadow color, bottom shadow color, or background) is used as the foreground parameter for all of these pixmaps when the widget makes the call to XwCreateTile to create the pixmap. The background parameter to the XwCreateTile call is *always* the foreground color resource. This may seem counterintuitive, but it is necessary for proper resource defaulting.

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, XwFrame, 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 look 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 turning the shadowing off with the XtNshadowOn resource.
- Any enclosing manager widget should appear to be lower to enhance the raised look of its selectable children. The XwFrame widget supports the lowering of managers by automatically reversing the shadowing attributes.

To give the impression that the widget is raised above its parent, set the key resources as follows:

- Set XtNtopShadowColor to a light color.
- Set XtNbackground to a medium color.
- Set the XtNbottomShadowColor to a dark color.
- Set XtNtopShadowTile and XtNbottomShadowTile to XwFOREGROUND.

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 will cause the effect to be lost.

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.

3.9.3 Example Resource Values

The following are resource settings found to be useful for both color and monochrome systems.

```
*Toggle.shadowOn:       False
*StaticText.shadowOn:   False
*RowCol.hSpace:         6
*RowCol.vSpace:         6
```

The following are resource settings found to be useful for color systems.

```
*borderWidth:           0
*topShadowTile:         foreground
*bottomShadowTile:      foreground
*invertOnSelect:        False
*invertOnEnter:         False
```

The following sets of resources contain some reasonably coordinated colors for color systems. The color names are selected from the color name data base, `/usr/lib/X11/rgb.txt`.

```
#  Blue colors
#
*topShadowColor:                        light blue
*bottomShadowColor:                     navy blue
*foreground:                            white
*background:                            sky blue

#  Green colors
#
*topShadowColor:                        pale green
*bottomShadowColor:                     dark slate gray
*foreground:                            dark green
*background:                            forest green
```

```
#  Red colors
#
*topShadowColor:                    wheat
*bottomShadowColor:                 indian red
*foreground:                        brown
*background:                        salmon

#  Gray colors
#
*topShadowColor:                    wheat
*bottomShadowColor:                 dark slate gray
*foreground:                        dim gray
*background:                        light gray
```

3.10 Advanced Programming Techniques

The sample program `demo1` described in earlier sections of this chapter illustrated the writing of a very simple widget program. The Xt Intrinsics provide additional mechanisms for programmers. For more details on any of the Xt Intrinsics functions mentioned in this section, refer to *Programming With the Xt Intrinsics*.

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.

```
Arg args[10];
args[0].name = XtNwidth;
args[0].value = (XtArgVal) 250;
args[1].name = XtNheight;
args[1].value = (XtArgVal) 150;
args[2].name = XtNlabel;
args[2].value = (XtArgVal) "Push Here";
```

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`). You can use each list with a different widget, or you can combine them using the `XtMergeArgLists(...)` function described in *Programming With the Xt Intrinsics*.

```
static Arg args[] = {
   {XtNwidth, (XtArgVal) 250},
   {XtNheight, (XtArgVal) 150},
   {XtNlabel, (XtArgVal) "Push Here"},
};
```

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", XwpushButtonWidgetClass,
   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 `XtNheight` is changed in the program from its initialized value of 150 to a run-time value of 250.

```
static Arg args[] = {
   {XtNwidth, (XtArgVal) 500},              /* item 0 */
   {XtNheight, (XtArgVal) 150},             /* item 1 */
   {XtNlabel, (XtArgVal) NULL},             /* item 2 */
};
args[1].value = (XtArgVal) 250;
args[2].value = (XtArgVal) "Push Here";
```

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 demo2 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

`XtAddCallbacks` 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 chapter 12 "Translation Management" of *Programming With the Xt Intrinsics*.

Separating Widget Creation and Management

By using `XtCreateManagedWidget`, the sample program automatically added 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

This section illustrates use of several important concepts in using widgets:

- Using layout widgets (manager widgets) in conjunction with selection and display widgets (primitive widgets) to create a widget hierarchy (widget tree).
- Modifying widgets after they have been created.
- Calling `Xlib` routines in an Xt Intrinsics program.
- Specifying shadow colors to create a three-dimensional effect.

The source code and the application defaults file for the sample program are listed in section 3.11.5 "Source Code" later in this section. They are also located on your system in `/usr/contrib/Xw/examples` as demo2.c and `DEmo2`. Instructions for compiling this program can be found in section 1.3 of chapter 1.

If you are on a monochrome system, you may want to change the colors to shades of gray in both `demo2.c` and `DEmo2`.

3.11.1 Description

The application `demo2` is a simple game, the "Color Game." The program displays an application window as shown in figure 3-3. The two sets of toggles control the color of the static text displayed at the top of the window. You can set the foreground color by moving the mouse pointer to one of the toggles in the left column and clicking mouse button 1. The right column of toggles controls the background color. To terminate the program, move the pointer over the "quit" push button and press mouse button 1.

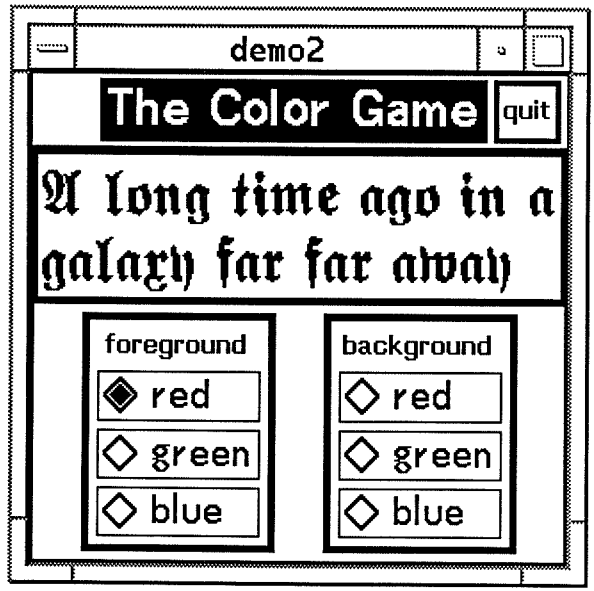


Figure 3-3. Color Game

3.11.2 Widget Hierarchy

The first step in widget application design is designing the screen layout (as is shown in figure 3-3). The second step is drawing an appropriate widget tree. The tree for the sample program is shown in figure 3-4.

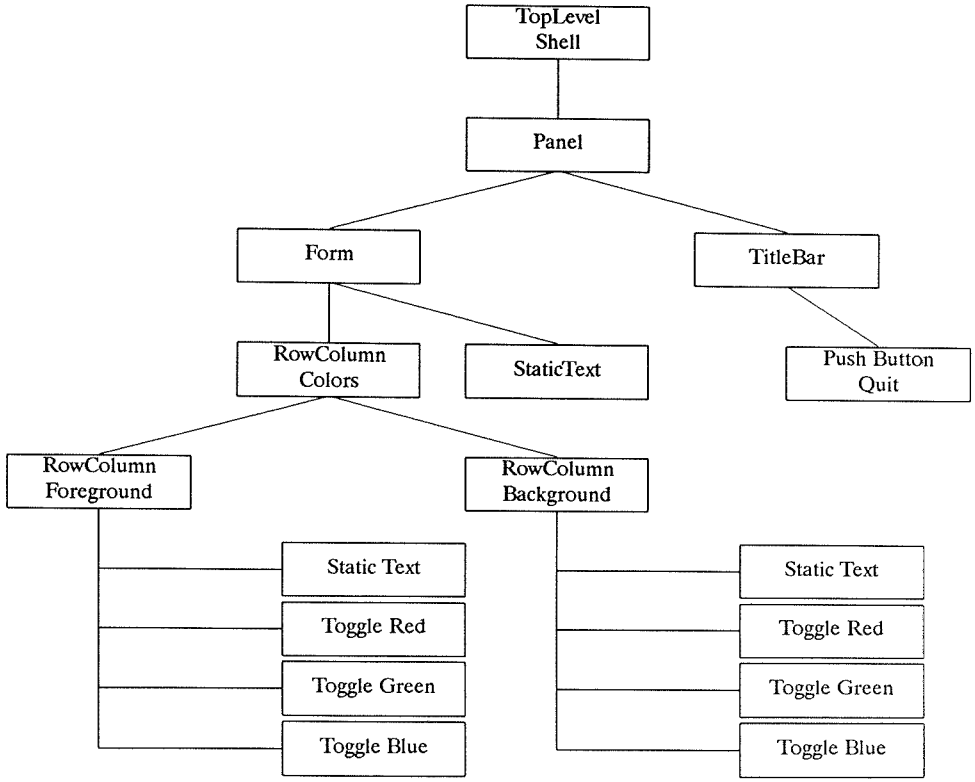


Figure 3-4. Sample Program Widget Tree

Widgets in Xt Intrinsics applications form a hierarchy, with the Shell widget (returned from a call to `XtInitialize`) at the top. A Panel widget is the highest level widget defined by the application `demo2`. As specified in Panel widget's man page, it has three areas: a title area, a menu area, and a workspace area (which can be made up of any other widget or combinations of widgets). The Color Game program specifies a TitleBar widget for the title area and a Form widget for the workspace area. The menu area is not used in this case.

The TitleBar has one child widget, a PushButton with the label "quit" that is right justified in the TitleBar. The string for the TitleBar widget ("Color Game") is contained in a StaticText widget that is internal to the TitleBar, so the application does not have to create that StaticText widget.

A Form widget serves as the parent widget for the workspace area so that exact x-y coordinates do not have to be specified. The Form widget is an extremely powerful layout widget that allows you to control the position of its children by setting special constraint arguments. The Form widget makes sure that these relationships hold true even if you

resize the window using standard window manager commands. Form widget options are explained in chapter 5. The children of the form specify these arguments as each child is created.

The StaticText widget child of the Form widget contains the string whose foreground and background colors are controlled by the toggles.

All color selection information is contained in a RowCol widget that divides the space into two equal columns, one for the foreground settings and one for the background settings. Another RowCol widget controls the layout for each of the color columns. This new RowCol widget has just one column with four rows: one row for the static text widget containing the title "foreground" or "background" and the remaining three rows for each of the color options. Separate RowCol widgets are necessary for the foreground and background to ensure that you can select both a single foreground color and a single background color.

3.11.3 Setting Arguments

As you read the next two sections, you should refer to the source code for the program "demo2.c" listed in section 3.11.5

The Color Game uses both the application defaults file `DEmo2` and argument lists to specify widget options. Note that although colors are specified as strings ("red," for example) in defaults files, you must provide a pixel value when setting a color inside your program. The subroutine `ColorNameToPixel` performs this task. The routine `ColorNameToPixel` uses Xlib functions to obtain an index into the color map. This index will correspond to the color name specified as a string. The color name must be in the color data base `/usr/lib/X11/rgb.txt` for the display server.

Arguments for each widget consist not only of arguments for the widget itself, but also of arguments specifying its spatial relationship to its siblings. The arguments controlling the layout are documented on the man page for the parent manager widget, but specified as part of the argument list for the child widget.

3.11.4 Writing the Callback Procedures

The Color Game contains three callback procedures: `QuitCB` to exit the program, `ForegroundCB` to change the foreground color, and `BackgroundCB` to change the background color.

The procedure `QuitCB` terminates the program using the standard HP-UX system call `exit(0)`.

Of the other two callback procedures, one handles changes to the foreground color, the other controls the background color. Each is passed a pixel value as an argument when it is invoked and uses `XtSetArg` and `XtSetValues` to set either the background or

foreground color resource of the static text widget.

A callback function has to be created for each of the six toggles. That record specifies which callback procedure to call (`ForegroundCB` or `BackgroundCB`) and which color to set (red, green, or blue). The callback procedure name is initialized at compile time. Calculation of the color pixels must wait until run time.

3.11.5 Source Code

The Program

```
/**---------------------------------------------------------------------
***
***     file:           demo2.c
***
***     project:        X Widgets example programs
***
***     description:    This program displays a text image and two sets of
***                     buttons.  The buttons control the background and
***                     foreground colors of the displayed text.
***
***
***     Copyright (c) 1988, Hewlett-Packard Company.
***     All rights are reserved.
***
***-------------------------------------------------------------------*/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/Form.h>
#include <Xw/Frame.h>
#include <Xw/PButton.h>
#include <Xw/Panel.h>
#include <Xw/RCManager.h>
#include <Xw/SText.h>
#include <Xw/SWindow.h>
#include <Xw/TitleBar.h>
#include <Xw/Toggle.h>

#define MAX_ARGS 20
#define TITLE_STRING "The Color Game"


/*  functions defined in this file  */
void main ();                   /*  main logic for demo2 program        */

Widget CreateButtonBox ();      /*  create box with buttons for colors  */
Pixel ColorNameToPixel ();      /*  convert color name to pixel         */
```

```
void ForegroundCB ();           /*  callback for setting fg color       */
void BackgroundCB ();           /*  callback for setting bg color       */
void QuitCB ();                 /*  callback for quit button            */


/*  global variables  */
Widget          text;           /*  display widget which changes color  */
Pixel           red, blue, green;       /*  pixel values for fg and bg  */



/*-------------------------------------------------------------
**      main            - main logic for demo2 program
*/
void main (argc,argv)
        unsigned int    argc;
        char            **argv;
{
        Widget          toplevel;       /*  Shell widget        */
        Widget          panel;          /*  Panel widget        */
        Widget          titlebar;       /*  TitleBar widget     */
        Widget          button;         /*  PushButton widget   */
        Widget          form;           /*  Form widget         */
        Widget          rowcol;         /*  RowCol widget       */
        Widget          frame;          /*  Frame widget        */

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


/*  initialize toolkit and pixel values  */
        toplevel = XtInitialize ("main", "DEmo2", NULL, 0, &argc, argv);

        red = ColorNameToPixel ("red", XtDisplay (toplevel));
        green = ColorNameToPixel ("green", XtDisplay (toplevel));
        blue = ColorNameToPixel ("blue", XtDisplay (toplevel));

/*  create panel and titlebar pane  */
        /*  create main window Panel  */
        n = 0;
        panel = XtCreateManagedWidget ("panel", XwpanelWidgetClass,
                        toplevel, args, n);

        /*  create TitleBar in Panel  */
        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwTITLE);  n++;
        XtSetArg (args[n], XtNstring, TITLE_STRING);  n++;
        XtSetArg (args[n], XtNwrap, False);  n++;
        titlebar = XtCreateManagedWidget ("titlebar", XwtitlebarWidgetClass,
                        panel, args, n);
```

```
        /*  create quit PushButton in TitleBar  */
        n = 0;
        XtSetArg (args[n], XtNlabel, "quit");  n++;
        XtSetArg (args[n], XtNregion, XwALIGN_RIGHT);  n++;
        button = XtCreateManagedWidget ("quit", XwpushButtonWidgetClass,
                        titlebar, args, n);
        XtAddCallback (button, XtNselect, QuitCB, NULL);

/*  create workspace pane  */
        /*  create Form as workspace area of Panel  */
        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwWORK_SPACE);  n++;
        form = XtCreateManagedWidget ("form", XwformWidgetClass,
                        panel, args, n);

        /*  create StaticText in Form to display color text  */
        n = 0;
        /*  constraint resources for parent  */
        XtSetArg (args[n], XtNxRefWidget, form);  n++;
        XtSetArg (args[n], XtNxResizable, True);  n++;
        XtSetArg (args[n], XtNxAttachRight, True);  n++;
        XtSetArg (args[n], XtNyRefWidget, form);  n++;
        XtSetArg (args[n], XtNyResizable, True);  n++;
        text = XtCreateManagedWidget ("text", XwstatictextWidgetClass,
                        form, args, n);

        /*  create RowCol in Form to hold button boxes  */
        n = 0;
        XtSetArg (args[n], XtNcolumns, 2);  n++;
        XtSetArg (args[n], XtNforceSize, True);  n++;
        XtSetArg (args[n], XtNlayout, XwMAXIMIZE);  n++;
        /*  constraint resources for parent  */
        XtSetArg (args[n], XtNxResizable, True);  n++;
        XtSetArg (args[n], XtNxAttachRight, True);  n++;
        XtSetArg (args[n], XtNyRefWidget, text);  n++;
        XtSetArg (args[n], XtNyAddHeight, True);  n++;
        XtSetArg (args[n], XtNyResizable, True);  n++;
        rowcol = XtCreateManagedWidget ("rowcol", XwrowColWidgetClass,
                        form, args, n);

/*  create foreground and background button boxes  */
        frame = CreateButtonBox (rowcol, "foreground", ForegroundCB);
        frame = CreateButtonBox (rowcol, "background", BackgroundCB);

/*  realize widget hierarchy  */
        XtRealizeWidget (toplevel);

/*  get and dispatch events  */
        XtMainLoop ();
}
```

```
/*-------------------------------------------------------------
**      CreateButtonBox         - create box with buttons for colors
*/
Widget CreateButtonBox (parent, title, callback)
        Widget          parent;         /*  parent widget       */
        char            *title;         /*  title string        */
        XtCallbackProc  callback;       /*  callback for buttons  */
{
        Widget          rowcol;         /*  RowCol widget       */
        Widget          frame;          /*  Frame widget        */
        Widget          label;          /*  StaticText widget   */
        Widget          separator;      /*  MenuSep widget      */
        Widget          button;         /*  Toggle widget       */

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


/*  create frame, box, and title label  */
        /*  create Frame for 3d border around button box  */
        n = 0;
        XtSetArg (args[n], XtNmode, XwONE_OF_MANY);  n++;
        XtSetArg (args[n], XtNforceSize, True);  n++;
        frame = XtCreateManagedWidget ("frame", XwframeWidgetClass,
                        parent, args, n);

        /*  create RowCol in Frame  */
        n = 0;
        XtSetArg (args[n], XtNmode, XwONE_OF_MANY);  n++;
        XtSetArg (args[n], XtNforceSize, True);  n++;
        rowcol = XtCreateManagedWidget ("rowcol", XwrowColWidgetClass,
                        frame, args, n);

        /*  create StaticText in RowCol to display button box title  */
        n = 0;
        XtSetArg (args[n], XtNstring, title);  n++;
        XtSetArg (args[n], XtNborderWidth, 0);  n++;
        label = XtCreateManagedWidget ("label", XwstatictextWidgetClass,
                        rowcol, args, n);

/*  create buttons for colors  */
        /*  create Toggle in RowCol for red  */
        n = 0;
        XtSetArg (args[n], XtNlabel, "red");  n++;
        XtSetArg (args[n], XtNselectColor, red);  n++;
        button = XtCreateManagedWidget ("button", XwtoggleWidgetClass,
                        rowcol, args, n);
        XtAddCallback (button, XtNselect, callback, red);

        /*  create Toggle in RowCol for green  */
        n = 0;
        XtSetArg (args[n], XtNlabel, "green");  n++;
        XtSetArg (args[n], XtNselectColor, green);  n++;
        button = XtCreateManagedWidget ("button", XwtoggleWidgetClass,
                        rowcol, args, n);
        XtAddCallback (button, XtNselect, callback, green);
```

```
        /*  create Toggle in RowCol for blue  */
        n = 0;
        XtSetArg (args[n], XtNlabel, "blue");  n++;
        XtSetArg (args[n], XtNselectColor, blue);  n++;
        button = XtCreateManagedWidget ("button", XwtoggleWidgetClass,
                        rowcol, args, n);
        XtAddCallback (button, XtNselect, callback, blue);

/*  return top widget in button box subtree  */
        return (frame);
}



/*-------------------------------------------------------------
**      ColorNameToPixel        - convert color name to pixel
*/
Pixel ColorNameToPixel (name, display)
        char            *name;          /*  color name          */
        Display         *display;       /*  server connection   */
{
        Colormap        colormap;       /*  colormap structure  */
        XColor          color;          /*  color structure     */

/*  get default color map  */
        colormap = XDefaultColormap (display, DefaultScreen (display));
/*  get rgb values from name  */
        XParseColor (display, colormap, name, &color);
/*  get pixel from rgb values  */
        XAllocColor (display, colormap, &color);
        return (color.pixel);
}



/*-------------------------------------------------------------
**      ForegroundCB            - callback for setting fg color
*/
void ForegroundCB (w, client_data, call_data)
        Widget          w;              /*  widget id           */
        caddr_t         client_data;    /*  pixel value         */
        caddr_t         call_data;      /*  data from widget class  */
{
        Arg             args[MAX_ARGS]; /*  arg list            */
        register int    n;              /*  arg count           */

/*  set foreground color of displayed text  */
        n = 0;
        XtSetArg (args[n], XtNforeground, (Pixel) client_data);  n++;
        XtSetValues (text, args, n);
}
```

```
/*-------------------------------------------------------------
**      BackgroundCB            - callback for setting bg color
*/
void BackgroundCB (w, client_data, call_data)
        Widget          w;              /*  widget id           */
        caddr_t         client_data;    /*  pixel value         */
        caddr_t         call_data;      /*  data from widget class  */
{
        Arg             args[MAX_ARGS]; /*  arg list            */
        register int    n;              /*  arg count           */

/*  set background color of displayed text  */
        n = 0;
        XtSetArg (args[n], XtNbackground, (Pixel) client_data);  n++;
        XtSetValues (text, args, n);
}


/*-------------------------------------------------------------
**      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 program  */
        exit (0);
}
```

The Defaults File

```
#   XFonts app-defaults file for xfonts program
#
#   general appearance and behavior defaults
*topShadowTile:                 foreground
*bottomShadowTile:              foreground
*topShadowColor:                light blue
*bottomShadowColor:             navy blue
*foreground:                    white
*background:                    sky blue
*allowShellResize:              true
*allowResize:                   true
*invertOnSelect:                false
*borderWidth:                   0
#
#   specific defaults for this program
*font:                          hp8.8x16b
*ScrollBar*granularity:         400
*StaticText*shadowOn:           false
*frame.topShadowColor:          navy blue
*frame.bottomShadowColor:       light blue
*panel.borderColor:             sky blue
*panel.borderWidth:             4
*sWindow*vsbWidth:              20
*sWindow*hsbHeight:             20
*sWindow.width:                 580
*sWindow.height:                400
*rowCol.width:                  580
*rowCol.height:                 400
```

3.12 Another Advanced Sample Program

The program presented in this section, `xfonts`, 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/Xw/examples/xfonts.c` and `/usr/contrib/Xw/examples/XFonts`. Instructions for compiling this program can be found in section 1.3 of chapter 1.

You can change the background and foreground colors and other visual attributes by changing the parameters in the app-defaults file `XFonts`.

NOTE

When running with the HP Window Manager (hpwm), you must set the resource `interactivePlacement` to True in order to be able to place the `xfonts` windows where you want them to appear on the screen. You can do this by including the following line in your `.Xdefaults` file:

```
Hpwm*interactivePlacement: True
```

If you do not have this resource set to True, all windows will automatically be placed on the screen with their upper left corner at the 0,0 (upper left) point in the root window.

When you run the program, you will see the window shown in figure 3-5.

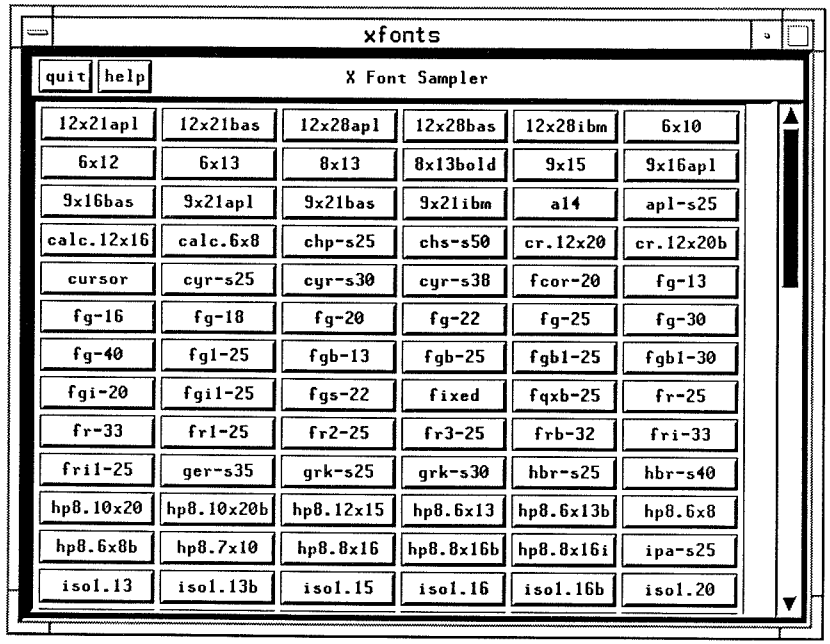


Figure 3-5. Program xfonts Main Window

You move the cursor 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. 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 "quit" pushbutton and press the left mouse button.

3.12.1 Windows Used in xfonts

There are three independent windows displayed in this program (see figure 3-5, figure 3-6, and figure 3-7):

Main Window

The main window that displays the PushButtons (see figure 3-5). This is a combination of a Frame widget, a Panel widget, a TitleBar widget, a ScrolledWindow and a RowCol widget, and a number of PushButton widgets. The Frame widget is used for refining the visual appearance of the window. The Panel widget was chosen because it has TitleBar capability and is a convenient envelope for many applications. Although Panel can have three areas (see chapter 2 and figure 2-8), only two of the areas are needed here, the title and workspace areas. In this case the title area consists of a TitleBar widget that is the parent of two PushButtons, one for help and one to quit the program. The workspace area consists of a ScrolledWindow with a RowCol widget inside it. PushButtons for each font are then placed within the RowCol widget. Layout type for the RowCol widget is `XwMAXIMUM_COLUMNS`, meaning that the maximum number of columns that can fit within this RowCol widget is calculated and the children (the font PushButtons) are laid out accordingly. A List widget could have been used instead of the combination of ScrolledWindow and RowCol widgets. In that case, StaticText widgets would have to be used instead of PushButton widgets. PushButton widgets will work within a List but the highlighting that normally occurs when a PushButton is selected is disabled when it is contained within a List widget.

Help window

The "help" window is a popup window that consists of another Frame and Panel widget combination (see figure 3-6).

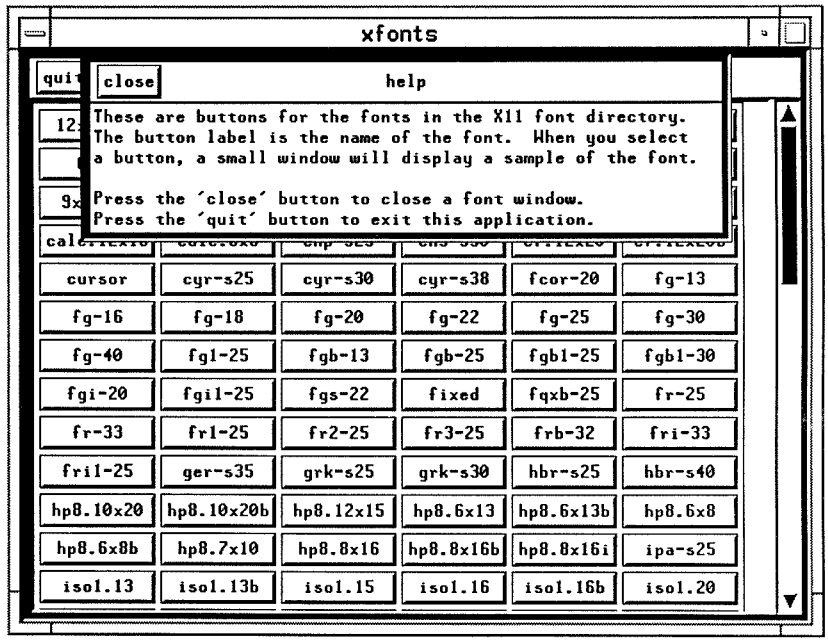


Figure 3-6. Program xfonts Help Window

The title area is much the same as the title area for the main window, and the workspace area is simply a StaticText widget that displays the help information. Other combinations of widgets could have been used to accomplish this task, but the Frame-Panel combination was chosen to maintain consistency within the application. The shell for this window is an OverrideShell widget, meaning that this shell is not under the control of the window manager. Also, as long as the help window is visible, you cannot select a font for display because of the "grab" specified in the popup function..

Font Display Window

The window that displays the selected font is also a popup window that is another Frame-Panel widget combination in the same fashion as the "help" window (see figure 3-7).

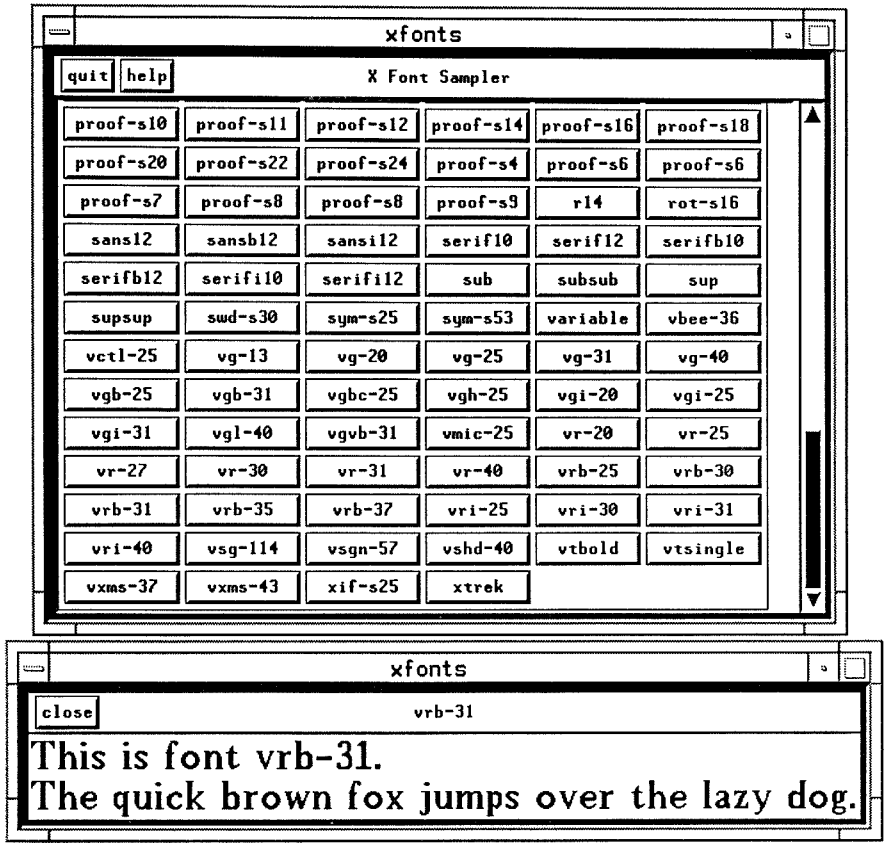


Figure 3-7. Program xfonts Text Display Window

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.

3.12.2 Widget Hierarchy

This program features three "top level" windows. The first one contains all the pushbuttons. It is created using `XtInitialize` that you used in the previous example programs. Since `XtInitialize` can only create one "top level" widget, the other two windows cannot be created by using `XtInitialize`. Since we want both the help window and the text display window to pop up, their shells are created using `XtCreatePopupShell`. The widget tree for `xfonts` is shown in figure 3-8.

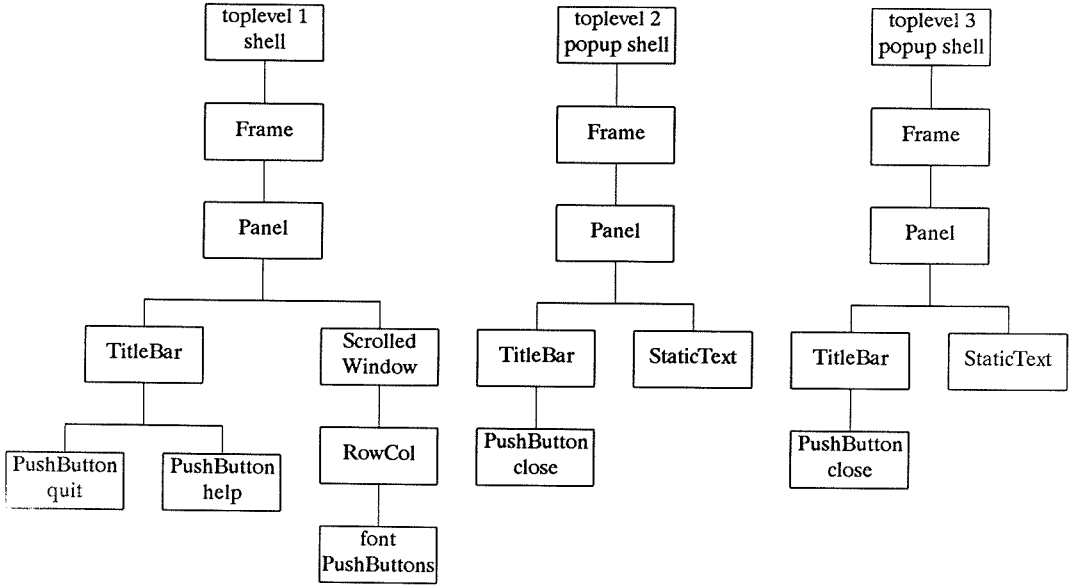


Figure 3-8. Sample Program xfonts Widget Tree

3.12.3 Source Code

The source code for `xfonts` and the default file `XFonts` are listed in the following sections.

The Program

```
/**---------------------------------------------------------------------
***
***     file:           xfonts.c
***
***     project:        X 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.
***
***
***     Copyright (c) 1988, Hewlett-Packard Company.
***     All rights are reserved.
***
***---------------------------------------------------------------------*/
#include <stdio.h>
#include <ndir.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/Frame.h>
#include <Xw/PButton.h>
#include <Xw/Panel.h>
#include <Xw/RCManager.h>
#include <Xw/SText.h>
#include <Xw/SWindow.h>
#include <Xw/TitleBar.h>

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


/*  XwName() returns the name of a widget  */
/*  this macro should be defined by the Xt Intrinsics, but isn't  */
#define XwName(widget) XrmQuarkToString ((widget) -> core.xrm_name)


/*  functions defined in this file  */

void main ();
```

```
Widget CreateApplication ();
Widget CreateFontSample ();
Widget CreateHelp ();

void SelectFontCB ();
void CloseCB ();
void HelpCB ();
void QuitCB ();


/*--------------------------------------------------------------
**      main
*/
void main (argc,argv)
        unsigned int    argc;
        char            **argv;
{
        Widget          toplevel;
        Widget          frame;


/*  initialize toolkit  */
        toplevel = XtInitialize ("main", "XFonts", NULL, 0, &argc, argv);

/*  create and realize main application window  */
        frame = CreateApplication (toplevel);
        XtRealizeWidget (toplevel);

/*  get and dispatch events  */
        XtMainLoop ();
}


/*--------------------------------------------------------------
**      CreateApplication       - create xfonts main window
*/
Widget CreateApplication (parent)
        Widget          parent;
{
        Widget          frame;
        Widget          panel;
        Widget          pane;
        Widget          rowcol;
        Widget          text;
        Widget          button;

        Arg             args[MAX_ARGS];
        register int    n;

        DIR             *dirp;
        struct direct   *item;
        char            name[15];
        int             len;
```

```
/*  create panel and titlebar pane  */
        n = 0;
        frame = XtCreateManagedWidget ("frame", XwframeWidgetClass,
                        parent, args, n);

        n = 0;
        panel = XtCreateManagedWidget ("panel", XwpanelWidgetClass,
                        frame, args, n);

        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwTITLE);  n++;
        XtSetArg (args[n], XtNstring, TITLE_STRING);  n++;
        pane = XtCreateManagedWidget ("titlebar", XwtitlebarWidgetClass,
                        panel, args, n);

        n = 0;
        XtSetArg (args[n], XtNlabel, "quit");  n++;
        XtSetArg (args[n], XtNregion, XwALIGN_LEFT);  n++;
        button = XtCreateManagedWidget ("pushButton", XwpushButtonWidgetClass,
                        pane, args, n);
        XtAddCallback (button, XtNselect, QuitCB, NULL);

        n = 0;
        XtSetArg (args[n], XtNlabel, "help");  n++;
        XtSetArg (args[n], XtNregion, XwALIGN_LEFT);  n++;
        button = XtCreateManagedWidget ("pushButton", XwpushButtonWidgetClass,
                        pane, args, n);
        XtAddCallback (button, XtNselect, HelpCB, NULL);

/*  create workspace pane  */
        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwWORK_SPACE);  n++;
        pane = XtCreateManagedWidget ("sWindow", XwswindowWidgetClass,
                        panel, args, n);

        n = 0;
        XtSetArg (args[n], XtNlayoutType, XwMAXIMUM_COLUMNS);  n++;
        XtSetArg (args[n], XtNforceSize, TRUE);  n++;
        rowcol = XtCreateManagedWidget ("rowCol", XwrowColWidgetClass,
                        pane, args, n);

/*  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] = '\0';
                n = 0;
                XtSetArg (args[n], XtNstring, name);  n++;
                button = XtCreateManagedWidget (name, XwpushButtonWidgetClass,
                                rowcol, args, n);
                XtAddCallback (button, XtNselect, SelectFontCB, NULL);
        }

        return (panel);
}


/*------------------------------------------------------------
**      CreateFontSample        - create font display window
*/
Widget CreateFontSample (parent)
        Widget          parent;
{
        Widget          shell;
        Widget          frame;
        Widget          panel;
        Widget          pane;
        Widget          text;
        Widget          button;

        Arg             args[MAX_ARGS];
        register int    n;

        char            *name;
        XFontStruct     *font = NULL;
        static char     message[BUFSIZ];


/*  get font name from parent button  */
        name = XwName (parent);

/*  create panel and titlebar pane  */
        n = 0;
        shell = XtCreatePopupShell ("font sample", topLevelShellWidgetClass,
                        parent, args, n);

        n = 0;
        frame = XtCreateManagedWidget ("frame", XwframeWidgetClass,
                        shell, args, n);

        n = 0;
        panel = XtCreateManagedWidget ("panel", XwpanelWidgetClass,
                        frame, args, n);

        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwTITLE);  n++;
        XtSetArg (args[n], XtNstring, name);  n++;
        pane = XtCreateManagedWidget ("titleBar", XwtitlebarWidgetClass,
                        panel, args, n);
```

```
        n = 0;
        XtSetArg (args[n], XtNlabel, "close");  n++;
        button = XtCreateManagedWidget ("close", XwpushButtonWidgetClass,
                        pane, args, n);

/*  load font and generate message to display  */
        font = XLoadQueryFont (XtDisplay (shell) , name);
        if  (!font)  sprintf (message, "Unable to load font: %s\0", name);
        else sprintf (message, "\
This is font %s.\n\
The quick brown fox jumps over the lazy dog.\0", name);

        XtAddCallback (button, XtNselect, CloseCB, font);

/*  create workspace pane and widget panes  */
        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwWORK_SPACE);  n++;
        XtSetArg (args[n], XtNstring, message);  n++;
        if  (font)  XtSetArg (args[n], XtNfont, font);  n++;
        text = XtCreateManagedWidget ("staticText", XwstatictextWidgetClass,
                        panel, args, n);

        return (shell);
}


/*-------------------------------------------------------------
**      CreateHelp              - create help window
*/
Widget CreateHelp (parent)
        Widget          parent;
{
        Widget          shell;
        Widget          frame;
        Widget          panel;
        Widget          pane;
        Widget          text;
        Widget          button;

        Arg             args[MAX_ARGS];
        register int    n;

        char            *name;
        XFontStruct     *font = NULL;
        static char     message[BUFSIZ];


/*  create panel and titlebar pane  */
        n = 0;
        shell = XtCreatePopupShell ("help", overrideShellWidgetClass,
                        parent, args, n);

        n = 0;
        frame = XtCreateManagedWidget ("frame", XwframeWidgetClass,
                        shell, args, n);
```

```
        n = 0;
        panel = XtCreateManagedWidget ("panel", XwpanelWidgetClass,
                        frame, args, n);

        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwTITLE);  n++;
        XtSetArg (args[n], XtNstring, "help");  n++;
        pane = XtCreateManagedWidget ("titleBar", XwtitlebarWidgetClass,
                        panel, args, n);

        n = 0;
        XtSetArg (args[n], XtNlabel, "close");  n++;
        button = XtCreateManagedWidget ("close", XwpushButtonWidgetClass,
                        pane, args, n);
        XtAddCallback (button, XtNselect, CloseCB, 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\
Press the 'quit' button to exit this application.\0");

/*  create workspace pane and widget panes  */
        n = 0;
        XtSetArg (args[n], XtNwidgetType, XwWORK_SPACE);  n++;
        XtSetArg (args[n], XtNstring, message);  n++;
        text = XtCreateManagedWidget ("staticText", XwstatictextWidgetClass,
                        panel, args, n);

        return (shell);
}


/*-------------------------------------------------------------
**      SelectFontCB            - create popup window with font sample
*/
void SelectFontCB (w, client_data, call_data)
        Widget          w;
        caddr_t         client_data;
        caddr_t         call_data;

{
        Widget          shell;

/*  create and popup font sample window  */
        shell = CreateFontSample (w);
        XtPopup (shell, XtGrabNone);
}
```

```
/*-------------------------------------------------------------
**      CloseCB                 - callback for Close button
*/
void CloseCB  (w, client_data, call_data)
        Widget          w;
        caddr_t         client_data;
        caddr_t         call_data;
{
        Window          window;
        Widget          titlebar, panel, frame, shell;
        XFontStruct     *font;

/*  free font and destroy widgets  */
        if (client_data)
        {
                font = (XFontStruct *) client_data;
                XFreeFont (XtDisplay (w), font);
        }
/*  traverse widget hierarchy to find shell  */
        titlebar  = XtParent (w);
        panel = XtParent (titlebar);
        frame = XtParent (panel);
        shell = XtParent (frame);

/*  pop down and destroy the shell widget  */
        XtPopdown (shell);
        XtDestroyWidget (shell);
}


/*-------------------------------------------------------------
**      HelpCB                  - callback for help button
*/
void HelpCB (w, client_data, call_data)
        Widget          w;
        caddr_t         client_data;
        caddr_t         call_data;
{
        Widget          shell;
        Arg             args[MAX_ARGS];
        register int    n;

        Window          root, child;
        int             root_x, root_y, win_x, win_y, mask;


/*  get the location of the mouse  */
        XQueryPointer (XtDisplay (w), XtWindow (w), &root, &child,
                        &root_x, &root_y, &win_x, &win_y, &mask);

/*  create and realize the help window  */
        shell = CreateHelp (w);
        XtRealizeWidget (shell);

/*  popup the help window over the help button  */
        XtMoveWidget (shell, root_x - 30, root_y - 20);
        XtPopup (shell, XtGrabExclusive);
```

```
/*  unset the help button since it lost the release event  */
        n = 0;
        XtSetArg (args[n], XtNset, FALSE);  n++;
        XtSetValues (w, args, n);
}


/*-------------------------------------------------------------
**      QuitCB                  - callback for quit button
*/
void QuitCB (w, client_data, call_data)
        Widget          w;
        caddr_t         client_data;
        caddr_t         call_data;
{
/*  terminate the application  */
        exit (0);
}
```

The Defaults File

```
#-------------------------------------------------------------
#   XFonts app-defaults file for xfonts program
#
#   general appearance and behavior defaults
*topShadowTile:              foreground
*bottomShadowTile:           foreground
*topShadowColor:             light blue
*bottomShadowColor:          navy blue
*foreground:                 white
*background:                 sky blue
*allowShellResize:           true
*allowResize:                true
*invertOnSelect:             false
*borderWidth:                0
#
#   specific defaults for this program
*font:                       hp8.8x16b
*ScrollBar*granularity:      400
*StaticText*shadowOn:        false
*frame.topShadowColor:       navy blue
*frame.bottomShadowColor:    light blue
*panel.borderColor:          sky blue
*panel.borderWidth:          4
*sWindow*vsbWidth:           20
*sWindow*hsbHeight:          20
*sWindow.width:              580
*sWindow.height:             400
*rowCol.width:               580
*rowCol.height:              400
#-------------------------------------------------------------*/
```

This page left blank intentionally.

Menus 4

The HP Widgets menus are versatile and easy to use. The menu system provides resource entries that allow you to tailor menus to fit any application. This chapter will explain menus and how you can use them.

4.1 Menu System Description

The menu system is composed of several widgets that are arranged as children of shell widgets. This section describes the menu hierarchy and menu views. A later section will describe the widgets that make up the menu system: `MenuMgr`, `MenuPane`, `MenuSep`, and `MenuButton`.

4.1.1 Menu Hierarchy

A menu consists of a hierarchy of widgets, as shown in figure 4-1.

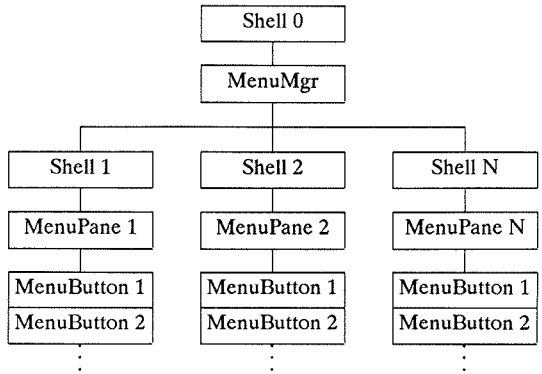


Figure 4-1. Menu Hierarchy

A menu consists of MenuButton and MenuSep widgets that are placed into MenuPane widgets and managed by the MenuMgr. A MenuButton widget is a single menu item, while a MenuSep widget is used to separate unrelated buttons or groups of buttons within the MenuPane. A MenuPane widget displays one level of the menu including all of the MenuButtons in that level. A menu with several levels is built by placing the MenuPane

widgets into the MenuMgr. The MenuMgr, MenuPane, MenuSep, and MenuButton widgets are discussed in later sections of this chapter.

4.1.2 Menu Manager Views

The MenuMgr is capable of displaying the menu in one of two views, popup or pulldown. The major difference between these two views is the manner in which they are posted. The toplevel of the pulldown is always visible, while the popup appears as the result of some user input, normally by clicking one of the mouse buttons. Both views can have cascading submenus beneath the toplevel. Each of these views is described below.

Popup

The popup menu usually appears due to a post action generated by the application user. The popup menu is displayed as a vertical list of menu items and may optionally have a title on the top of the list. The menu items that contain a cascade indicator (an arrow by default) can be used to display a cascading submenu. The menu can cascade down as many levels as desired.

Figure 4-2 and figure 4-3 show how a popup menu is cascaded. In figure 4-2, the bottom MenuButton, labeled "More," has an arrow indicating the presence of a submenu. Moving the pointer into this MenuButton and then over the arrow causes the submenu to appear, as shown in figure 4-3. Note that there is a submenu available from the bottom MenuButton here also.

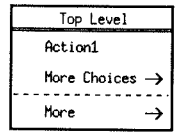


Figure 4-2. Popup Top Level Menu

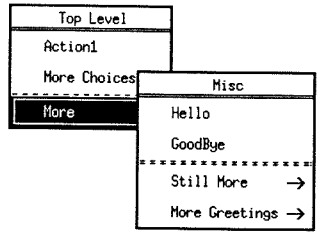


Figure 4-3. Popup Top Level and Cascading Menus

Pulldown

The toplevel of the Pulldown menu is always available. An example of the toplevel of a

Pulldown menu is shown in figure 4-4.

Figure 4-4. Top Level of a Pulldown Menu

Performing a post action on one of these items allows you to see the next menu level. For example, if you move the mouse pointer to "Houses" and press mouse button 1 (the default post action), the next menu level under "Houses" will appear, as shown in figure 4-5.

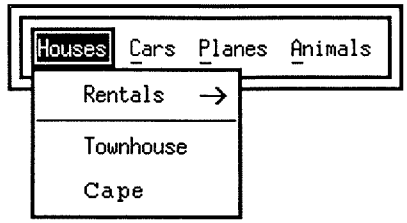


Figure 4-5. Next Level of a Pulldown Menu

The second level menus may have one or more cascade indicators that can be used to display additional cascading submenus. In figure 4-5 for example, the MenuButton labeled "Rentals" has an arrow in the cascade field, indicating the presence of a submenu. Moving the mouse pointer to this arrow causes the submenu to be displayed, as shown in figure 4-6.

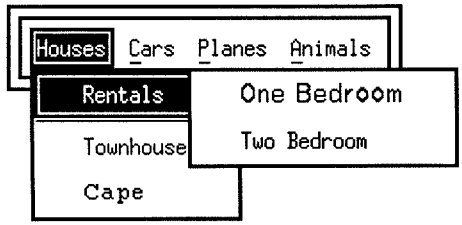


Figure 4-6. Second Level of a Pulldown Menu

Note the underscores under the first letters of each item in the toplevel of the Pulldown menu. These are called *mnemonics*, and they are used to post toplevel MenuPanes and to select MenuButtons by using the keyboard. The toplevel mnemonics are always active and

are selected by using a combination of an accelerator keystroke (for example, Extend Char) and the mnemonic keystroke. Mnemonics in menus at lower levels are active only when that MenuPane is showing, and when active they can be accessed by simply pressing the mnemonic key.

4.1.3 Data Specification

The data specification for the menu hierarchy is independent of the menu view. The data is specified in a single tree that resembles a cascading menu. This tree is interpreted differently for the two views. The cascading tree is shown in figure 4-7.

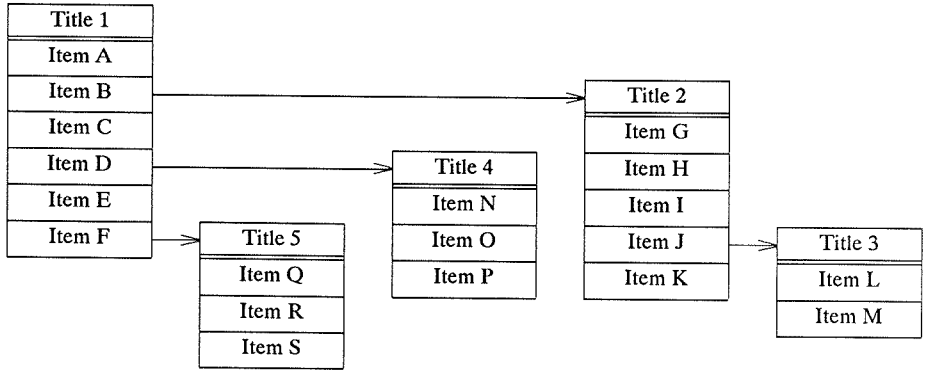


Figure 4-7. Cascading Menu Tree

A Pulldown menu using this data tree can be displayed in one of two ways. The resource `XtNallowCascades` determines what is done on the lower levels of the tree. For Pulldown menus, this resource entry defaults to TRUE, thus allowing cascading submenus. The resulting menu layout is shown in figure 4-8.

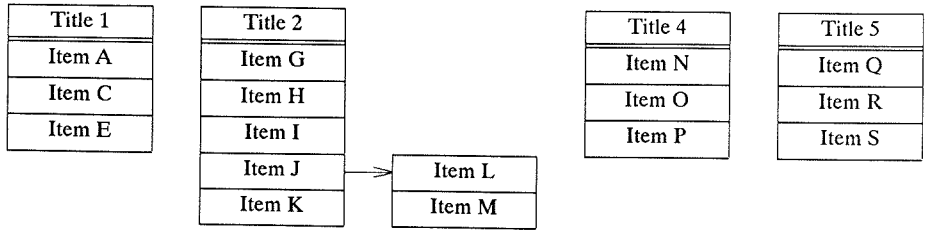


Figure 4-8. Pulldown Menu Layout

The title entries are essential for the Pulldown menu style. If a title is not specified, the MenuMgr uses the name of the widget for the title. Note that the entries in the first MenuPane that have a submenu attached (such as *Item B*) no longer appear in that MenuPane.

The cascading lower levels of the Pulldown menus can be disabled by setting a resource entry in the MenuMgr. The menu layout shown in figure 4-9 is the result. Note that *Item J* is now *title 3*, and has been folded into the toplevel.

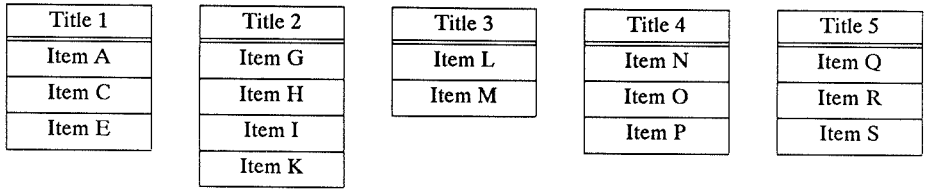


Figure 4-9. Menu Layout

CAUTION

Do not attempt to attach a MenuPane to one of its own children. Doing so will cause an unbreakable loop and may cause your program to crash.

4.2 Menu Components

The menu system is composed of several widgets that are arranged as children of shell widgets (see figure 4-1). This section describes the individual components of the menu system.

4.2.1 Menu Manager

The MenuMgr widget is a composite manager meta class. (Refer to chapter 6, "Composite Widgets," in *Programming With the Xt Intrinsics*.) A meta class widget is never instantiated, but serves as a mechanism for providing a set of resources that are common to many other types of MenuMgr widgets. The MenuMgr is the controller of the menu system and handles the presentation of the two menu views, popup and pulldown. It manipulates the MenuPane, MenuSep, and MenuButton widgets to present the view.

A MenuMgr can be associated with a single widget or a widget and all of its children. This widget is referred to as the *associated widget*. A menu associated with a widget and all of its children is very useful when that widget is a manager (such as a panel), especially if the number of children in the widget varies during the course of the application. The MenuMgr sets up the necessary translations on the associated widget. If the menu is also associated with the widget's children, then button or key "grabs" are set up to capture all

of the events of interest.

The MenuMgr sets up the input translation and the "grabs" necessary for the menu system to be transparently added to the associated widget. You simply specify the "post" and "select" button actions in the MenuMgr. Through the use of these resources, you can easily customize the menu system to operate according to a "drag" model, a "double click" model, or whatever model is desired.

The MenuMgr exports a resource by which the application may specify a keyboard event that can be used to unpost the menus without selecting an item. The MenuMgr also exports a resource that may be used to specify a keyboard event that will select the currently highlighted menu item. Both of these resources are useful when keyboard traversal is active in a menu hierarchy.

PopupMgr

The PopupMgr supports a post keyboard accelerator as an alternative way to post the menu. A menu posted with the keyboard accelerator functions the same as one posted with the post button action, except that the pointer may not be initially positioned in the menu.

The PopupMgr supports a *sticky* menu mode. In this mode the menu system remembers the set of cascading MenuPanes that were posted when you last selected a menu item. The next time you request that the menu be posted, that entire set of MenuPanes is displayed. The pointer is moved to the exact MenuButton that was last selected if the menu was posted with the post button action. If the sticky menu mode is not enabled, then only the toplevel MenuPane is displayed each time the menu is posted.

The PopupMgr handles the positioning of the MenuPanes when they are posted. When you post the toplevel MenuPane using the post button action, the pane is positioned so that the pointer is located in the first item of the MenuPane, unless the sticky menu mode is enabled. The cascading submenus are positioned next to the MenuButton that brought up the submenu. When you post the toplevel MenuPane using the post keyboard accelerator, the pane is positioned in the center of its associated widget. If the MenuPane will not fit on the display, it is repositioned so that the entire MenuPane is visible.

A menu item is selected in the menu by either executing the button select action (as defined in the PopupMgr), by pressing the key defined as the "keyboard select" key, or by entering a keyboard accelerator (as defined by a particular MenuButton widget in the menu). Regardless of how the menu item is selected, the visible MenuPanes are removed from the display before the selected item is called.

The PopupMgr allows its MenuPanes to have a title on the top, bottom, or both top and bottom. The display of the titles is determined by the MenuPanes' resource settings and is not modified by the PopupMgr.

The PopupMgr provides a global function (`XwPostPopup()`) that may be used by an application to force a toplevel MenuPane to be posted. The application must specify a position (relative to a specific widget) to which the MenuPane will be posted, and the identification of the specific widget.

Pulldown Manager

The Pulldown menu manager creates a Pulldown widget across the top of the associated widget. If the associated widget is not a composite widget, then the Pulldown widget cannot be created and no menu is attached. If this situation occurs, a warning will be displayed in the application window. The Pulldown menu manager notifies the associated widget of the Pulldown widget's presence so that the associated widget may treat the Pulldown widget specially. For example, if the associated widget is a Panel widget, the Panel may insure that the Pulldown widget is always positioned at the top of the Panel and always spans the width of the Panel. The Pulldown widget contains titlebuttons created by the Pulldown menu manager for each first-level MenuPane in the menu system. The first-level MenuPanes are those that may be accessed directly from the Pulldown widget; each titlebutton corresponds to a first level MenuPane. Cascading submenus are not first-level MenuPanes. Each titlebutton is defined by the title attributes in the associated MenuPane widget. If a title attribute is changed in a MenuPane, then the corresponding change also occurs in the titlebutton.

The Pulldown menu manager allows you to determine whether or not cascading submenus are allowed. If cascading submenus are disabled, then any cascading MenuPanes are folded into first-level MenuPanes and are accessed from titlebuttons. Cascading submenus are enabled as a default.

The menu is posted by executing the post button action in one of the titlebuttons or by executing the post mnemonic defined for one of the first-level MenuPanes. The MenuPane is positioned just below the corresponding titlebutton. If the MenuPane will not fit on the display, then it is repositioned so that the entire MenuPane is visible.

A menu item is selected by executing the select button action, by typing a keyboard accelerator (as defined in a particular MenuButton widget in the menu), by typing a mnemonic (as defined in a displayed MenuButton widget), or by using the keyboard select action.

The Pulldown menu manager disables all titles within the MenuPanes that make up the menu. The title information is extracted and used to build the titlebuttons in the Pulldown widget. If the MenuPane is a cascading submenu, its title is also disabled.

4.2.2 Menu Pane Widget

The MenuPane widget is a composite manager meta class. (Refer to chapter 6, "Composite Widgets," in the *Programming With the Xt Intrinsics* manual.) Remember that a meta class widget is never instantiated, but serves as a mechanism for providing a set of resources that are common to many other types of widgets. Thus, the MenuPane widget provides resources that are common to many other types of MenuPane widgets.

The common resources include such pieces of information as the title that is displayed in the MenuPane, the font and color of the title, the type of title to use (string or image), the name of the MenuButton to which the MenuPane is attached, and a mnemonic which may be used by certain MenuMgr widgets for posting the MenuPane.

Cascade

The Cascade widget is a subclass of the MenuPane meta class and is classified as a Composite Manager widget. It is a specialized manager because it can only manage widgets that are a subclass of the MenuButton widget. This type of widget is normally used when constructing Popup or Pulldown menus.

As MenuButton or MenuSep widgets are added to a Cascade widget, they are positioned according to the insertion algorithm supplied by the Manager meta class. When the MenuButton is created, its argument list is searched for the argument name `XtNchildPosition`. The value associated with `XtNchildPosition` indicates where this MenuButton is to be placed with respect to the other MenuButtons contained within the MenuPane.

The Cascade widget supplies a single additional resource that allows the location of the MenuPane title to be specified. A title may be displayed at the top, bottom, or top and bottom of a cascade widget.

A Cascade widget always assumes its ideal size, as determined by its children and its title. As its children grow or shrink, the MenuPane adapts as needed.

4.2.3 MenuButton Widget

The MenuButton widget provides a wide range of resources so that each item in a menu can be defined with unique attributes (such as color, font, string or image, sensitive state, and so on). The MenuButton is a subclass of the `XwButtonClass` widget.

The MenuButton widget consists of three areas:

- The mark area.
- The label area.
- The cascade indicator area.

These areas are shown in figure 4-10.

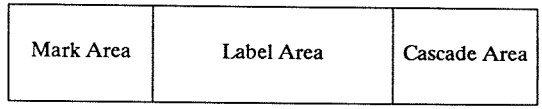


Figure 4-10. Menu Button Areas

The mark area can be used to checkmark a menu item. The label area contains the desired menu item string or image. The cascade indicator area is primarily used by the MenuMgr for indicating that a cascading submenu is present. By default, the mark is a checkmark, the label is the name of the widget, and the cascade indicator is an arrow. The label can be set to any text string or image, and the label area attempts to grow or shrink to accommodate the new size. The size is typically set by the MenuPane widget to insure consistency throughout a single MenuPane. You may modify the image used for the mark and cascade indicators, but the height and width of these areas will not change.

A select callback is provided on the MenuButtons. The select translation is typically determined in the MenuMgr and the selected MenuButton callback routines are called. The MenuButton provides keyboard accelerators and mnemonics. The MenuMgr determines whether or not the keyboard accelerator and mnemonic are made available. If the mnemonic specified is found in the label string, it is underlined, unless disallowed by the MenuMgr.

4.2.4 MenuSep Widget

The MenuSep widget is a Primitive widget that is used to separate MenuButtons or groups of MenuButtons within a MenuPane. Several different line styles are available, and the selection is made by means of the resource `XtNseparatorType`. The default is a single line, but double lines and dashed lines may be selected. Common resources from Core and Primitive meta classes can be set for this widget also. Refer to the MenuSep man page for more information.

4.3 Creating a Menu

The steps required to create the menu hierarchy shown in figure 4-1 are given below. Following each step is a code segment that shows how a popup cascading menu could be created. These steps provide the greatest degree of control over the menu, although it may be somewhat cumbersome to build.

1. Create the widget (such as a Panel) that the menu is to be associated with.

```
panel=XtCreateManagedWidget("panel",XwpanelWidgetClass,parent,NULL,0);
```

2. Create a popup shell as a child of the associated widget.

```
mmgrshell=XtCreatePopupShell("mmgrshell",shellWidgetClass,panel,NULL,0);
```

3. Create the MenuMgr as a child of the shell.

```
mmgr=XtCreateManagedWidget("mmgr",XwpopupmgrWidgetClass,mmgrshell,NULL,0);
```

4. Create a popup shell as a child of the MenuMgr.

```
paneshell=XtCreateManagedWidget("paneshell",shellWidgetClass,mmgr,NULL,0);
```

5. Create a MenuPane as a child of the shell.

```
pane=XtCreateManagedWidget("pane",XwcascadeWidgetClass,paneshell,NULL,0);
```

6. Create a MenuButton as a child of the MenuPane.

```
button1=XtCreateManagedWidget("button1",XwmenubuttonWidgetClass,pane,NULL,0);
```

7. Continue creating MenuButton widgets as children of the MenuPane until all desired MenuButton widgets are added.

```
button2=XtCreateManagedWidget("button2",XwmenubuttonWidgetClass,pane,NULL,0);
button3=XtCreateManagedWidget("button3",XwmenubuttonWidgetClass,pane,NULL,0);
```

8. Continue creating the shell-MenuPane pairs with the desired MenuButton widgets until the menu is complete.

```
paneshell2=XtCreateManagedWidget("paneshell2",shellWidgetClass,mmgr,NULL,0);
pane2=XtCreateManagedWidget("pane2",XwcascadeWidgetClass,paneshell2,NULL,0);
button4=XtCreateManagedWidget("button4",XwmenubuttonWidgetClass,pane2,NULL,0);
```

4.4 Using Menus

Once the menu system has been built, it is relatively easy to use. Communication with the menu system is achieved through the callback functions or by the application modifying menu attributes. These subjects are covered below, followed by a description of keyboard traversal.

4.4.1 Callbacks

The menu system provides applications with the means for attaching callback functions to the MenuButton or MenuPane select actions. These functions are called after the menu system has been removed from the screen. Typically the MenuPane select is used only to remove the menu system. The MenuButton select callbacks are most often used by an application to perform some action.

4.4.2 Keyboard Interface

The menu system provides a keyboard interface to the menus through the use of keyboard accelerators to post a MenuPane, keyboard accelerators to select a MenuButton, and traversal key definitions. The general keyboard interface scheme is discussed in chapter 6, "Keyboard Interface."

By enabling traversal, the menu system can be driven from the keyboard. The menu highlighting scheme changes so that the menu item that has the focus (that is, the menu item that is active) is highlighted. The focus and highlighting are moved to another menu item by means of the traversal key definitions. The pointer movements through the menu do not change the focus or the highlight. Note that enabling traversal and then using the pointer to operate the menu can be confusing, since the highlighting does not track the pointer although the pointer select action is still defined. If the menu traversal is enabled, you should plan on operating the menu with the keyboard only.

NOTE

For the current release, keyboard traversal is *not* supported by the Pulldown menu manager widget.

4.5 Mixing Menu Accelerators and Traversal

Due to limitations imposed by the Xt Intrinsics, menu accelerators and mnemonics will not function if the application has enabled keyboard traversal within the application. You can overcome this problem with the addition of a single event handler that is attached to each primitive widget in which the menu accelerators or mnemonics would be expected to function.

When you enter a menu accelerator or mnemonic from an application that has keyboard traversal *disabled*, the request is directed to the appropriate MenuMgr widget which then processes the request. When you enter a menu accelerator or mnemonic from an

application that has keyboard traversal *enabled*, the Xt Intrinsics redirect this to the Primitive widget that currently has the traversal focus. The MenuMgr does not receive the request and therefore the request is not processed. By attaching the special event handler to the Primitive widgets, you can "forward" accelerators and mnemonics to the MenuMgr for processing.

The event handler is defined as follows:

```
void ForwardAccelerators (w, menuMgr, event)
        Widget    w;          /* Widget with the traversal focus */
        Widget    menuMgr;    /* MenuMgr to which request is forwarded */
        XKeyEvent *event;     /* Potential accelerator or mnemonic */
{
        Arg       args[1];
        Boolean   associateChildren = False;

        XtSetArg(args[0], XtNassociateChildren, &associateChildren);
        XtGetValues(menuMgr, args, 1);

        /*
         * If the reporting widget is the widget to which the menu is
         * attached, OR if the menu system is configured to accept
         * input from children of the widget to which the menu is
         * associated, then forward the event to the MenuMgr.
         */

         if ((XtParent(XtParent(menuMgr)) == w) || associateChildren)
         {
                 (*(((XwMenuMgrWidgetClass)(XtClass(menuMgr)))->
                   menu_mgr_class.widgetEventHandler)) (w, menuMgr, event);
    }
}
```

The following code segment shows the correct usage of the event handler:

```
        /*
         * This code segment assumes that you have created a RowCol widget
         * with a PopupMgr menu attached to it.  We are also assuming that
         * keyboard traversal is enabled within the application.
         *
         * We will now create several children of the RowCol widget and
         * attach the event handler to each of them so that accelerators
         * and mnemonics will work properly.
         *
         * The "menuMgr" parameter to the XtAddEventHandler() calls is
         * the widget id of the MenuMgr widget to which the accelerators
         * or mnemonics are to be forwarded.
         */

    XtSetArg(args[0], XtNtraversalType, XwHIGHLIGHT_TRAVERSAL);
        btn1 = XtCreateManagedWidget("btn1", XwpushButtonWidgetClass,
                rcparent, args, 1);
```

```
XtAddEventHandler(btn1, KeyPressMask, False, ForwardAccelerators,
        menuMgr);

btn2 = XtCreateManagedWidget("btn2", XwpushButtonWidgetClass,
        rcparent, args, 1);

XtAddEventHandler(btn2, KeyPressMask, False, ForwardAccelerators,
        menuMgr);
```

4.6 A Sample Program

The program listing on the following pages is that of a very basic use of the MenuMgr, MenuPane, and MenuButton widgets. The source for this program can be found in `/usr/contrib/Xw/examples/menudemo.c`. See section 1.3 of chapter 1 for instructions on compiling this program.

```
#include <X11/Xlib.h>
#include <X11/IntrinsicP.h>
#include <X11/Intrinsic.h>
#include <X11/Xatom.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <Xw/Xw.h>
#include <Xw/XwP.h>
#include <Xw/MenuBtn.h>
#include <Xw/Cascade.h>
#include <Xw/PopupMgr.h>
#include <Xw/BBoard.h>

Widget toplevel, bboard;
Widget mmgrshell, mmgr;
Widget paneshellA, paneshellB, paneshellC;
Widget menupaneA, menupaneB, menupaneC;
Widget buttona1;
Widget buttonb1, buttonb2, buttonb3;
Widget buttonc1, buttonc2, buttonc3;


/****** Widget & Children ArgLists ******/

static Arg toplevelArgs [] = {
   {XtNallowShellResize, (XtArgVal) True}
};

static Arg bboardArgs [] = {
   {XtNwidth, (XtArgVal) 200},
   {XtNheight, (XtArgVal) 300}
};
```

```
/****** Menu Manager ArgLists ******/


static Arg menumgrArgs [] = {
   {XtNassociateChildren, (XtArgVal) True},
   {XtNstickyMenus, (XtArgVal) True},
   {XtNmenuPost, (XtArgVal) "<Btn1Down>"},
   {XtNmenuSelect, (XtArgVal) "<Btn3Down>"},
};


/****** Menu Pane ArgLists ******/


static Arg MenuPaneAArgs [] = {
   {XtNattachTo, (XtArgVal) "mmgr"}
};

static Arg menupaneBArgs [] = {
   {XtNattachTo, (XtArgVal) "more"}
};

static Arg menupaneCArgs [] = {
   {XtNattachTo, (XtArgVal) "stillmore"}
};


/****** Menu Buttons (pane A) ArgLists ******/

static Arg buttona1Args [] = {
   {XtNlabel, (XtArgVal) "More"},
};


/****** Menu Buttons (pane B) ArgLists ******/

static Arg buttonb1Args [] = {
   {XtNlabel, (XtArgVal) "Hello"},
};

static Arg buttonb2Args [] = {
   {XtNlabel, (XtArgVal) "GoodBye"},
};

static Arg buttonb3Args [] = {
   {XtNlabel, (XtArgVal) "Still More"},
};


/****** Menu Buttons (pane C) ArgLists ******/

static Arg buttonc1Args [] = {
   {XtNlabel, (XtArgVal) "Boy"},
};
```

```
static Arg buttonc2Args [] = {
   {XtNlabel, (XtArgVal) "Girl"},
};

static Arg buttonc3Args [] = {
   {XtNlabel, (XtArgVal) "??"},
};


/*********************************/

void Hello()
{
   printf ("Hello!!\n");
}

/*********************************/

void Goodbye()
{
   printf ("Goodbye!!\n");
}

/*********************************/

void Boy()
{
   printf ("Boy!!\n");
}

/*********************************/

void Girl()
{
   printf ("Girl!!\n");
}

/*********************************/

void Question()
{
   printf ("??\n");
}

/*********************************/


void
main(argc, argv)
    int argc;
    char **argv;
{
   Widget children[20];

   /*********************************/
   /* Create the toplevel widget */
   /*********************************/
```

```
toplevel = XtInitialize (argv[0], "menusample", NULL, 0, &argc, argv);
XtSetValues (toplevel, toplevelArgs, XtNumber(toplevelArgs));


/*****************************************/
/* A bulletin board will be our primary widget */
/*****************************************/
bboard = XtCreateManagedWidget("bboard", XwbulletinWidgetClass, toplevel,
                               bboardArgs, XtNumber(bboardArgs));


/*************************/
/* Create a MenuMgr */
/*************************/

mmgrshell = XtCreatePopupShell("mgrshell", shellWidgetClass,
                               bboard, NULL, 0);
mmgr = XtCreateManagedWidget("mmgr", XwpopupmgrWidgetClass,
                               mmgrshell, menumgrArgs, XtNumber(menumgrArgs));


/*************************************************************************/
/* Create 3 MenuPanes, which will be used to build the menu hierarchy */
/*************************************************************************/

paneshellA = XtCreatePopupShell("paneshellA", shellWidgetClass, mmgr,
                                NULL, 0);
menupaneA = XtCreateManagedWidget("menupaneA", XwcascadeWidgetClass,
                                paneshellA, menupaneAArgs,
                                XtNumber(menupaneAArgs));

paneshellB = XtCreatePopupShell("paneshellB", shellWidgetClass, mmgr,
                                NULL, 0);
menupaneB = XtCreateManagedWidget("menupaneB", XwcascadeWidgetClass,
                                paneshellB, menupaneBArgs,
                                XtNumber(menupaneBArgs));

paneshellC = XtCreatePopupShell("paneshellC", shellWidgetClass, mmgr,
                                NULL, 0);
menupaneC = XtCreateManagedWidget("menupaneC", XwcascadeWidgetClass,
                                paneshellC, menupaneCArgs,
                                XtNumber(menupaneCArgs));


/*****************************************/
/* Create the MenuButtons for MenuPane A */
/*****************************************/

buttona1 = XtCreateManagedWidget ("more", XwmenubuttonWidgetClass,
               menupaneA, buttonalArgs, XtNumber (buttonalArgs));


/*****************************************/
/* Create the MenuButtons for MenuPane B */
/*****************************************/
```

```
    children[0] = buttonb1 = XtCreateWidget ("hello",
                  XwmenubuttonWidgetClass, menupaneB, buttonb1Args,
                  XtNumber (buttonb1Args));
    children[1] = buttonb2 = XtCreateWidget ("goodbye",
                  XwmenubuttonWidgetClass, menupaneB, buttonb2Args,
                  XtNumber (buttonb2Args));
    children[2] = buttonb3 = XtCreateWidget ("stillmore",
                  XwmenubuttonWidgetClass, menupaneB, buttonb3Args,
                  XtNumber (buttonb3Args));
    XtManageChildren (children, 3);


    /****************************************/
    /* Create the MenuButtons for MenuPane C */
    /****************************************/

    children[0] = buttonc1 = XtCreateWidget ("boy", XwmenubuttonWidgetClass,
                  menupaneC, buttonc1Args, XtNumber (buttonc1Args));
    children[1] = buttonc2 = XtCreateWidget ("girl", XwmenubuttonWidgetClass,
                  menupaneC, buttonc2Args, XtNumber (buttonc2Args));
    children[2] = buttonc3 = XtCreateWidget ("question",
                  XwmenubuttonWidgetClass, menupaneC, buttonc3Args,
                  XtNumber (buttonc3Args));
    XtManageChildren (children, 3);


    /*********************************/
    /* Attach all action callbacks */
    /*********************************/

    XtAddCallback (buttonb1, XtNselect, Hello, NULL);
    XtAddCallback (buttonb2, XtNselect, Goodbye, NULL);
    XtAddCallback (buttonc1, XtNselect, Boy, NULL);
    XtAddCallback (buttonc2, XtNselect, Girl, NULL);
    XtAddCallback (buttonc3, XtNselect, Question, NULL);

    /*********************************************************/
    /* Realize the widget tree, and start processing events */
    /*********************************************************/

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

This page left blank intentionally.

Form Widgets 5

The Form widget is a special kind of manager (or layout) widget. It allows an application to specify a desired set of relationships between the children being laid out. The Form widget remembers the relationships specified and uses these relationships, or constraints, to manage its children whenever any of the following conditions occur:

- When the Form widget is resized.
- When new children are added.
- When existing children are resized, unmanaged, remanaged or destroyed.

For example, the Form widget allows the application to state that widget B should have the same *y* coordinate as widget A, and that widget B should be attached to the left side of the Form widget. Further, widget C should be attached to the left, bottom, and right side of the Form widget. It should be resizable so that when the Form widget changes size it will also be resized to maintain the specified relationships.

5.1 Using the Form Widget

Use the Form widget when you have a collection of widgets that will be dynamically changing but you want to retain a certain spatial relationship (that might otherwise be lost) among the children. If you are creating a box with some buttons that is basically static after it is created, it would be simpler to use another layout widget (such as the row column manager widget) rather than the Form widget.

The Form widget accomplishes its functionality by exporting a constraint language. In the language of the Xt Intrinsics, the Form widget is a Constraint widget. Each widget created as a child of the Form widget has appended to it a block of information called a constraint record. It is in this constraint record that the Form widget stores the relationships or constraints that determine how a child will be laid out. When a child is created, these constraints are specified by arguments to `XtCreateWidget`. Table 5-1 is a table of constraints supported by the Form widget and will assist in the discussion of how these constraints might be used. For a complete explanation of the fields in the table, refer to the Form widget man page at the end of this document.

TABLE 5-1. Constraint Resource Set – Children of FORM(3X)

Name	Class	Type	Default
XtNxRefName	XtCXRefName	String	NULL
XtNxRefWidget	XtCXRefWidget	Widget	the parent form
XtNxOffset	XtCXOffset	int	0
XtNxAddWidth	XtCXAddWidth	Boolean	False
XtNxVaryOffset	XtCXVaryOffset	Boolean	False
XtNxResizable	XtCXResizable	Boolean	False
XtNxAttachRight	XtCXAttachRight	Boolean	False
XtNxAttachOffset	XtCXAttachOffset	int	0
XtNyRefName	XtCYRefName	String	NULL
XtNyRefWidget	XtCYRefWidget	Widget	the parent form
XtNyOffset	XtCYOffset	int	0
XtNyAddHeight	XtCYAddHeight	Boolean	False
XtNyVaryOffset	XtCYVaryOffset	Boolean	False
XtNyResizable	XtCYResizable	Boolean	False
XtNyAttachBottom	XtCYAttachBottom	Boolean	False
XtNyAttachOffset	XtCYAttachOffset	int	0

It is often useful to be able to specify that a child will span the entire width or height (or both) of its parent, regardless of the sizes the parent is forced to take. For example, you normally would want a TitleBar positioned at the top of the window and have it span the width of its parent. The following code segment shows how this could be accomplished.

```
Widget toplevel, form1, tbar1;

/* Create the Form */

form1 = XtCreateManagedWidget ("form1", XwformWidgetClass,
                    toplevel, (ArgList) args, 0);

/* Create the TitleBar;  say that both its x and its y coordinates
 * are to match that of its parent, the Form; give the TitleBar the
 * string "Radio Buttons" to display; anchor the TitleBar to the
 * right side of the Form; and say it is resizeable in both
 * the x and y directions--this allows it to conform to the
 * constraints that it be attached to both the left and the right
 * sides of the Form.
 */
XtSetArg (args[0], XtNxRefWidget, (caddr_t) form1);
XtSetArg (args[1], XtNyRefWidget, (caddr_t) form1);
XtSetArg (args[2], XtNstring, "Radio Buttons");
XtSetArg (args[3], XtNxAttachRight, TRUE);
XtSetArg (args[4], XtNxResizable, TRUE);
XtSetArg (args[5], XtNyResizable, TRUE);
tbar1= XtCreateManagedWidget ("title1", XwtitlebarWidgetClass,
                              form1, (ArgList) args, 6);
```

The next example creates a set of relationships between three widgets. The widgets are aligned into a single row with the leftmost widget attached to the left edge of the form, the rightmost widget attached to the right edge of the form and the middle widget centered in the form between the other two. See figure 5-1.

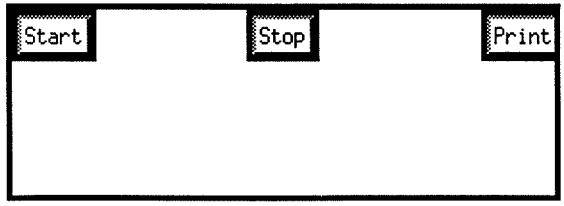


Figure 5-1. Form Widget

```
/* Create the Form*/
 form1 = XtCreateManagedWidget ("form1", XwformWidgetClass,
                         toplevel, (ArgList) args, 0);


/* Create the leftmost PushButton; both its x and y coordinates
 * reference its parent, the Form; add 10 pixels of offset from
 * both the top and the edge of the Form.
 */
 XtSetArg (args[0], XtNxRefWidget, (caddr_t) form1);
 XtSetArg (args[1], XtNyRefWidget, (caddr_t) form1);
 XtSetArg (args[2], XtNxOffset, 10);
 XtSetArg (args[3], XtNyOffset, 10);
 pb1= XtCreateManagedWidget ("pb1", XwpushButtonWidgetClass,
                               form1, (ArgList) args, 4);
```

```
/* Create the middle PushButton; both its x and y coordinates
 * reference the first PushButton, pb1.  If no other adjustment
 * were made to its x,y values, this widget would occupy the same
 * space as the first PushButton.  The XtNxAddWidth says to
 * add the width of the reference widget (here its pb1) to the
 * x coordinate;  say that this widget can be resized in the
 * x direction if necessary (for this example, it is resized when
 * the Form is made smaller);  allow the x offset between this
 * widget, pb2, and its x coordinate reference widget, pb1, to
 * vary.  When the Form is made larger, the space between pb1
 * and pb2 will grow; when the Form is made smaller the space
 * will shrink. NOTE that in this example whether the PushButtons
 * can actually be made to touch each other depends on their
 * highlight thickness settings.
 */
XtSetArg (args[0], XtNxRefWidget, (caddr_t) pb1);
XtSetArg (args[1], XtNyRefWidget, (caddr_t) pb1);
XtSetArg (args[2], XtNxAddWidth, TRUE);
XtSetArg (args[3], XtNxResizable, TRUE);
XtSetArg (args[4], XtNxVaryOffset, TRUE);
pb2= XtCreateManagedWidget ("pb2", XwpushButtonWidgetClass,
                            form1, (ArgList) args, 5);

/* Create the rightmost PushButton; both its x and  y coordinates
 * reference the middle PushButton, pb2.  We again add the width
 * of the referenced widget to its x coordinate to prevent overlapping;
 * attach the widget to the right edge of the Form; allow the offset
 * between this widget and its x coordinate reference widget to
 * vary.
 */
XtSetArg (args[0], XtNxRefWidget, (caddr_t) pb2);
XtSetArg (args[1], XtNyRefWidget, (caddr_t) pb2);
XtSetArg (args[2], XtNxAddWidth, TRUE);
XtSetArg (args[3], XtNxAttachRight, TRUE);
XtSetArg (args[4], XtNxVaryOffset, TRUE);
pb3= XtCreateManagedWidget ("pb3", XwpushButtonWidgetClass,
                            form1, (ArgList) args, 5);
```

5.2 Summary

Most of the Form widget's constraints are illustrated in the previous examples. It is a powerful but complicated widget and may sometimes require experimentation to learn how the various constraints interact. The man page for the Form widget at the end of this manual provides additional general information, as well as an explanation for each of the constraints.

Keyboard Interface 6

The HP X Widgets keyboard interface mechanism provides a method of keyboard input and interaction to augment the mouse. This capability is necessary for a variety of application classes, including mouseless systems and systems that include a mouse but have a number of text edit or page edit input components.

The keyboard interface involves two distinct processing areas:

- Keyboard input processing to an individual widget.
- Keyboard traversal between widgets.

6.1 Keyboard Input Processing

The keyboard input processing is handled through proper definitions of the widget's default translations that handle keyboard input. For example, the normal processing of the PushButton is defined by the following translations:

- When mouse button 1 is pressed, a select procedure is invoked.
- When mouse button 1 is released, a release procedure is invoked.

These translations define the processing functions contained within the widget to be called when the particular events occur. You can set these translations to understand keyboard input as well as the mouse button input. For the PushButton widget, this can be defined as follows (on HP keyboards):

- When button 1 *or* the Select key is pressed, a select procedure is invoked.
- When button 1 *or* the Select key is released, a release procedure is invoked.

The translation manager and a proper set of translations can handle the keyboard input processing needed for a keyboard interface.

6.2 Keyboard Traversal

Keyboard traversal is more difficult to accomplish since it requires interaction between widgets, not just interaction within a single widget. The traversal mechanism provided solves the majority of the traversal problems encountered by applications. The following list is the set of general capabilities of the traversal handling mechanism.

- Intra-widget hierarchy and inter-widget hierarchy traversal.
- Traversal routing within complex widget hierarchies.
- Continuation of traversal when interrupted by mouse selection.
- Active and inactive widgets within a traversal set.

NOTE

Keyboard traversal is not implemented for Pulldown menus.

6.2.1 Visual Attributes

When keyboard traversal is on, the widget that is activated displays its border according to its highlight style and highlight thickness. As the keyboard traversal keys are pressed, the highlight moves from widget to widget. If the pointer is moved out of the toplevel widget of the active widget hierarchy, the border highlight on the active widget is removed. When the pointer moves back into the toplevel widget, the highlight is restored to the same widget.

6.2.2 Application Control

There is a set of simple control mechanisms that allow an application to control and modify how traversal works. These mechanisms fall into three main areas.

- Keyboard traversal activation and deactivation.
- Keyboard traversal key definitions through translation definition.
- Keyboard traversal handling between multiple root window-based widget hierarchies. This is accomplished through an application-supplied callback used in conjunction with a keyboard traversal invocation function supplied by the HP X Widgets. Refer to the `XwMoveFocus` man page for more information.

Traversal Activation

Widgets indicate the desire for keyboard traversal through an argument type. There is an argument type for subclasses of Primitive widgets (such as PushButton) and subclasses of manager widgets (such as row column manager).

To activate keyboard traversal for an entire widget hierarchy, the application can set the manager resource `XtNtraversalOn` to TRUE and the primitive resource `XtCTraversalType` to `highlight_traversal`.

If the application creates a manager widget with `XtNtraversalOn` value set to FALSE, the immediate child widgets of the manager widget will not perform traversal. However, if one of these children is also a manager widget and its `XtNtraversalOn` value is TRUE, its children *will* have traversal active.

When the keyboard traversal mechanism makes a widget active, the widget is given the keyboard focus. When this occurs, all keyboard input directed at the application will go to the active widget regardless of the location of the pointer. The exception to this processing state is if the pointer is moved out of the toplevel widget in the hierarchy. When this occurs, the focus is taken from the active widget.

The keyboard traversal mechanism is implemented to keep the active field visible whenever possible. This places an additional constraint on composite widgets (such as the row column manager) to calculate their children's visibility when keyboard traversal is to occur. The manager widgets support two ways to handle keyboard traversal visibility:

- Traverse only to currently visible children.
- If a child becomes invisible, move the traversal focus to a visible widget.

Traversal Key Definitions

The keyboard input that drives the traversal between widgets is defined through the translation manager. Each widget supports eight directions: up, down, left, right, next, previous, next top, and home.

The translations necessary to support keyboard traversal are defined in the primitive and manager meta classes. These definitions augment the translation table for a widget *if* it has specified that it wants keyboard traversal.

On HP keyboards, the keys shown in the table on the next page generally control the directions indicated. Refer to the primitive and manager widget man pages for detailed information.

Direction	Key	Meaning
Up	▲	Traverse to the widget most nearly above this widget. If no widget is above it, find the widget that is *below* this widget and closest to the bottom of the widget hierarchy. In other words, wrap the search to the bottom of the window and search up from there.
Down	▼	Traverse to the widget that is most nearly below the widget. If no widget is below it, find the widget that is *above* this widget and closest to the top of the widget hierarchy. In other words, wrap the search to the top of the window and search down from there.
Left	◄	Traverse to the widget that is most nearly to the left of this widget. If no widget is to the left of this widget, find the widget that is to the *right* of this widget and closest to the right edge of the window. In other words, wrap the search to the right and search to the left from there.
Right	►	Traverse to the widget that is most nearly to the right of this widget. If no widget is to the right of this widget, find the widget that is to the *left* of this widget and closest to the left edge of the window. In other words, wrap the search to the left and search to the right from there.
Next	Next	Traverse to the widget that appears next (regardless of the physical position) in the list of children maintained by the parent of the widget that has the focus. If there are no more children in the parent's list, traverse to the next child in the *grandparent's* list of children. When the end of the children is reached, wrap to the beginning of the list of children.
Prev	Prev	Traverse to the widget that appears prior to the widget that has the focus (regardless of the physical position) in the list of children maintained by the parent of the widget that has the focus. If there are no previous children in the parent's list, traverse to the previous child in the *grandparent's* list of children. When the beginning of the widget hierarchy is reached, wrap to the end of the list of children.
Home	Home	Traverse to the widget that is closest to the parent's origin (0,0) point. If this widget already has the focus, move to the widget that is closest to the *grandparent's* origin (0,0) point.
Next Top	Enter	Find the topmost widget in this hierarchy that is a subclass of manager, and have it issue any `XtNnextTopCallbacks` that have been registered. Note that the Enter key is located on the numeric keypad to the right of the main keyboard.

When a keyboard traversal key is pressed, it is translated by the translation manager, which then calls one of the widget's traversal routines. The widget then makes a request to its parent widget to perform the traversal. The parent does this by calculating the next widget to be traversed to and then issuing a focus activation call to the widget to get it activated.

The order of traversal between widgets is controlled by the parent (manager) of the widgets. The specific widget managers perform the location calculations based on their widget ordering algorithms and the direction of traversal supplied by the child widget. The managers also ensure the visibility of the widget to be traversed.

Traversal Search

Keyboard traversal uses a search path based on the height and width of the active widget (including the border highlight). Figure 6-1 shows how this works.

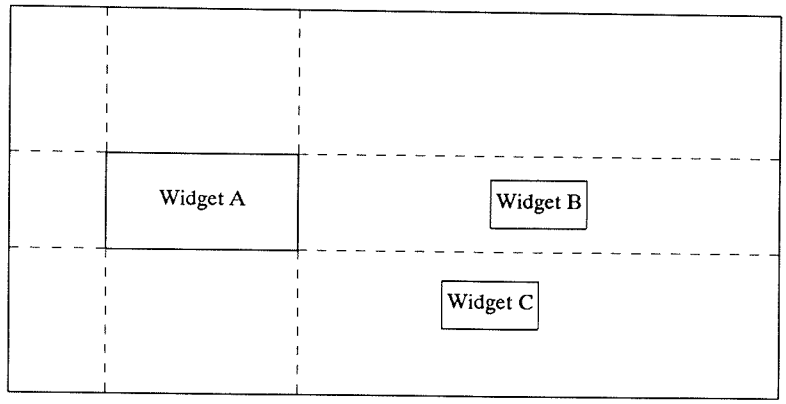


Figure 6-1. Widget Search Path

The search path for Widget A is shown by the horizontal and vertical dashed lines. If an up or down direction is specified, no traversal will occur since there is no widget within the search path in either direction. Traversal to the right or left will ultimately find Widget B since it lies within the search path. Traversal from Widget A to Widget C (or the reverse) is not possible since neither widget lies within the other's search path. Traversal from Widget B to Widget C (or the reverse) is possible. Note that only a portion of a widget needs to be in the search path for traversal to occur.

Traversal Between Root Level Widget Sets

Traversal between distinct widget hierarchies cannot be handled automatically by the traversal mechanism because it relies on the ability to ascend and descend the widget hierarchies. A mechanism is provided to traverse between widget hierarchies with minimal application intervention. All subclasses of manager widgets implement a callback and all subclasses of primitive widgets define an input translation to drive the callback for a `NextTop` traversal key.

To use this capability, the application defines a callback function and adds it to the toplevel widget of each widget hierarchy in the traversal set. When the key for `nextTop` is pressed, an internal function is invoked that ascends the widget hierarchy until the toplevel widget is found. It then invokes any application-supplied callback functions that are attached to that widget.

At this point the application's callback function needs to determine which widget hierarchy to activate. When it has done this, the hierarchy can be activated by calling the following function.

```
XwMoveFocus (w)
Widget w;
```

The widget specified as a parameter should be the toplevel shell widget of the widget hierarchy to be activated.

6.3 Internal Implementation Requirements for Traversal

There are several requirements placed on both Primitive and Composite widgets that are necessary to implement keyboard traversal properly. New widgets will need to meet these requirements to correctly implement keyboard traversal for these widgets.

6.3.1 Primitive Widget Requirements

The requirements placed on Primitive widgets are to incorporate the Primitive class and instance structure into their class hierarchy definition. The initialize and set values procedures provided by the Primitive class will augment the widget's translation table to support keyboard traversal.

6.3.2 Manager Widget Requirements

The requirements placed on Composite widgets are:

- Incorporate the Manager class and instance structures into the class hierarchy definition.
- Inherit the generalized keyboard traversal handling procedure from the manager class record.

The next chapter deals with a more advanced topic, writing new widgets. If you do not plan to write your own widgets, you need not read this chapter.

Writing New Widgets 7

The previous chapters of this manual were concerned with using the set of widgets contained in the HP X Widgets library. The HP X Widgets library contains a base set of widgets, and it was intended that these widgets, alone and in combination, would solve a majority of an application writer's needs. However, there may be occasions when new widgets are needed. This chapter will describe how to create a new widget.

7.1 Widget Description

This chapter will describe the construction of a new Primitive widget – a multistate button. The number of states can be set by the application writer, thus allowing different instantiations of the same class to have a different number of states. The button will display a different label to indicate each state. When selected, the button will inform the application which state has been selected and will then change state and display the appropriate label. This new widget will be of class `XwmultiButtonWidgetClass`.

One of the advantages of building new widgets on top of an existing widget library such as the HP X Widgets library is that many of super and meta classes are available upon which to base the new widget. For our purposes, the new widget will be a subclass of `XwbuttonWidgetClass`, and it will therefore inherit resources and procedures from that class. These resources are:

- **Core**. Includes basic resources (such as x, y, width, and height) needed by all widgets.
- **Primitive**. Includes resources to accomplish highlighting around the button, as well as resources and procedures to handle keyboard focus manipulation. It also defines the set of callbacks available to all Primitive widgets.
- **Button**. Includes additional resources and procedures that aid in the display and management of all types of buttons.

Thus, many of the resources common to all widgets are provided by these classes.

7.2 Constructing a Widget

The process of constructing a widget consists of the creation of three separate files:

- A private header file that contains definitions for the widget class and instance structures. This file will be used by other widgets that wish to become subclasses of the new widget. Its purpose is to hide the widget structure from the application writer. This forces the programmer to use set values rather than the actual fields. For this widget, the private header file will be named `MButtonP.h`, and it will reside in the directory `/usr/include/Xw`.
- A public header file that contains the external definitions needed for an application writer to make use of the widget. It will typically contain the name of the widget class (in this case it will be `XwmultiButtonWidgetClass`), as well as the definition of any new resources needed to define the widget. For this widget, the public header file will be named `MButton.h`, and it will also reside in the directory `/usr/include/Xw`.
- A source code file that contains C source code for the widget.

The following sections will describe each of these files in detail.

7.2.1 The Private Header File

The *Xt Intrinsics* manual describes a widget as having both a class record and an instance record. There is always exactly one class record for every widget class and it is from this class record that the data and procedures needed to create instances of this widget are found.

The class record for a widget class is a collection of structures (or class parts) contributed by each of the superclasses of the widget, as well as a class part for the widget being constructed. Since it is not necessary to create any new procedures for the new widget, the class part for the multi-state button will be empty. The inclusion of the "int" field in the code segment below is solely to placate fussy C compilers.

```
typedef struct {int nada;} XwMultiButtonClassPart;
```

The entire class record for the new widget will then be:

```
typedef struct _XwMultiButtonClassRec {
   CoreClassPart           core_class;
   XwPrimitiveClassPart    primitive_class;
   XwButtonClassPart       button_class;
   XwMultiButtonClassPart  multi_button_class;
} XwMultiButtonClassRec;
```

Once the class record for the new widget has been built, the instance record template can be built. Each time an instance of the new widget is created, this template will be used in conjunction with the procedures and data in the class record to fill out the fields in the instance record. Like the class record, the instance record is also composed of structures contributed by the superclasses of the widget and any special instance parts required for this widget. The new multi-state button widget will need four additional fields:

- A pointer to an array of strings. Each of the strings will be used to label the button depending on the state of the button.
- A count of the labels provided by the application.
- A state field to allow the button to keep track of the following data:
  - The label to be displayed.
  - The state information to be returned to the application when the button is selected.
- A flag telling the button to invert its foreground or background. This is most useful on black-and-white systems where the change in the shadow highlights may not be noticeable. It defaults to true (meaning invert on selection) since all HP X Widgets resource defaults are targeted to a black and white system.

Given this, the `XwMultiButtonPart` for each instance record of the new widget will be:

```
typedef struct _XwMultiButtonPart{
    String * labels;
    int      num_labels;
    int      state;
    Boolean invert_on_select;
} XwMultiButtonPart;
```

Note that `String` is defined to be a "char *" in `Intrinsic.h`. When this is added to the instance record parts of the superclasses of `XwMultiButton`, the full instance record becomes:

```
typedef struct _XwMultiButtonRec{
    CorePart              core;
    XwPrimitivePart       primitive;
    XwButtonPart          button;
    XwMultiButtonPart     multi_button;
} XwMultiButtonRec;
```

The above code segments represent the entire contents of the private header file. Below is a complete description of the contents of an instance record for the multi-state button. It is interesting to note what each superclass contributes to the widget, and this information should facilitate future discussions of how size values are calculated. A later section of this chapter will detail the class record for the multi-state button widget.

A pointer to a multi-state button widget actually points to the following:

```
{
  /* CORE PART */
  {
    Widget          self;               /* pointer to widget itself          */
    WidgetClass     widget_class;       /* pointer to Widget's ClassRec      */
    Widget          parent;             /* parent widget             */
    String          name;               /* widget resource name              */
    XrmName         xrm_name;           /* widget resource name quarkified   */
    Screen          *screen;            /* window's screen                   */
    Colormap        colormap;          /* colormap                         */
    Window          window;             /* window ID                         */
    Position        x, y;               /* window position                   */
    Dimension       width, height;      /* window dimensions                 */
    Cardinal        depth;              /* number of planes in window        */
    Dimension       border_width; /* window border width               */
    Pixel           border_pixel;       /* window border pixel               */
    Pixmap          border_pixmap;      /* window border pixmap or NULL      */
    Pixel           background_pixel;   /* window background pixel           */
    Pixmap          background_pixmap;  /* window background pixmap or NULL  */
    struct _XtEventRec *event_table;    /* private to event dispatcher       */
    struct _TMRec   tm;                /* translation management           */
    caddr_t         constraints;       /* constraint record                 */
    Boolean         visible;            /* is window mapped and not occluded?*/
    Boolean         sensitive;          /* is widget sensitive to user events*/
    Boolean         ancestor_sensitive; /* are all ancestors sensitive?      */
    Boolean         managed;            /* is widget geometry managed?       */
    Boolean         mapped_when_managed;/* map window if it's managed?       */
    Boolean         being_destroyed;    /* marked for destroy                */
    XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
    WidgetList      popup_list;        /* list of popups                   */
    Cardinal        num_popups;        /* how many popups                  */
  }
```

```
 /* XWPRIMITIVE PART */
{
  Pixel   foreground;
  int     background_tile;
  int     traversal_type;
  Boolean I_have_traversal;
  int     highlight_thickness;
  int     highlight_style;
  Pixel   highlight_color;
  int     highlight_tile;
  Boolean shadow_on;
  Pixel   top_shadow_color;
  int     top_shadow_tile;
  Pixel   bottom_shadow_color;
  int     bottom_shadow_tile;
  Boolean recompute_size;
  GC      highlight_GC;
  GC      top_shadow_GC;
  GC      bottom_shadow_GC;
  Boolean display_sensitive;
  Boolean highlighted;
  Boolean display_highlighted;
  XtCallbackList select;
  XtCallbackList release;
  XtCallbackList toggle;
}
/* XWBUTTON PART */
{
  XFontStruct * font;
  char        * label;
  int           label_location;
  Dimension     internal_height;
  Dimension     internal_width;
  int           sensitive_tile;
  GC            normal_GC;
  GC            inverse_GC;
  Position      label_x;
  Position      label_y;
  Dimension     label_width;
  Dimension     label_height;
  unsigned int  label_len;
  Boolean       set;
  Boolean       display_set;
}
/* XWMULTIBUTTON PART */
{
 String * labels;
 int      num_labels;
 int      state;
 Boolean  invert_on_select;
}
}
```

7.2.2 The Public Header File

The public header file is included in applications that wish to create instances of the multi-state button widget. It provides definitions for all new resources used by the widget, as well as a pointer to the class record of the widget. This pointer is used as the `widget_class` parameter in the Xt Intrinsics functions `XtCreateWidget` and `XtCreateManagedWidget`. Thus, the public header file for the multi-state button widget will contain the following:

```
#define XtNlabels               "labels"
#define XtCLabels               "Labels"
#define XtNinvertOnSelect       "invertOnSelect"
#define XtCInvertOnSelect       "InvertOnSelect"
#define XtNnumLabels            "numLabels"
#define XtCNumLabels            "NumLabels"

extern WidgetClass  XwmultiButtonWidgetClass;

typedef struct _XwMultiButtonClassRec * XwMultiButtonWidgetClass;
typedef struct _XwMultiButtonRec      * XwMultiButtonWidget;
```

For a full explanation of why the above resources (such as `XtNlabels`) are defined, refer to chapter 11, "Resource Management," in the *Xt Intrinsics* manual.

7.2.3 The Source Code File

The source code for the widget is found in a C source file, the ".c" file. The following sections discuss each of the procedures or declarations needed to produce the multi-state button widget. The C source code is listed at the end of this section and can be found in the directory `/usr/contrib/Xw/examples/MultiButton`.

The Header Files

The following header files must be included:

1. <`Xll/Intrinsic.h`>. This file provides access to definitions of toolkit structures and macros available to both the widget writer and the application writer.
2. <`Xll/IntrinsicP.h`>. This file provides access to the private structures of the Xt Intrinsics meta classes (core, composite, and constraint), as well as macros and external references for functions intended for the widget writer.
3. <`Xll/StringDefs.h`>. This file provides access to all of the string definitions used for base resource (name, class, and representation) used by the Xt Intrinsics.
4. <`Xll/Misc.h`>. This file provides access to macros such as `Max`, `Min`, `AssignMax`, and `AssignMin`.
5. <`Xw/Xw.h`>. This file provides access to all of the public resource definitions for meta classes (such as primitive and manager) as well as definitions to be used in

argument lists to set these resources.

6. <Xw/XwP.h>. This file provides access to the private structure of HP X Widgets meta classes (such as primitive and manager), as well as external definitions of procedures and macros for widget writers.
7. <Xw/MButtonP.h>. The private header for the multi-state button widget.
8. <Xw/MButton.h>. The public header for the multi-state button widget.

Translations and Action Lists

Translations are used by a widget to bind events occurring in that widget to predefined actions. The application or the user can redefine which events get bound, but the list of actions to handle these events are hard-coded into the widget.

The translations are encoded as a string where a particular character sequence maps into an X event. For example, in the language of the Xt Intrinsics, the term "<Btn1Down>" corresponds to the X event generated when the mouse button 1 is depressed. In addition to the string specifying the event, there is also a string specifying the action to be taken when that event occurs. For instance:

```
static char defaultTranslations[] =

      "<Btn1Down>:          xyz()\n\
       <Btn1Up>:            abc()";
```

There are two events in the above code segment, button 1 down and button 1 up, bound to two actions that are identified by the strings "xyz" and "abc" respectively. An action list corresponding to these translations contains the identifying strings ("xyz" and "abc") and matches them to procedures within the widget:

```
static XtActionsRec  actionList[]=
 {
  {"xyz", (XtActionProc) Select},
  {"abc", (XtActionProc) Unselect},
 };
```

The indirection between the translations and the actions is necessitated by the fact that most systems do not support dynamic linking or loading. Thus, a user can specify new translations for a widget in his .Xdefaults file, and this allows the Xt Intrinsics to match the specified events with procedures in the widget through the use of the identifying strings. For more detail on these issues, refer to chapter 12, "Translation Management" in the *Programming With the Xt Intrinsics* manual.

The translations and actions for the multi-state button widget are as follows:

```
static char defaultTranslations[] =

    "<Btn1Down>:            select()\n\
     <Btn1Up>:              rotate()\n\
     <EnterWindow>:         enter()\n\
     <LeaveWindow>:         leave()\n\
     <KeyUp>Select:         rotate()\n\
     <KeyDown>Select:       select()";


static XtActionsRec actionsList[] =
{
  {"select", (XtActionProc) Select},
  {"rotate", (XtActionProc) Rotate},
  {"enter", (XtActionProc) _XwPrimitiveEnter},
  {"leave", (XtActionProc) _XwPrimitiveLeave},
};
```

When a <Btn1Down> event occurs, the `Select` procedure is invoked. It visually indicates that the button has been selected and calls any callbacks that have been registered for the `XtNselect` callback. It passes the current state to the callback procedure as `call_data`, and it marks the button as being set. The same sequence of actions are initiated when this widget has the keyboard focus and the Select key is pressed.

When a <Btn1Up> occurs, the `Rotate` procedure is invoked. It will increment the internal state flag and display the next label in the label sequence on the button face. It marks the button as being *not* set and calls any callbacks that have been registered for the `XtNrelease` callback. It passes the new state to the callback procedure as `call_data`. The same sequence of actions is initiated when this widget has the keyboard focus and the Select key is released.

When an <EnterWindow> occurs, an HP X Widgets library routine is called. If the button is to be highlighted when the cursor enters it, this routine will draw a highlight around the edge of the button window. This capability is handled by the primitive meta class and requires no effort on the part of the multi-state button beyond including the translations and actions in its code.

When a <LeaveWindow> occurs, an HP X Widgets library routine is called. If the button was highlighted when the cursor entered it, it will be unhighlighted.

Notice that multiple events can be bound to the same action in the translations. Also, it is possible to have an event invoke multiple actions. The code segment below shows this functionality.

```
<Btn1Down>:        select() notify()
```

Resources for the Multi-State Button

The public header file defines name and class strings to identify three resources for the multi-state button widget that can be set by applications. There are many other such resources provided by the widgets superclasses. A default setting must be provided for these resources to handle the case where an application does not specify their setting. The resource list for the multi-state button widget will be:

```
static XtResource resources[] =
{
   {
     XtNlabels, XtCLabels, XtRLabels, sizeof (caddr_t),
     XtOffset (XwMultiButtonWidget, multi_button.labels),
     XtRPointer, (caddr_t) NULL
   },

   {
     XtNnumLabels, XtCNumLabels, XtRInt, sizeof (int),
     XtOffset (XwMultiButtonWidget, multi_button.num_labels),
     XtRString, "0"
   },
  {
     XtNinvertOnSelect, XtCInvertOnSelect, XtRBoolean, sizeof (Boolean),
     XtOffset (XwMultiButtonWidget, multi_button.invert_on_select),
     XtRString, "True"
  },
};
```

For more information on the resource list and its structure, refer to chapter 11, "Resource Management," in the *Xt Intrinsics* manual. Note that a special resource converter to enable a user to specify a list of button labels will have to be written and registered. This procedure, called `CvtLabelsToPointer`, is included in the source code listing at the end of this chapter.

The Class Record

A class record structure for the multi-state button widget is defined in the private header file. Remember that there is only one class record for each widget class, but there may be many instances of a widget class. This class record is statically initialized and is part of the code file. The following code segment defines the class record for the multi-state button widget.

```
XwMultiButtonClassRec XwmultButtonClassRec = {
  {
/* core_class fields */
    /* superclass          */    (WidgetClass) &XwbuttonClassRec,
    /* class_name          */    "MultiButton",
    /* widget_size         */    sizeof(XwMultiButtonRec),
    /* class_initialize    */   ClassInitialize,
    /* class_part_init     */   NULL,
    /* class_inited        */     FALSE,
    /* initialize          */     Initialize,
```

```
    /* initialize_hook    */    NULL,
    /* realize            */     _XwRealize,
    /* actions            */     actionsList,
    /* num_actions        */     XtNumber(actionsList),
    /* resources   */    resources,
    /* num_resources      */     XtNumber(resources),
    /* xrm_class   */    NULLQUARK,
    /* compress_motion    */     TRUE,
    /* compress_exposure  */     TRUE,
    /* compress_enterlv   */    TRUE,
    /* visible_interest   */     FALSE,
    /* destroy            */     NULL,
    /* resize             */     Resize,
    /* expose             */     Redisplay,
    /* set_values         */     SetValues,
    /* set_values_hook    */    NULL,
    /* set_values_almost  */    XtInheritSetValuesAlmost,
    /* get_values_hook  */        NULL,
    /* accept_focus       */     NULL,
    /* version          */        XtVersion,
    /* callback_private */       NULL,
    /* tm_table         */       defaultTranslations,
    /* query_geometry   */        NULL,
  },
/* XwPrimitive Class Part */
 {
   /* border_highlight proc */          NULL,
   /* border_unhighlight proc */        NULL,
   /* selection proc */                 NULL,
   /* release proc */                   NULL,
   /* toggle proc */                     NULL,
   /* keyboard focus translations */    NULL,
 }

/*
   NOTE that XwButton and XwMultiButton have no fields
   of interest and thus are ignored.
*/

};
WidgetClass XwmultiButtonWidgetClass = (WidgetClass)&XwmultiButtonClassRec;
```

Note that many of the fields are NULL, meaning that the multi-state button widget does not use them. Refer to chapter 2, "Widgets," in the *Xt Intrinsics* manual for pointers to the sections that describe the above fields.

The Class Initialization Procedure

The first time an instance of a widget class (or an instance of a subclass widget) is created, the class initialize procedure for that widget is called. It allows the widget class to set up any fields or compute any values it will need to make instances of itself. It also provides the means for widgets to register any special resource converter routines. For the multi-state button widget, a special resource converter will be needed to handle lists of button labels defined in resource files such as the `.Xdefaults` file. This procedure, `CvtLabelsToPointer`, is included in the source code listing at the end of this chapter.

The Initialize Procedure

The invocation of the initialize procedure is usually the first indication a widget class has that an instance of itself is being created. When this procedure is invoked, the widget class can validate resources, compute size, and set any other needed fields. For the multi-state button widget, the initialize procedure will do the following:

- If the number of labels (`multi_button.num_labels`) is equal to or less than 0, it will be set to 1.
- If the pointer to the array of labels is NULL, the button's name (found in `core.name`) will be used as the single label. If the name is NULL, there will still be a pointer to a NULL label. If the name is not NULL, space will be allocated for the array of label pointers as well as the labels. The labels will then be copied into this space. Thus, all data is kept in the widget's data space instead of allowing some of it to reside in the application's data space. This is standard widget programming practice.
- If the height and width have the default values of zero, an ideal height and width will be calculated. The height will be based on the settings of the internal height (`button.internal_height`), the highlight thickness (`primitive.highlight_thickness`), and the font height (`button.font`). The width will be based on the settings of internal width, the highlight thickness, and the width of the longest label for the button.
- The button label will be set to display the first string in the sequence of labels provided by the application. The button state will be set to zero to reflect the label sequence.

The Redisplay Procedure

The redisplay procedure is probably the most difficult routine to write for this widget. Like other X Widget buttons, the multi-state button can display a shadow border to give it a three-dimensional illusion. For this reason, we distinguish between having to redraw only the button face and the entire button, shadows and all. Thus, the redisplay routine actually consists of two procedures:

- `RedisplayButtonFace`. The select and rotate procedures call `RedrawButtonFace` with a flag set so that only the button face will be redrawn.
- `Redisplay`. `Redisplay` simply calls `RedrawButtonFace` with the flag set to redraw the entire button face.

The rest of this section details the actions of the `RedrawButtonFace` procedure.

To correctly design and implement this routine you must remember a key feature of the Xt Intrinsics: a widget does not control its height and width. Thus, the multi-state button widget can compute an *optimum* height and width, but the *actual* height and width it receives is totally dependent on its parent. This leads to two possible cases:

- The button is too small to correctly display the entire button label.
- The button is much larger than necessary to display the label.

In the first case we will need to clip the label (potentially in both its height and width) to prevent it from overwriting the button shadow and destroying the three-dimensional illusion. Thus, the first thing the `RedrawButtonFace` procedure does is to see if it will need to clip either the height or the width of the label. The button is then displayed, and if the label needs to be clipped it is done at this point. Note that the clipping could have been accomplished by setting clipping rectangles in the graphic context (GC) that performed the label drawing, but this would have introduced a potentially unlimited number of new GCs to the server. The Xt Intrinsics always attempt to minimize the number of GCs by caching them and allowing widgets to share them. Setting clipping rectangles would have necessitated unique GCs for each button.

Finally, the button shadow is drawn (if `primitive.shadow_on` is true) and the button is highlighted or unhighlighted as needed.

The SetValues Procedure

The `SetValues` procedure is invoked whenever an application executes an `XtSetValues` to the multi-state button widget or one of its subclasses. `XtSetValues` works by invoking the `SetValues` procedures of each of the multi-state button widget's superclasses (starting at core). Each of these `SetValues` procedures handles the resources created and managed by that class. Thus, the multi-state button widget's `SetValues` procedure handles the following:

- Any changes that may have been made to resources it created.
- If there have been changes, indicate to the intrinsics code whether or not the widget should be redisplayed.

A general discussion of the `SetValues` procedure can be found in chapter 11, "Resource Management," in the *Programming With the Xt Intrinsics* manual.

The multi-state button widget's `SetValues` procedure first checks to see if the number of labels has been changed. If so, the `multi_button.state` flag will be reset to 0 and a check made to insure that the new number of labels is at least equal to 1. If the pointer to the array of button labels has changed, then the old set of labels will be deleted, and space will be allocated for the new labels, which will then be stored. The same subroutine (`HandleLabelAllocation`) used in the initialize procedure is used to accomplish this.

Test to see if any fields have changed that would cause the button to seek a new size. Specifically these fields are:

- The button font.
- The button label. Note that if the `multi_button.label` fields have changed, the `HandleLabelAllocation` procedure will already have updated this field, ensuring

that the need for a resize is flagged.

- The highlight thickness.

However, even if one or more of these fields has changed, all of the following must be true before the widget will seek a new size:

- The recompute size flag in primitive must be set to TRUE.
- The height in the new copy of the widget must be the same as that in the current copy of the widget in order to seek a new height.
- The width in the new copy of the widget must be the same as that in the current copy of the widget in order to seek a new width.

Finally, TRUE is returned as the value of the `SetValues` procedure if any value that indicates that the button should be redisplayed has changed. Otherwise, FALSE is returned.

The Destroy Procedure

Whenever a widget class creates fields that allocate and manage blocks of memory, it must provide a destroy procedure to release that memory when the widget is destroyed. In the case of the multi-state button widget, an array of pointers and some number of strings are allocated. All of these must be released when the widget is destroyed.

The Resize

Whenever a widget is resized by its parent or by an application, its resize procedure will be invoked. It is the responsibility of this procedure to adjust the visible contents of the widget to reflect the new size. For the multi-state button widget, the new height and width are used to center the button label within the the new dimensions of the button.

7.2.4 Source Code

All of the source code necessary to implement the `XwMultiButtonWidgetClass` is listed on the following pages. These files can also be found in the directory `/usr/contrib/Xw/examples/MultiButton`. There is also a "README" file in this directory that contains up-to-date information. Be sure to read this file before proceeding.

```
/*************************************<+>*************************************
 ******************************************************************************
 **
 **   File:         MButtonP.h
 **
 **   Project:      X Widgets
 **
 **   Description:  Private include file for widgets which are
 **                 subclasses of multibutton or which need to
 **                 access directly the instance and class fields
 **                 of the multibutton widget.
 **
 **
 **    Copyright (c) 1988 by Hewlett-Packard Company
 **    All Rights Reserved.
 **
 ******************************************************************************
 *************************************<+>*************************************/

/*********************************************
 *
 *   No new fields need to be defined
 *   for the MultiButton widget class record
 *
 *********************************************/

typedef struct {int nada;} XwMultiButtonClassPart;


/*****************************************************
 *
 * Full class record declaration for MultiButton class
 *
 *****************************************************/
typedef struct _XwMultiButtonClassRec {
    CoreClassPart          core_class;
    XwPrimitiveClassPart  primitive_class;
    XwButtonClassPart     button_class;
    XwMultiButtonClassPart multi_button_class;
} XwMultiButtonClassRec;


extern XwMultiButtonClassRec XwmultiButtonClassRec;


/*********************************************
 *
 * New fields needed for instance record
 *
 *********************************************/

typedef struct _XwMultiButtonPart{
     String  *labels;
     int     num_labels;
     int     state;
     Boolean invert_on_select;
} XwMultiButtonPart;
```

```
/************************************************************************
 *
 * Full instance record declaration
 *
 ************************************************************************/

typedef struct _XwMultiButtonRec {
    CorePart          core;
    XwPrimitivePart  primitive;
    XwButtonPart     button;
    XwMultiButtonPart multi_button;
} XwMultiButtonRec;

/************************************<+>*************************************
 ************************************************************************
 **
 **   File:         MButton.h
 **
 **   Project:      X Widgets
 **
 **   Description: Public include file for applications using the
 **                multibutton widget.
 **
 **
 **    Copyright (c) 1988 by Hewlett-Packard Company
 **    All Rights Reserved.
 **
 **
 ************************************<+>*************************************/

/************************************************************************
 *
 * MultiButton Widget
 *
 ************************************************************************/


/*  Resources for MultiButton  */

#define XtNlabels         "labels"
#define XtCLabels        "Labels"
#define XtRLabels        "Labels"
#define XtNinvertOnSelect "invertOnSelect"
#define XtCInvertOnSelect "InvertOnSelect"
#define XtNnumLabels     "numLabels"
#define XtCNumLabels     "NumLabels"

extern WidgetClass XwmultiButtonWidgetClass;

typedef struct _XwMultiButtonClassRec *XwMultiButtonWidgetClass;
typedef struct _XwMultiButtonRec      *XwMultiButtonWidget;
```

```
/************************************<+>*************************************
 ******************************************************************************
 **
 **   File:         MButton.c
 **
 **   Project:      X Widgets
 **
 **   Description: Contains code for primitive widget class: MultiButton
 **
 **
 **    Copyright (c) 1988 by Hewlett-Packard Company
 **    All Rights Reserved.
 **
 ******************************************************************************
 ************************************<+>*************************************/

/*
 * Include files & Static Routine Definitions
 */

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <X11/Misc.h>
#include <Xw/Xw.h>
#include <Xw/XwP.h>
#include <Xw/MButtonP.h>
#include <Xw/MButton.h>


static void Redisplay();
static void RedrawButtonFace();
static Boolean SetValues();
static void ClassInitialize();
static void Initialize();
static void Select();
static void Rotate();
static void Resize();
static Boolean RecomputeSize();
static void Destroy();
static int HandleLabelAllocation();

/************************************<->*************************************
 *
 *
 *   Description:  default translation table for class: MultiButton
 *   -----------
 *
 *   Matches events with string descriptors for internal routines.
 *
 ************************************<->************************************/
```

```
static char defaultTranslations[] =
   "<Btn1Down>:              select() \n\
    <Btn1Up>:                rotate() \n\
    <EnterWindow>:           enter() \n\
    <LeaveWindow>:           leave() \n\
    <KeyUp>Select:           rotate() \n\
    <KeyDown>Select:         select()";


/************************************<->*************************************
 *
 *
 *   Description:  action list for class: MultiButton
 *   -----------
 *
 *   Matches string descriptors with internal routines.
 *   Note that Primitive will register additional event handlers
 *   for traversal.
 *
 ************************************<->************************************/

static XtActionsRec actionsList[] =
{
  {"select", (XtActionProc) Select},
  {"rotate", (XtActionProc) Rotate},
  {"enter", (XtActionProc) _XwPrimitiveEnter},
  {"leave", (XtActionProc) _XwPrimitiveLeave},
};


/*  The resource list for MultiButton  */

static XtResource resources[] =
{
   {
     XtNlabels, XtCLabels, XtRLabels, sizeof (caddr_t),
     XtOffset (XwMultiButtonWidget, multi_button.labels),
     XtRString, (caddr_t) NULL
   },

   {
     XtNnumLabels, XtCNumLabels, XtRInt, sizeof (int),
     XtOffset (XwMultiButtonWidget, multi_button.num_labels),
     XtRString, "0"
   },
   {
     XtNinvertOnSelect, XtCInvertOnSelect, XtRBoolean, sizeof (Boolean),
     XtOffset (XwMultiButtonWidget, multi_button.invert_on_select),
     XtRString, "True"
   },
};
```

```
/************************************<->*************************************
 *
 *
 *   Description:  global class record for instances of class: MultiButton
 *   -----------
 *
 *   Defines default field settings for this class record.
 *
 ************************************<->************************************/

XwMultiButtonClassRec XwmultiButtonClassRec = {
  {
/* core_class fields */
    /* superclass          */    (WidgetClass) &XwbuttonClassRec,
    /* class_name          */    "MultiButton",
    /* widget_size         */    sizeof(XwMultiButtonRec),
    /* class_initialize   */    ClassInitialize,
    /* class_part_init    */    NULL,
    /* class_inited       */     FALSE,
    /* initialize          */    Initialize,
    /* initialize_hook    */    NULL,
    /* realize             */     _XwRealize,
    /* actions             */     actionsList,
    /* num_actions         */     XtNumber(actionsList),
    /* resources   */     resources,
    /* num_resources       */     XtNumber(resources),
    /* xrm_class   */     NULLQUARK,
    /* compress_motion     */     TRUE,
    /* compress_exposure  */      TRUE,
    /* compress_enterlv   */     TRUE,
    /* visible_interest    */     FALSE,
    /* destroy             */     Destroy,
    /* resize              */     Resize,
    /* expose              */     Redisplay,
    /* set_values          */     SetValues,
    /* set_values_hook    */     NULL,
    /* set_values_almost  */     XtInheritSetValuesAlmost,
    /* get_values_hook  */        NULL,
    /* accept_focus        */     NULL,
    /* version          */        XtVersion,
    /* callback_private */       NULL,
    /* tm_table         */       defaultTranslations,
    /* query_geometry   */        NULL,
  }
};
WidgetClass XwmultiButtonWidgetClass = (WidgetClass)&XwmultiButtonClassRec;
```

```
/************************************<->*************************************
 *
 *  Select (w, event)
 *
 *   Description:
 *   -----------
 *     Invert or change highlight (depending on setting of shadow_on
 *     flag.
 *     Issue any select callbacks and give them the current
 *     state value.
 *
 *   Inputs:
 *   ------
 *     w           =   widget instance that was selected.
 *     event       =   event record
 *
 ************************************<->*************************************/

static void Select (w,event)
Widget w;
XEvent *event;

{
   XwMultiButtonWidget mb = (XwMultiButtonWidget) w;

   mb->button.set = TRUE;

   RedrawButtonFace (w, event, FALSE);
   XFlush (XtDisplay(w));
   XtCallCallbacks (w, XtNselect, (caddr_t) mb->multi_button.state);

}


/************************************<->*************************************
 *
 *  Rotate (w, event)
 *
 *   Description:
 *   -----------
 *     Mark button as not set, rotate label (if there are any to
 *     rotate) Generate unselect callbacks and give them the new
 *     state.
 *
 *   Inputs:
 *   ------
 *     w           =   widget instance that was selected.
 *     event       =   event record
 *
 ************************************<->*************************************/

static void Rotate(w,event)
Widget w;
XEvent *event;
```

```
{
   XwMultiButtonWidget mb = (XwMultiButtonWidget) w;
   int newState = (mb->multi_button.state+1) % mb->multi_button.num_labels;

   mb->button.set = FALSE;
   mb->button.label= mb->multi_button.labels[newState];
   mb->multi_button.state = newState;
   _XwSetTextWidthAndHeight(mb);
   Resize(w);

   RedrawButtonFace (w, event, FALSE);
   XFlush(XtDisplay(w));
   XtCallCallbacks (w, XtNrelease, (caddr_t) newState);
}


/*************************************<->*************************************
 *
 *  Initialize
 *
 *   Description:
 *   -----------
 *    If the core height and width fields are set to 0, treat that as a flag
 *    and compute the optimum size for this button.  Then using whatever
 *    the core fields are set to, compute the text placement fields.
 *    Make sure that the label location field is properly set for the
 *    Resize call.
 *
 *
 *   Inputs:
 *   ------
 *     request           =       request widget, old data.
 *
 *     new               =       new widget, new data; cumulative effect
 *                               of initialize procedures.
 *
 *************************************<->*************************************/
static void Initialize (request, new)
Widget request, new;

{
   XwMultiButtonWidget mb = (XwMultiButtonWidget) new;
   int maxWidth = 0;

   /*************************************************************************
       Needed width:
        2 * highlight thickness +
        2 * internal width (padding between label and button) +
        Max(pixel width of labels)

      Needed height:
       2 * highlight thickness +
       2 * internal height (padding) +
       label height

    *************************************************************************/
```

```
    maxWidth = HandleLabelAllocation(mb);

   if (request->core.width == 0)  mb->core.width =  maxWidth +
              2 * ( mb->button.internal_width +    /* white space */
                    mb->primitive.highlight_thickness);

   if (request->core.height == 0) mb->core.height = mb->button.label_height +
       2 * (mb->button.internal_height + mb->primitive.highlight_thickness);

   Resize(new);
}


/*************************************<->*************************************
 *
 *  CvtLabelsToPointer
 *
 *   Description:
 *   -----------
 *     Convert a string containing button labels into an array of pointers
 *     to a  sequence of  labels.  Labels appear in the input string
 *     surrounded by double quotes.
 *
 *     NOTE that this routine will not handle more than 20 labels
 *     or more than 400 characters.
 *
 *
 *************************************<->*************************************/

#define MAXLABELS 20
#define MAXCHARS  400

static char storage[MAXCHARS];
static char * labels[MAXLABELS];
static char * labelsPtr = labels;


static void CvtLabelsToPointer(args, num_args, fromVal, toVal)
       XrmValuePtr args;
       int * num_args;
       XrmValuePtr fromVal;
       XrmValuePtr toVal;
{

       char * instr = (char *) (fromVal->addr);
       char * str_pos = storage;
       int i;

   /* Fail Safe: in case we get garbage return NULL */

       (*toVal).size = sizeof (caddr_t);
       (*toVal).addr = (caddr_t) &labelsPtr;
       for (i=0; i<MAXLABELS; i++) labels[i]=NULL;

       i=0;
```

```
      /* We'll only look for MAXLABELS */
       while (i < MAXLABELS)
          {
            /* Find beginning of label or end of input */
            while(*instr != '\0' && *instr != '"') instr++;

            if (*instr == '\0')    return;

            instr++;
            labels[i] = str_pos;

            /* Move string into storage space*/
            while (*instr != '"' && *instr != '\0')
               *str_pos++ = *instr++;

            if (*instr == '\0')
                XtError("Improper definition for MultiButton labels resource.");

           /* Append null to end of string, step beyond '"' marking end
            * of this label, increment "i" our label counter.
            */
            *str_pos++ = '\0';
            instr++;
            i++;

          }

}


/*************************************<->*************************************
 *
 *  ClassInitialize
 *
 *   Description:
 *   -----------
 *    Set fields in primitive class part of our class record so that
 *    the traversal code can invoke our button select/unselect procedures.
 *    Register specialized resource converter for this widget class.
 *
 *************************************<->*************************************/

static void ClassInitialize()
{
   XwmultiButtonClassRec.primitive_class.select_proc = (XtWidgetProc) Select;
   XwmultiButtonClassRec.primitive_class.release_proc = (XtWidgetProc) Rotate;
   XtAddConverter(XtRString, XtRLabels, CvtLabelsToPointer, NULL, 0);
}
```

```
/************************************<->*************************************
 *
 *  Redisplay (w, event)
 *
 *   Description:
 *   -----------
 *     Cause the widget, identified by w, to be redisplayed.
 *
 *
 *   Inputs:
 *   ------
 *     w = widget to be redisplayed;
 *     event = event structure identifying need for redisplay on this
 *             widget.
 *
 ************************************<->************************************/

static void Redisplay (w, event)
Widget w;
XEvent *event;

{
    RedrawButtonFace (w, event, TRUE);
}


static void RedrawButtonFace (w, event, all)
XwMultiButtonWidget w;
XEvent *event;
Boolean all;

{
   register XwMultiButtonWidget mb = (XwMultiButtonWidget) w;
   int available_height, available_width;
   Boolean clipHeight, clipWidth;

   /* COMPUTE SPACE AVAILABLE FOR DRAWING LABEL */

   available_width = Max(0,mb->core.width - 2*(mb->button.internal_width +
                                 mb->primitive.highlight_thickness));

   available_height = Max(0, mb->core.height - 2*(mb->button.internal_height +
                                 mb->primitive.highlight_thickness));



   /* SEE IF WE NEED TO CLIP THIS LABEL ON TOP AND/OR BOTTOM */

   if (mb->button.label_width > available_width)
       clipWidth = True;
   else
       clipWidth = False;
```

```
if (mb->button.label_height > available_height)
    clipHeight = True;
else
    clipHeight = False;


/* COMPUTE & DRAW MULTIBUTTON */


/* COMPUTE x LOCATION FOR STRING & DRAW STRING                    */
/* Draw only if all or the multibutton is set to invert the text  */
/* when it is selected and unselected.                            */


if (mb->button.label_len > 0 &&
    (all || mb -> multi_button.invert_on_select))
{
 XFillRectangle (XtDisplay(w), XtWindow(w),
                      ((mb->button.set && mb->multi_button.invert_on_select)
                                 ? mb->button.normal_GC
                                 : mb->button.inverse_GC),
                   w -> primitive.highlight_thickness + 1,
                   w -> primitive.highlight_thickness + 1,
                   w->core.width-2 * w->primitive.highlight_thickness-2,
                   w->core.height-2 * w->primitive.highlight_thickness-2);

  XDrawString(XtDisplay(w), XtWindow(w),
               ((mb->button.set && mb->multi_button.invert_on_select)
                                 ? mb->button.inverse_GC
                                 : mb->button.normal_GC),
              ((mb->core.width + 1 - mb->button.label_width) / 2),
               mb->button.label_y,  mb->button.label,
               mb->button.label_len);


 if (clipWidth)
   {
     XClearArea (XtDisplay(w), XtWindow(w), 0,0,
                  (mb->primitive.highlight_thickness +
                    mb->button.internal_width), mb->core.height, FALSE);

     XClearArea (XtDisplay(w), XtWindow(w),
                  (mb->core.width - mb->primitive.highlight_thickness -
                    mb->button.internal_width),0,
                  (mb->primitive.highlight_thickness +
                    mb->button.internal_width), mb->core.height, FALSE);
   }
```

```
    if (clipHeight)
      {
        XClearArea (XtDisplay(w), XtWindow(w), 0,0, mb->core.width,
                    (mb->primitive.highlight_thickness +
                      mb->button.internal_height), FALSE);
        XClearArea (XtDisplay(w), XtWindow(w), 0,
                    (mb->core.height - mb->primitive.highlight_thickness -
                      mb->button.internal_height), mb->core.width,
                    (mb->primitive.highlight_thickness +
                      mb->button.internal_height), FALSE);
      }
  }


  /* NOW DRAW SHADOW */

if (w -> primitive.shadow_on)
  _XwDrawShadow (XtDisplay (w), XtWindow (w),
                 ((mb->button.set) ?
                      w -> primitive.bottom_shadow_GC :
                      w -> primitive.top_shadow_GC),
                 ((mb->button.set) ?
                      w -> primitive.top_shadow_GC :
                      w -> primitive.bottom_shadow_GC),
                 w -> primitive.highlight_thickness - 2,
                 w -> primitive.highlight_thickness - 2,
                 w->core.width - 2 * w->primitive.highlight_thickness + 4,
                 w->core.height - 2 * w->primitive.highlight_thickness + 4);


  /*
   * Draw traversal/enter highlight if actual exposure or
   * if we had to clip text area
   */

  if (all || clipWidth || clipHeight)
  {
     if (mb->primitive.highlighted)
        _XwHighlightBorder(w);
     else if (mb->primitive.display_highlighted)
        _XwUnhighlightBorder(w);
  }
}
```

```
/************************************<->*************************************
 *
 *  SetValues(current, request, new)
 *
 *   DESCRIPTION:
 *   -----------
 *     This is the set values procedure for the multi_button class.  It is
 *     called last (the set values routines for its superclasses are called
 *     first).
 *
 *
 *   Inputs:
 *   ------
 *    current = original widget;
 *    request = copy of widget as requested by application;
 *    new = copy of request which reflects changes made to it by
 *          set values procedures of its superclasses;
 *
 ************************************<->*************************************/

static Boolean SetValues(current, request, new)
Widget current, request, new;

{
    XwMultiButtonWidget curmb = (XwMultiButtonWidget) current;
    XwMultiButtonWidget newmb = (XwMultiButtonWidget) new;
    Boolean  flag = FALSE;    /* our return value */
    int maxWidth = -1;
    XFontStruct  *fs = newmb->button.font;
    int i;


   /* Validate fields unique to MultiButton */

    if (curmb->multi_button.num_labels != newmb->multi_button.num_labels)
      {
        /* Reset state to 0 */
        newmb->multi_button.state = 0;

       /* Don't allow new value to be illegal */
        if (newmb->multi_button.num_labels < 1)
                newmb->multi_button.num_labels = 1;

      }


    if (curmb->multi_button.labels != newmb->multi_button.labels)
      {
       Destroy(curmb);   /* free up pointers and strings */
       maxWidth = HandleLabelAllocation(newmb);
       flag = TRUE;
      }
```

```
    /*************************************************************************
     * Calculate the window size:  The assumption here is that if
     * the width and height are the same in the new and current instance
     * record that those fields were not changed with set values.  Therefore
     * its okay to recompute the necessary width and height.  However, if
     * the new and current do have different width/heights then leave them
     * alone because that's what the user wants.  Also, use the
     * RecomputeSize procedure (defined below) to test if we should
     * recompute the size.
     *************************************************************************/
    if ((curmb->core.width == newmb->core.width) &&
         (RecomputeSize(current, new)))
     {
       if (maxWidth < 0)
          for (i=0; i<newmb->multi_button.num_labels; i++)
              maxWidth = Max(maxWidth,
                             XTextWidth(fs, newmb->multi_button.labels[i]));

        newmb->core.width = maxWidth +  2*(newmb->button.internal_width +
                                           newmb->primitive.highlight_thickness);
        flag = TRUE;
     }

    if ((curmb->core.height == newmb->core.height) &&
         (RecomputeSize(current, new)))
     {
        newmb->core.height =
            newmb->button.label_height + 2*(newmb->button.internal_height +
                    newmb->primitive.highlight_thickness);
        flag = TRUE;
     }

   return(flag);
}



/****************************************<->***************************************
 *
 *  Resize(w)
 *
 *   Description:
 *   -----------
 *     Recompute location of button text (center text in the button
 *     face).
 *
 *   Inputs:
 *   ------
 *     w  = widget to be resized.
 *
 *
 ****************************************<->***************************************/
```

```
static void Resize(w)
    Widget w;
{
    XwMultiButtonWidget mb = (XwMultiButtonWidget) w;

    mb->button.label_x = (mb->core.width + 1 - mb->button.label_width) / 2;

    mb->button.label_y =
       (mb->core.height - mb->button.label_height) / 2
        + mb->button.font->max_bounds.ascent;
}



/*************************************<->*************************************
 *  Boolean
 *  RecomputeSize(current, new)
 *
 *   Description:
 *   -----------
 *     Used during SetValues.
 *
 *     If the font has changed OR the label has changed OR
 *     the internal spacing has changed OR the highlight
 *     thickness has changed AND the recompute flag is TRUE
 *     (in the new widget) return TRUE, else return FALSE.
 *
 *
 *   Inputs:
 *   ------
 *    current = current version of widget
 *    new = new version of widget
 *
 *   Outputs:
 *   -------
 *     TRUE if resize is needed and okay, FALSE otherwise.
 *
 *************************************<->*************************************/
static  Boolean RecomputeSize(current, new)
  XwButtonWidget current, new;
{
  if (((new->button.font != current->button.font) ||
     (new->button.label != current->button.label) ||
     (new->primitive.highlight_thickness !=
         current->primitive.highlight_thickness) ||
     (new->button.internal_height != current->button.internal_height) ||
     (new->button.internal_width != current->button.internal_width)) &&
     (new->primitive.recompute_size == TRUE))
         return(TRUE);
  else
         return(FALSE);
}
```

```
/************************************<->*************************************
 *
 *  Destroy (mb)
 *
 *   Description:
 *   -----------
 *     Free up the memory allocated especially for the
 *     multibutton part of the widget instance record.
 *
 *
 *   Inputs:
 *   ------
 *     mb = multibutton widget.
 *
 ************************************<->*************************************/
static void Destroy(mb)
  XwMultiButtonWidget mb;
{
   int i;

   /* Free each of the labels */
   for (i=0; i < mb->multi_button.num_labels; i++)
     XtFree(mb->multi_button.labels[i]);

  /* Free the array of pointers to the labels */
   XtFree((char *)mb->multi_button.labels);

}
```

```
/************************************<->*************************************
 *  int
 *  HandleLabelAllocation(mb)
 *
 *   Description:
 *   -----------
 *     If no labels have been provided, use the button name as
 *     the single label.  Allocate an array of pointers to strings
 *     to hold pointer to button labels; then allocate space for
 *     each button label and copy label to this space.  Compute
 *     the widest (in pixels) label and return this to figure to
 *     the caller (it will be used to compute the optimum width
 *     for the button).  Set the multibutton state to 0 and put
 *     a pointer to the corresponding label into button.label.
 *
 *
 *   Inputs:
 *   ------
 *     mb = multibutton whose labels are to be allocated.
 *
 *   Outputs:
 *   -------
 *     returns width (in pixels) of widest label.
 *
 ************************************<->*************************************/
static int HandleLabelAllocation(mb)
   XwMultiButtonWidget mb;
{
   String * labels;
   int i;
   int maxWidth = 0;
   XFontStruct  *fs = mb->button.font;

   /* If user has not given us any labels then try to use
    * widget's name as a label (of course this too could be NULL but
    * it still shouldn't hurt us).
    */
   if (mb->multi_button.labels == NULL)
       mb->multi_button.labels = &(mb->core.name);

   /* There is ALWAYS one label, even if that label is NULL (i.e., the
    * application provides no names and the button's name is NULL).
    */
   if (mb->multi_button.num_labels < 1)
       mb->multi_button.num_labels = 1;

   /* Allocate array of label pointers */
   labels = (String *) XtMalloc(sizeof(char *)* mb->multi_button.num_labels);
```

```
    /* Allocate space for each label and copy pointer to new string
     * into array allocated above. Also compute, in pixels, width of
     * widest label.
     */
    for (i=0; i< mb->multi_button.num_labels; i++)
      {
        labels[i]= (char *)strcpy(
                                XtMalloc((unsigned)mb->multi_button.labels[i] + 1),
                                 mb->multi_button.labels[i]);
        maxWidth = Max(maxWidth, XTextWidth(fs,labels[i], XwStrlen(labels[i])));
      }


    /* Now put correct label, set state and compute label location. */
    mb->multi_button.labels = labels;
    mb->button.label = labels[0];
    mb->multi_button.state = 0;
    _XwSetTextWidthAndHeight(mb);

   /* Return width (in pixels) of widest label */
    return(maxWidth);
  }
```

7.2.5 Putting the New Widget Together

Now that you have a new widget, you need a test program to exercise its features. Listed below is a simple program that will place a static text label and multi-state button into a bulletin board widget. Turning on the multi-state button will change the color used to draw the static text. The source code for this program can be found in the file `multiTest.c` in the directory `/usr/contrib/Xw/examples/MultiButton`.

```
/*****************************************************************
  This is a simple program that creates a bulletin board manager
  which contains a static text label and a multibutton.

*****************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/SText.h>
#include <Xw/BBoard.h>
#include <Xw/MButton.h>
#include <X11/Xresource.h>

Widget bbox, colorbox, mbutton;
Widget toplevel, outer_box, pbutton;
Arg myArgs[10];
Pixel color;
```

```
Pixel CvtStringToPixel(src_string)
    char *  src_string;
{
    XColor aColor, bColor;

    XAllocNamedColor(XtDisplay(toplevel),
                  DefaultColormapOfScreen(XtScreen(toplevel)),
                   src_string, &aColor, &bColor);

    return(aColor.pixel);
}



/*  Set up callbacks for buttons */


void ToggleColor(w, closure, call_data)
    Widget      w;
    caddr_t     closure;
    caddr_t     call_data;
{
    int state = (int) call_data;

    switch(state)
      {
      case 0 :  color = CvtStringToPixel("red");
                break;
      case 1:   color = CvtStringToPixel("black");
                break;
      case 2:   color = CvtStringToPixel("green");
               break;
      default: color = CvtStringToPixel("orange");
       }
   XtSetArg(myArgs[0], XtNforeground, (XtArgVal) color);
   XtSetValues(colorbox, (ArgList)myArgs, 1);
}

static XtCallbackRec colorCallback[] =
{
   { ToggleColor, (caddr_t) NULL },
   { NULL,         (caddr_t) NULL },
};



void main(argc, argv)
    unsigned int argc;
    char **argv;
{
    static char * labels[] =
      { "RED", "PAINT IT BLACK, YOU DEVIL", "GREEN", "ORANGE" };
```

```
    toplevel = XtInitialize(
        argv[0], "XTest1", NULL, 0, &argc, argv);

    XtSetArg(myArgs[0],XtNlayout, XwIGNORE);
    XtSetArg(myArgs[1],XtNwidth, 500);
    XtSetArg(myArgs[2],XtNheight, 300);
    outer_box = XtCreateManagedWidget
                ("bb1", XwbulletinWidgetClass, toplevel,
                          (ArgList)myArgs, 3);


    XtSetArg(myArgs[0],XtNwidth, 300);
    XtSetArg(myArgs[1],XtNalignment, XwALIGN_CENTER);
    XtSetArg(myArgs[2],XtNstring,
                (XtArgVal) "One Of Many Button Box");
    colorbox = XtCreateManagedWidget
               ("text",XwstatictextWidgetClass, outer_box,
                          (ArgList)myArgs, 3);

    XtSetArg(myArgs[0],XtNx, 10);
    XtSetArg(myArgs[1],XtNy, 100);
    XtSetArg(myArgs[2],XtNnumLabels, 4);
    XtSetArg(myArgs[3],XtNselect, (XtArgVal) colorCallback);
    XtSetArg(myArgs[4],XtNlabels, labels);
    mbutton = XtCreateManagedWidget
               ("MultiButton!", XwmultiButtonWidgetClass, outer_box,
                          (ArgList)myArgs, 5);



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

You can compile this program on an HP 9000 Series 300 machine by using following command:

```
cc test.c -o test -Wc,-Nd2000 -Wc,-Ns2000 -lXw -lXt -lX11
```

7.3 Widget Classing

Widget classes with similar functions and parallel programmatic interface can be associated into groups known as tasks, and each class within a task is called a view. Each task group has a unique pointer associated with it. If widgets are created by referencing a task pointer instead of a widget class pointer, the precise determination of which view to be instantiated can be controlled through resource settings. This feature is called *Widget Classing*. You could use widget classing to allow the application user to choose which type of button widget should be used by the application.

To use widget classing, application programs must include the file <Xw/WClassing.h>. This file provides the ability to associate widget classes into task groups and select views through resource settings. The following sections will show you how to modify <XwWClassing.h> to add and delete task groups and views within task groups.

7.3.1 Implementing Widget Classing

Widget classing is implemented by replacing XtCreateWidget and XtCreateManagedWidget with routines that understand the difference between a widget class pointer and a task pointer. The following code is found at the end of <Xw/WClassing.h>.

```
/*
 * This allows us to trap the XtCreateWidget call in the application
 * code and preprocess the class pointer.  In this way we can get
 * complete resource specification.  We also can pass task pointers
 * in and magically turn them into widget class pointer.
 */
extern Widget XwCreateWidget();

#define XtCreateWidget(name,class,parent,args,num)\
            XwCreateWidget(name,class,parent,args,num)

/*
 * This is the trap for XtCreateManagedWidget.
 */
extern Widget XwCreateManagedWidget();

#define XtCreateManagedWidget(name,class,parent,args,num)\
            XwCreateManagedWidget(name,class,parent,args,num)
```

Since XtCreateManagedWidget consists of calls to XtCreateWidget and XtManageWidget, it follows that XwCreateManagedWidget consists of calls to XwCreateWidget and XtManageWidget. The following discussion will therefore only need to deal with XwCreateWidget.

XwCreateWidget examines the class argument to see if it is the same value as any of the task pointers contained in XwTasks (discussed below). If the class parameter is equal to one of the task pointers, XwCreateWidget will resolve the task pointer to a specific view and call XtCreateWidget using the widget class associated with the chosen view. If the class parameter is not equal to any task pointer contained in the XwTasks structure, XwCreateWidget assumes that the argument is a valid widget class and calls XtCreateWidget directly.

7.3.2 XwTasks

Tables 7-1 and 7-2 show the structure of `XwTasks`. The "View" column of table 7-1 is simply a pointer to the view structure that is shown in table 7-2.

TABLE 7-1. Task Table

Task	Task Class	View	Number Of Views
XwButtonTask	ButtonTask	.	2
XwImageEditTask	ImageEditTask	.	1
XwLayoutTask	LayoutTask	.	4
.	.	.	.
.	.	.	.

TABLE 7-2. View Table

View Name	Widget Class Pointer
XwPushButton	XwPushButtonWidgetClass
XwToggle	XwToggleWidgetClass

Each of the fields in the tables are defined below.

- **Task.** Task pointers are simply pointers to unique areas in memory. By pointing somewhere besides valid widget class structures, task pointers are guaranteed to never equal widget class pointers. Currently, uniqueness of reference is guaranteed by declaring an unused integer for each task group and then making the task pointer point to this unused integer. For the task "ButtonTask," the code is:

```
static int XwButtonTsp;
static int *XwButtonTask = &XwButtonTsp;
```

- **Task Class.** The resource class for this task.
- **View.** A pointer to the views for this task group.
- **Number of Views.** The number of views in this task group.
- **View Name.** The resource name for view.
- **Widget Class Pointer.** The widget class pointer for the view.

Because the addresses of widget class structures are not known at compile-time, the view tables cannot be initialized then. Run-time initialization is accomplished as a three-part process. First, the view table is declared. For ButtonTask, the declaration is:

```
    static XwViewTableEntry XwButtonViews[2];
```

For each view table there is an initialization function definition. The initialization function defined for ButtonTask is:

```
static XwWCViewLoadProc XwLoadButtonViews()
{
  XwLoadViewTable(XwButtonViews,0,
                        "XwPushButton",XwpushButtonWidgetClass);
  XwLoadViewTable(XwButtonViews,1,
                        "XwToggle",XwtoggleWidgetClass);
  }
```

At this point `XwtoggleWidgetClass` is considered a reference and is resolved by the linker. At runtime, after linker resolution, the widget class pointers are loaded into the structures. The programmer must insure that the needed include files have been included. For the above example, both <Xw/PButton.h> and <Xw/Toggle.h> must be included before the definition of `XwLoadButtonViews()`.

All initialization functions are simply repeated calls to `XwLoadViewTable`. `XwLoadViewTable` is a very simple function and is defined as follows:

```
void XwLoadViewTable(table,index,name,class)
    XwViewTableEntry    table[];
    int index;
    char    *name;
    WidgetClass class;
    {
      table[index].name = name;
      table[index].wClass = class;
    }
```

The last step in table initialization is to initialize a list of procedures. All procedures in this list are executed during the first call to `XwCreateWidget`. For the distributed widget classing code, the initialization table is:

```
XwWCViewLoadProc XwWCViewLoadProcs[] = {
 (XwWCViewLoadProc) XwLoadButtonViews,
 (XwWCViewLoadProc) XwLoadImageEditViews,
 (XwWCViewLoadProc) XwLoadLayoutViews,
 (XwWCViewLoadProc) XwLoadMenuViews,
 (XwWCViewLoadProc) XwLoadScrollViews,
 (XwWCViewLoadProc) XwLoadTextEditViews,
 (XwWCViewLoadProc) XwLoadTitleViews,
};
```

To modify the task groups, modifications must be made to the structures declared in <Xw/WClassing.h>. Deletion of views is the simplest modification. Perform the following steps to delete a view:

1. Reduce the size of the view table.

2. Remove the call to `XwLoadViewTable` that loads the undesired view.
3. Realign the view indices.

For example, to remove the `XwpushButtonWidgetClass` view from the `XwButtonTask` group, you would change

```
static XwViewTableEntry XwButtonViews[2];
```

to

```
static XwViewTableEntry XwButtonViews[1];
```

The definition of `XwLoadButtonViews()` would be changed from

```
static XwWCViewLoadProc XwLoadButtonViews()
{
XwLoadViewTable(XwButtonViews,0,
  "XwPushButton",XwpushButtonWidgetClass);
XwLoadViewTable(XwButtonViews,1,
  "XwToggle",XwtoggleWidgetClass);
}
```

to

```
static XwWCViewLoadProc XwLoadButtonViews()
{
   XwLoadViewTable(XwButtonViews,0,
   "XwToggle",XwtoggleWidgetClass);
}
```

Deleting task groups is a more complicated process. Perform the following steps to delete a task group:

1. Remove the declaration of the view table.
2. Remove the view load function from the initialization function list.
3. Remove the task table entry for the task.
4. Remove the definition of the view table initialization function.
5. Remove the declaration and initialization of the task pointer.

Perform the following steps to add a task group:

1. Declare a task pointer.
2. Initialize the task pointer to point to some known address that is not a widget class structure.
3. Make sure that all public include files for all views are included.

4. Declare a view table of an appropriate size.
5. Define a view table initialization function.
6. Add the view table initialization function to the initialization function list (XwWCViewLoadProcs).

The following sample code will add a task group called "Jetsons." This task group consists of George, Jane, Elroy, Judy, and Astro widget classes.

```
/* Steps one and two */
static int JetsonsTSP;
static int *JetsonsTask = &JetsonTSP;

/* Step three */
#include <Hana-Barbera/George.h>
#include <Hana-Barbera/Jane.h>
#include <Hana-Barbera/Elroy.h>
#include <Hana-Barbera/Judy.h>
#include <Hana-Barbera/Astro.h>

/* Step four */
static XwViewTableEntry Jetsons[5];

/* Step five */
static XwWCViewLoadProc LoadJetsons()
{
XwLoadViewTable(Jetsons,0,
  "Gecrge",GeorgeWidgetClass);
XwLoadViewTable(Jetsons,0,
  "Jane",JaneWidgetClass);
XwLoadViewTable(Jetsons,0,
  "Elroy",ElroyWidgetClass);
XwLoadViewTable(Jetsons,0,
  "Judy",JudyWidgetClass);
XwLoadViewTable(Jetsons,0,
  "Astro",AstroWidgetClass);
}
```

7.3.3 Using Resources

When XwCreateWidget recognizes the class parameter to be a task pointer, it assembles a class list and a name list in preparation for resource data base query. If the query is successful, the view table for the task is searched for a matching string and the widget class associated with that string is instantiated. If the query is unsuccessful, the widget class of the first view in the view table is instantiated by default. The class list is of the form:

```
<Class list of Parents>.<task_class>.View
```

Where:

- "Class list of Parents" is the list of classes of all parents of this widget in the widget hierarchy.
- "task_class" is the second field of the `TaskTableEntry` associated with the task pointer used in the call to `XwCreateWidget`.

The name list is of the form:

```
<Name list of Parents>.<name>.view
```

Where:

- "Name list of Parents" is the list of names of all parents of this widget in the widget hierarchy.
- "name" is the name parameter sent in to `XwCreateWidget`.

Consider the following program:

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/WClassing.h>

void main(argc,argv)
  unsigned   intargc;
  char  *argv[];
  {
  Widget                toplevel, button;

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

  XtCreateManagedWidget("button",XwButtonTask,
                          toplevel,NULL,NULL);

  XtRealizeWidget(toplevel);

  XtMainLoop();
}
```

Because `XwpushButtonWidgetClass` is the widget class of the first view in the view table, the default widget to be instantiated will be `XwpushButtonWidgetClass`. The default is used when no resource has been set or when an invalid resource has been set.

All of the following resource definitions will cause an `XwtoggleWidgetClass` widget to be instantiated:

```
*ButtonTask.View:    XwToggle
*ButtonTask.view:    XwToggle
*button.view:    XwToggle
*button.View:    XwToggle
```

All of the following resource definitions will cause an XwpushButtonWidgetClass widget to be instantiated:

```
*ButtonTask.View:     NonsensicalString
*ButtonTask.View:     XwPushButton
*ButtonTask.view:     XwPushButton
*button.view:     XwPushButton
*button.View:     XwPushButton
```

7.4 Summary

This chapter detailed the process of writing a new widget, including the use of widget classing. You can use the methods shown here to create other new widgets that you may need.

Reference Information

This section contains reference information about HP Widgets included with the X Window System. The entries are arranged alphabetically, with each entry beginning on its own "page 1."

MAN Pages	
Constraint(3X)	XwPopupMgr(3X)
Core(3X)	XwPrimitive(3X)
XwArrow(3X)	XwPulldown(3X)
XwBulletin(3X)	XwPushButton(3X)
XwButton(3X)	XwRegisterConverters(3X)
XwCascade(3X)	XwRowCol(3X)
XwCreateTile(3X)	XwSash(3X)
XwForm(3X)	XwScrollBar(3X)
XwFrame(3X)	XwScrolledWindow(3X)
XwImageEdit(3X)	XwStaticRaster(3X)
XwList(3X)	XwStaticText(3X)
XwManager(3X)	XwTextEdit(3X)
XwMenuButton(3X)	XwTitleBar(3X)
XwMenuMgr(3X)	XwToggle(3X)
XwMenuPane(3X)	XwValuator(3X)
XwMenuSep(3X)	XwVPanedWindow(3X)
XwMoveFocus(3X)	XwWorkSpace(3X)
XwPanel(3X)	

This page left blank intentionally.

NAME

Constraint - A description of the interface to constraint resources.

CLASSES

A sub-class of Core and Composite.

DESCRIPTION

When a constrained composite widget defines constraint resources, all of that widget's children effectively "inherit" all of those resources as their own. These constraint resources are set and read just the same as any other resources defined for the child. This resource "inheritance" extends exactly one generation down, or in other words only the first generation children of a constrained composite widget inherit that (the constrained composite) widget's constraint resources.

For example, Panel has three children, a child of class XwstatictextWidgetClass, a child of class XwrowColWidgetClass, and a child of class XwFormWidgetClass. All of these children inherit all of the constraint resources defined by XwpanelWidgetClass. None of the children of the rowCol child inherit any of the Panel constraint resources. None of the children of the Form inherit any of the Panel constraint resources, but all of the children of the Form do inherit all of the constraint resources defined by the XwformWidgetClass.

Because constraint resources are defined by the "parent" widgets and not the children, the child widgets never directly use the constraint resource data. Constraint resource data is instead used by the parents to attach child specific data to children.

SEE ALSO

CORE(3X), XWPANEL(3X)
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

CoreClass - the Xt Intrinsics core widget meta class

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>

CLASSES

All widgets are built from the Core class.

DESCRIPTION

The Core class is an Xt Intrinsics meta class. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other widget classes. It provides resources required by all widgets: x y location, width, height, window border width, and so on.

NEW RESOURCES

Core defines a set of resource types used by the programmer to specify the data for widgets which are subclasses of Core.

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

XtNancestorSensitive

This argument specifies whether the immediate parent of the widget will receive input events. Use the function XtSetSensitive if you are changing the argument to preserve data integrity (see XtNsensitive below).

XtNbackground

This argument specifies the background color for the widget.

XtNbackgroundPixmap

The application can specify a pixmap to be used for tiling the background. The first tile is place at the upper left hand corner of the widget's window.

XtNborderColor

This argument specifies the color of the border.

XtNborderPixmap

The application can specify a pixmap to be used for tiling the border. The first tile is place at the upper left hand corner of the border.

XtNborderWidth

This argument sets the width (in pixels) of the border that surrounds the widget's window on all four sides. A width of zero means no border will show. This argument is an

unsigned integer, so using a negative number to specify the width is not recommended.

XtNdepth
Determines how many bits should be used for each pixel in the widget's window. Programs should not change or set this, it will be set by the Xt Intrinsics when the widget is created.

XtNdestroyCallback
This is a pointer to a callback list containing routines to be called when the widget is destroyed.

XtNheight
This argument contains the height of the widget's window in pixels, not including the border area. Programs should not change this argument directly, but use geometry manager requests instead in order to ensure proper relationships with other widgets are maintained.

XtNmappedWhenManaged
If set to TRUE, the widget will be mapped (made visible) as soon as it is both realized and managed. If set to FALSE, the client is responsible for mapping and unmapping the widget. If the value is changed from TRUE to FALSE after the widget has been realized and managed, the widget is unmapped.

XtNsensitive
This argument determines whether a widget will receive input events. If a widget is sensitive, the Xt Intrinsic's Event Manager will dispatch to the widget all keyboard, mouse button, motion, window enter/leave, and focus events. Insensitive widgets do not receive these events. Use the function XtSetSensitive if you are changing the sensitivity argument. That way you ensure that if a parent widget has XtNsensitive set to FALSE, the ancestor-sensitive flag of all its children will be appropriately set.

XtNtranslations
This is a pointer to a translations list.

XtNwidth
This argument contains the width of the widget's window in pixels, not including the border area. Programs may request a width at creation or using XtSetValues, but such requests may not succeed due to layout requirements of the parent widget.

XtNx This argument contains the x-coordinate of the widget's upper left hand corner (excluding the border) in relation to its parent widget. Programs may request an x-coordinate at creation or using XtSetValues, but such requests may not succeed due to layout requirements of the parent widget.

XtNy This argument contains the y-coordinate of the widget's upper left hand corner (excluding the border) in relation to its parent widget. Programs may request a y-coordinate at creation or using XtSetValues, but such requests may not succeed due to layout requirements of the parent widget.

INHERITED RESOURCES
The Core class is the root class. It inherits no resources.

TRANSLATIONS
None

ACTIONS
None

ORIGIN
MIT.

SEE ALSO
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwarrowWidgetClass - the X Widget's arrow drawing widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Arrow.h>
```

CLASSES

The Arrow widget is built from the Core and Primitive classes.

The widget class to use when creating an arrow is **XwarrowWidgetClass.** The class name for this widget is **Arrow.**

DESCRIPTION

The Arrow widget supports drawing of an arrow within the bounds of its window. It uses the primitive widget's border highlighting and shadow drawing routines.

The arrow can be drawn in the directions of up, down, left and right. The Arrow widget also supports two types of callbacks: Button selections, and Button releases.

NEW RESOURCES

The Arrow widget defines a set of resources used by the programmer to specify the data for the arrow. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by the Arrow widget.

Arrow Resource Set			
Name	**Class**	**Type**	**Default**
XtNarrowDirection	XtCArrowDirection	int	arrow_up

XtNarrowDirection

This resource is the means by which the arrow direction is set. It can be defined in either of two ways: Through the .Xdefaults file by the strings "arrow_up," "arrow_down," "arrow_left," and "arrow_right." Within an arg list for use in XtSetValues() by the defines XwARROW_UP, XwARROW_DOWN, XwARROW_LEFT and XwARROW_RIGHT.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to this widget.

TRANSLATIONS

Input to the Arrow widget is driven by the mouse buttons. The Primitive class resources of XtNselect and XtNrelease define the callback lists used by the Arrow widget. Thus, to receive input from an arrow, the application adds callbacks to the arrow using these two resource types. The default translation set for the Arrow widget is as follows.

XWARROW(3X)

<Btn1Down>:	select()	
<Btn1Up>:	release()	
<EnterWindow>:	enter()	
<LeaveWindow>:	leave()	
<KeyDown>Select:	select()	HP "Select" key
<KeyUp>Select:	unselect()	HP "Select" key

ACTIONS

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be highlighted. Otherwise no action is taken.

leave: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be unhighlighted. Otherwise no action is taken.

release:

Release redraws the arrow in its normal mode and calls its primitive XtNrelease callbacks.

select: Selections occurring on an arrow cause the arrow to be displayed as selected and its primitive XtNselect callbacks are called.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWCREATETILE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWBULLETIN(3X)

NAME

XwbulletinWidgetClass - the X Widgets bulletin board manager widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/BBoard.h>
```

CLASSES

The bulletin board manager widget is built from the Core, Composite, Constraint and Manager classes. Note that the Constraint fields are not used in this widget and so are not listed in the resource tables below. Also, since the Composite class contains no resources that the user can set, there is no table for Composite class resources.

The widget class to use when creating a bulletin board is **XwbulletinWidgetClass.** The class name is **BulletinBoard**.

DESCRIPTION

The bulletin board manager widget is a composite widget that enforces no ordering on its children. It is up to the application to specify the x and y coordinates of the children inserted into this widget, otherwise they will all appear at (0,0).

This manager widget supports 3 different layout policies: minimize (the default), maximize and ignore. When the layout policy is set to minimize, the manager will create a box that is just large enough to contain all of its children, regardless of any provided width and height values. The ignore setting forces the manager to honor its given width and height, it will not grow or shrink in response to the addition, deletion or altering of its children. When set to the maximize setting, the BulletinBoard widget will ask for additional space when it needs it, but will not give up extra space.

The bulletin board manager also implements the X Widgets keyboard interface.

No callbacks are defined for this manager.

NEW RESOURCES

The bulletin board manager widget class does not define any additional resources; all necessary resources are present in its superclasses. The programmer should refer to the man pages for the bulletin board's superclasses to determine the resources that can be set and the defaults settings for these resources.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

XWBULLETIN(3X)

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNforeground	XtCForeground	Pixel	Black
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNtraversalOn	XtCTraversalOn	Boolean	TRUE

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at create time or during a call to XtSetValues, the XwManager superclass will automatically augment the bulletin board manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWBUTTON(3X)

NAME

XwbuttonWidgetClass - X Widget Button MetaClass

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

CLASSES

The Button widget is built from the Core and Primitive classes.

DESCRIPTION

The Button class is an X Widget meta class. It is never instantiated as a widget. It provides a set of resources that are needed by a variety of other X Widgets (for example: XwtoggleWidgetClass and XwpushButtonWidgetClass).

NEW RESOURCES

The XwButtonClass defines a set of resource types used by the programmer to specify the data for widgets that are subclasses of XwButtonClass. To specify any of these resources within the .Xdefaults file, drop the XtN prefix from the resource name. For example, XtNfont becomes font.

Button Resource Set			
Name	**Class**	**Type**	**Default**
XtNborderWidth	XtCBorderWidth	int	0
XtNfont	XtCFont	XFontStruct *	Fixed
XtNhSpace	XtCHSpace	int	2
XtNlabel	XtCLabel	caddr_t	NULL
XtNlabelLocation	XtCLabelLocation	int	right
XtNsensitiveTile	XtCSensitiveTile	int	75_foreground
XtNset	XtCSet	Boolean	FALSE
XtNvSpace	XtCVSpace	int	2

XtNborderWidth

This redefines the core class default border width from 1 pixel to 0 pixels.

XtNfont

The application may define the font to be used when displaying the button string. Any valid X11 font may be used.

XtNhSpace

The application may determine the number of pixels of space left between the left side of the button and the leftmost part of the button label, and between the rightmost part of the button label and the right side of the button.

XtNlabel

The application may define the button label by providing a pointer to a null terminated character string. If no label is provided the class name of the widget will be used.

XtNlabelLocation

For those buttons that have a separate graphic, this field specifies whether the label should appear to the left or to the right of that graphic. The acceptable values are the defines XwRIGHT (the default) and XwLEFT.

XtNsensitiveTile

The application can determine the mix of foreground and background that will be used to draw text to show insensitivity. The #defines for setting the values through an arg list and the strings to be used in the .Xdefault file are described in XwCreateTile(3X). The default is Xw75_FOREGROUND which is a 75/25 mix of foreground and background colors.

XtNset If set to true the button will display itself in its selected state. This is useful for showing some conditions as active when a set of buttons appear.

XtNvSpace
The application may determine the number of pixels of space left between the top of the button and the top of the button label, and between the bottom of the label and the bottom of the button.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwcascadeWidgetClass - the X Widgets popup and pulldown menupane widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/Cascade.h>
```

CLASSES

The Cascade menupane widget is built from the Core, Composite, Constraint, Manager and MenuPane classes. Note that the Constraint fields are not used in this widget and are not listed in the resource tables below. Also, since the Composite class contains no resources that can be set by the user, there is no table for Composite class resources.

The widget class to use when creating a cascading menupane is **XwcascadeWidgetClass**. The class name is **Cascade**.

DESCRIPTION

The Cascade menupane widget is a composite widget which may be used by an application when creating a set of menus.

The Cascade menupane widget always displays its managed children in a single column, and always attempts to size itself to the smallest possible size, as described by the children it contains; as the children grow or shrink in size, the menupane will attempt to adapt its size accordingly.

The Cascade menupane widget allows a title to be displayed at the top of the menupane, the bottom of the menupane, or at both places. Additionally, the title may be either a text string or an image. The title is always centered horizontally within the menupane.

NEW RESOURCES

The MenuPane defines a set of resource types used by the programmer to specify the data for the menupane. The programmer can also set the values for the Core, Composite Manager and MenuPane widget classes to set attributes for this widget. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by Cascade.

Cascade Resource Set			
Name	**Class**	**Type**	**Default**
XtNtitlePosition	XtCTitlePosition	int	top

XtNtitlePosition

This resource is used to control where the title is displayed within the cascading menupane. To programmatically set this resource, use either the XwTOP, XwBOTTOM or XwBOTH define. To set this resource using the .Xdefaults file, use one of the strings "top" or "bottom" or "both".

NOTE: The Cascade class provides a specialized insert child procedure. This procedure allows an application to provide a special argument list type XtNchildPosiiton with an integer value. This value specifies the position within the child list where the new widget will be inserted.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

MenuPane Resource Set -- XWMENUPANE(3X)			
Name	**Class**	**Type**	**Default**
XtNattachTo	XtCAttachTo	String	NULL
XtNfont	XtCFont	XFontStruct *	"fixed"
XtNmgrTitleOverride	XtCTitleOverride	Boolean	FALSE
XtNmnemonic	XtCMnemonic	String	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNtitleImage	XtCTitleImage	XImage *	NULL
XtNtitleShowing	XtCTitleShowing	Boolean	FALSE
XtNtitleString	XtCTitleString	String	widget name
XtNtitleType	XtCTitleType	int	XwSTRING

TRANSLATIONS

The input to the Cascade menupane widget is driven by the mouse buttons. The default translations set by this widget are as follows:

```
<Btn1Down>:      select()
<LeaveWindow>:   leave()
```

The following translations are added to the cascade menupane widget when traversal is enabled within a menu hierarchy:

XWCASCADE(3X)

<visible>:	visible()
<unmap>:	unmap()

ACTIONS

leave: This routine overrides the leave action routine provided by the XwManager meta class.

select: Informs the menu manager, if present, that a select occurred, and then invokes the select callbacks, unless instructed not to by the menu manager. If no menu manager is present, then the select callbacks will be invoked.

unmap:
This action overrides the unmap action provided by the XwManager meta class.

visible: This action overrides the visible action routine provided by the XwManager meta class.

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X), XWMENUPANE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwCreateTile - create a tile suitable for area filling or patterned text.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

Pixmap XwCreateTile (screen, foreground, background, tileType)
Screen * screen;
Pixel foreground;
Pixel background;
int tileType;
```

ARGUMENTS

screen This parameter contains the screen for which the tile is to be created.

foreground
This is the foreground color to use for creating the tile.

background
This is the background color to use for creating the tile.

tileType
This is an integer value representing a particular pattern to use when creating the tile.

DESCRIPTION

XwCreateTile is a function (not a widget) that creates and returns a pixmap of screen depth, using the foreground and background colors specified. The tileType parameter is used to select the particular tile to create. Duplicate requests for the same tile, screen, foreground and background are cached to reduce overhead.

There are nine available tile types. They are defined by a set of #define statements in the file Xw.h and are described in the following table.

Define	Description
XwBACKGROUND	A tile of solid background
XwFOREGROUND	A tile of solid foreground
Xw25_FOREGROUND	A tile of 25% foreground, 75% background
Xw50_FOREGROUND	A tile of 50% foreground, 50% background
Xw75_FOREGROUND	A tile of 75% foreground, 25% background
XwHORIZONTAL_TILE	A tile of horizontal lines of the two colors
XwSLANT_RIGHT	A tile of slanting lines of the two colors
XwSLANT_LEFT	A tile of slanting lines of the two colors
XwVERTICAL_TILE	A tile of vertical lines of the two colors

To use a tile created by this function, the returned tile should be placed into the tile field of a graphics context, and the fill_style should be set to FillTiled.

XWCREATETILE(3X)

RESOURCES

XwCreateTile gives the application or widget writer an easy mechanism to specify the tile type to use. The tile type can be specified within the .Xdefaults file or an argument list. A resource converter is present to convert .Xdefault strings into the matching defined value for each of the tiles. The strings to be contained within the .Xdefaults file are as follows.

Xdefault String	Define
background	XwBACKGROUND
foreground	XwFOREGROUND
25_foreground	Xw25_FOREGROUND
50_foreground	Xw50_FOREGROUND
75_foreground	Xw75_FOREGROUND
horizontal_tile	XwHORIZONTAL_TILE
slant_right	XwSLANT_RIGHT
slant_left	XwSLANT_LEFT
vertical_tile	XwVERTICAL_TILE

For widget writers who wish to incorporate settable tiles within their resource set, the representation member of the resource definition should be set to the define XtRTileType.

RETURN VALUES

XwCreateTile returns a pixmap if successful. If an invalid tile type or screen is specified, 0 is returned.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwformWidgetClass - the X Widget's general widget layout manager

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Form.h>
```

CLASSES

A Form widget is built from Core, Composite, Constraint and XwManager classes

The widget class to use when creating a form is **XwformWidgetClass**.

The class name of Form is **Form**.

DESCRIPTION

The Form widget is a constraint based manager that provides a layout language used to establish spatial relationships between its children. It then manipulates these relationships when the Form is resized, new children are added to the Form, or its children are resized, unmanaged, remanaged or destroyed.

The following list highlights the types of layout control provided by the form widget.

Automatic Form Resizing

The form calculates new sizes or positions for its children whenever they change size or position. The new form size thus generated is passed as a *geometry request* to the parent of the form. The parent can accept the request or modify it and return it as a *geometry almost*. When a *geometry almost* is returned by the parent, the form respecifies the constraints to match the parent's reply size.

Column Constraints

Sets of widgets can be displayed in a single column or in multiple columns. The form may increase or decrease the spacing between widgets or resize the widgets, but it will not allow the widgets to overlap.

Managing, Unmanaging and Destroying Children

When a widget within a form is unmanaged or destroyed, it is removed from the constraint processing and the constraints are reprocessed to reposition and/or resize the form and its contents. Any widgets that referenced it are rereferenced to the widget that it had been referencing. For the unmanaged case, if the widget is remanaged, the widgets that were previously referencing it are rereferenced to it, thus preserving the original layout.

Optimal Child Sizes and Positions

The Form widget also calculates the sizes and positions of its children to both match the constraints defined and to match either the initial size of the widget or the size given when the widget was modified through XtSetValues. These values are further constrained to match a given form size only when the form's size is being explicitly changed through its resize procedure, or its parent returns a geometry almost when the form makes a geometry request.

Row Constraints

Sets of widgets can be set up as a row so that resizing a form may increase or decrease the spacing between the widgets. The form may also make the widgets smaller if desired, but it will not allow the widgets to overlap.

Spanning Constraints

A widget can be created with a set of constraints such that it spans the width or height of a form. This is often used for the layout of scrollbars and titlebars. Constraints that cause a widget to span both the width and height of a form can also be specified.

NEW RESOURCES

The Form does not add any new resources. All of the functionality for the form is tied to its

constraint resources.

CONSTRAINT RESOURCES

The following resources are attached to every widget inserted into Form. To specify an of these resources within a .Xdefaults file, drop the XtN from the resource name. Refer to CONSTRAINT(3X) for a general discussion of constraint resources.

Constraint Resource Set -- Children of FORM(3X)			
Name	**Class**	**Type**	**Default**
XtNxAddWidth	XtCXAddWidth	Boolean	FALSE
XtNxAttachOffset	XtCXAttachOffset	int	0
XtNxAttachRight	XtCXAttachRight	Boolean	FALSE
XtNxOffset	XtCXOffset	int	0
XtNxRefName	XtCXRefName	String	NULL
XtNxRefWidget	XtCXRefWidget	Widget	the parent form
XtNxResizable	XtCXResizable	Boolean	FALSE
XtNxVaryOffset	XtCXVaryOffset	Boolean	FALSE
XtNyAddHeight	XtCYAddHeight	Boolean	FALSE
XtNyAttachBottom	XtCYAttachBottom	Boolean	False
XtNyAttachOffset	XtCYAttachOffset	int	0
XtNyOffset	XtCYOffset	int	0
XtNyRefName	XtCYRefName	String	NULL
XtNyRefWidget	XtCYRefWidget	Widget	the parent form
XtNyResizable	XtCYResizable	Boolean	FALSE
XtNyVaryOffset	XtCYVaryOffset	Boolean	FALSE

XtNxAddWidth XtNyAddHeight

This resource indicates whether or not to add the width or height of the reference widget to a widget's location when determining the widget's position.

XtNxAttachOffset XtNyAttachOffset

When a widget is attached to the right or bottom edge of the form (through the above resources), the separation between the widget and the form is defaulted to 0 pixels. This resource allows that separation to be set to some other value. Also, for widgets that are not attached to the right or bottom edge of the form, this constraint specifies the minimum spacing between the widget and the form.

XtNxAttachRight XtNyAttachBottom

Widgets are normally referenced from "form left" to "form right" or from "form top" to "form bottom." The attach resources allow this reference to occur on the opposite edge of the form. These resources, when used in conjunction with the varyOffset resources, allow a widget to float along the right or bottom edge of the form. This is done by setting both the Attach and VaryOffset resources to TRUE. A widget can also span the width and height of the form by setting the Attach resource to TRUE and the VaryOffset resource to FALSE.

XtNxOffset XtNyOffset

The location of a widget is determined by the widget it references. As the default, a widget's position on the form exactly matches its reference widget's location. There are two additional pieces of data used to determine the location. This resource defines an integer value representing the number of pixels to add to the reference widget's location when calculating the widget's location.

XtNxRefName XtNyRefName

When a widget is added as a child of the form its position is determined by the widget it references. The reference widget must be created before the widget which references it is created. These resources allow the name of the reference widget to be given. The form converts this name to a widget to use for the referencing. Any widget that is a direct child

of the form or the form widget itself can be used as a reference widget.

XtNxRefWidget XtNyRefWidget

The application can specify the reference widget as either a string representing the name of the widget (as described above) or as the Widget ID value returned from XtCreateWidget. This resource is the means by which a widget ID is specified.

XtNxResizable XtNyResizable

This resource specifies whether the form can resize (shrink) a widget. When a form's size becomes smaller the form will resize its children only after all of the inter-widget spacing of widget's with their VaryOffset resource set to TRUE. The form keeps track of a widgets initial size or size generated through XtSetValues so that when the form then becomes larger the widget will grow to it original size and no larger.

XtNxVaryOffset XtNyVaryOffset

When a form is resized, it processes the constraints contained within its children. This resource allows the spacing between a widget and the widget it references to vary (either increase or decrease) when a form's size changes. For widgets that directly reference the form widget this resource is ignored. The spacing between a widget and its reference widget can decrease to 0 pixels if the XtNAddWidth resource is FALSE or to 1 pixel if XtNAddWidth is TRUE.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNforeground	XtCForeground	Pixel	Black
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's

XWFORM(3X)

translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

EXAMPLES

The following examples list the desired layout of widgets within a form and the constraints needed to achieve the layout.

TitleBar

Use the following constraints to get a titlebar widget to span the top of a form the following constraints can be used. For a widget named *title* the .Xdefaults file will contain.

*title.xRefName:	"form widget name"	attach to the left edge of the form
*title.xOffset:	5	offset 5 pixels from the left edge
*title.xResizable:	TRUE	title is horizontally resizable
*title.xAttachRight:	TRUE	attach to the right edge of the form
*title.xAttachOffset:	5	offset 5 pixels from right edge
*title.yRefName:	"form widget name"	attach to the top edge of the form

Dynamic Scrolled Window

The above constraints work generally for any widget type that is to span the form and that need to be resized as the form increases or decreases in size. For example, if the child widget is a scrolled window named *sWin* that dynamically resizes as the form resizes in both the horizontal and vertical directions the constraints are as follows.

*sWin.xRefName:	"form widget name"	attach to the left edge of the form
*sWin.xOffset:	5	offset 5 pixels from the left edge
*sWin.xResizable:	TRUE	scrollWin is horizontally resizable
*sWin.xAttachRight:	TRUE	attach to the right edge of the form
*sWin.xAttachOffset:	5	offset 5 pixels from right edge
*sWin.yRefName:	"form widget name"	attach to the top edge of the form
*sWin.yOffset:	5	offset 5 pixels from the top edge
*sWin.yResizable:	TRUE	scrollWin is vertically resizable
*sWin.yAttachBottom:	TRUE	attach to the bottom edge of the form
*sWin.yAttachOffset:	5	offset 5 pixels from bottom edge

Right or Bottom Attached Widgets

For a widget named *widget* to float along the right or bottom edge of the form as it is resized the constraint set is the same as for the titlebar example with the following changes.

*widget.xRefName:	"any widget name"	the widget to the left of this one
*widget.yVaryOffset:	TRUE	adjust the spacing with the reference widget

Columns of Widgets

A set of widgets within a form can be arranged in columns by using of the constraint language. The following set of constraints sets up two columns of widgets. The naming of the widgets for this example is *wI,J* where *I* is the column and *J* is the row.

```
*w0,0.xRefName:     "form widget name"
*w0,0.xOffset:      5
*w0,0.xResizable:   TRUE
*w0,0.yRefName:     "form widget name"
*w0,0.yOffset:      5
*w0,0.yResizable:   TRUE

*w0,1.xRefName:     widget0,0
*w0,1.xResizable:   TRUE
*w0,1.yRefName:     widget0,0
*w0,1.yOffset:      5
*w0,1.yAddHeight:   TRUE
*w0,1.yResizable:   TRUE

*w1,0.xRefName:     widget0,0
*w1,0.xOffset:      20
*w1,0.yAddWidth:    TRUE
*w1,0.xResizable:   TRUE
*w1,0.yRefName:     widget0,0
*w1,0.yOffset:      5
*w1,0.yAddHeight:   TRUE
*w1,0.yResizable:   TRUE

*w1,1.xRefName:     widget1,0
*w1,1.xResizable:   TRUE
*w1,1.yRefName:     widget1,0
*w1,1.yOffset:      5
*w1,1.yAddHeight:   TRUE
*w1,1.yResizable:   TRUE
```

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), COMPOSITE(3X), CONSTRAINT(3X), XWMANAGERCLASS(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWFRAME(3X)

NAME

XwframeWidgetClass - the X Widget's frame widget

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Frame.h>

CLASSES

The Frame widget is built from the Core, Composite, and Manager classes.

The widget class to use when creating a frame is **XwframeWidgetClass.**

The class name for frame is **Frame.**

DESCRIPTION

The Frame widget is a very simple manager used to enclose a single child in a border drawn by the Frame widget. It uses the XwManager class resources for border drawing and performs geometry management such that its size will always match its child size plus the highlightThickness defined for it.

Frame is most often used to enclose other managers when the application developer desires the manager to have the same border appearance as the primitive widgets. Frame can also be used to enclose primitive widgets that do not support the same type of border drawing. This will give visual consistency when developing applications using diverse widget sets.

NEW RESOURCES

The Frame widget does not define any resources.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X)
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwimageEditWidgetClass - the X Widget's image editor widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/ImageEdit.h>
```

CLASSES

ImageEdit is built from the Core and Primitive classes.

The widget class to use when creating an image editor is **XwimageEditWidgetClass.**

The class name is **ImageEdit.**

DESCRIPTION

The ImageEdit widget allows an image to be displayed in an enlarged format so that it may be edited on a pixel-by-pixel basis. The specified image is displayed in a grid structure so that a user may see and modify the composition.

To change the image, the user moves the mouse to the desired point and presses the mouse button. The pixel under the cursor will change to the foreground color. If the cursor is moved while the button is pressed, all pixels that are touched will change to the foreground color.

NEW RESOURCES

The ImageEdit defines a set of resource types that can be used by the programmer to control the appearance and behavior of the widget. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string name. The following table contains the set of resources defined by ImageEdit.

ImageEdit Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackground	XtCBackground	Pixel	Black
XtNdrawColor	XtCBackground	Pixel	Black
XtNeraseColor	XtCBackground	Pixel	White
XtNeraseOn	XtCEraseOn	Boolean	TRUE
XtNgridThickness	XtCGridThickness	int	1
XtNimage	XtCImage	XImage *	NULL
XtNpixelScale	XtCPixelScale	int	6

XtNbackground

ImageEdit redefines the core class background resource to default it to the color black. This is then used as the background color for the widget's window which will be reflected in the grid color.

XtNdrawColor

This resource define the color to be used for drawing in the widget.

XtNeraseColor

This resource defines the color used for erasing in the widget. Erase is enabled by the **eraseOn** resource. When selections occur on the widget, the widget determines the color of the pixel selected. If the selected pixel is not the same as the draw color, the draw color will be used to draw until the button release occurs. If the selected pixel is the draw color, the erase color will be used for drawing until the button release occurs.

XtNeraseOn

This resource is a boolean variable that indicates whether erasing is enabled or not. If set to TRUE, drawing will occur as described above. If set to FALSE, only the draw color will be used for drawing.

XtNgridThickness
This resource defines the separation between the magnified pixels.

XtNimage
This is a pointer to the image that is displayed in the grid. It points to an XImage structure.

XtNpixelScale
This resource defines the magnification factor to use when displaying the expanded image.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer

XWIMAGEEDIT(3X)

to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to this widget.

TRANSLATIONS

The following translations are defined for the ImageEdit widget.

<BtnDown>:	select()
<BtnUp>:	release()
Button1<PtrMoved>:	moved()
<EnterWindow>:	enter()
<LeaveWindow>:	leave()

ACTIONS

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the image edit's border will be highlighted. Otherwise no action is taken.

leave: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the image edit's border will be unhighlighted. Otherwise no action is taken.

moved: Moved causes drawing or erasing to occur from the last cursor position to the current cursor position.

release: Release concludes a drawing sequence and invokes primitive class XtNrelease callbacks.

select: Selections occurring on an image edit cause drawing or erasing on the selected pixel, activate the moved action for continuous drawing and invoke the primitive class XtNselect callback functions.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwlistWidgetClass - the X Widget's list manager widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/SWindow.h>
#include <Xw/List.h>
```

CLASSES

List is built from the Core, Composite, Constraint, Manager and ScrolledWindow classes.

The widget class to use when creating a list manager is **XwlistWidgetClass.** The class name is **List.**

DESCRIPTION

The List widget allows a two-dimensional set of widgets to be presented to the user in a rows/columns fashion. The layout will typically consist of n columns, not all of which need to be visible on the screen at one time. Each column will have some number of objects, such as labels or icons, arranged vertically. Separate columns may have unequal numbers of members--column A may have 10 elements, while column B has 17 elements. All members of each column are not required to be visible on the screen. The entire list window can be scrolled either vertically or horizontally, but the individual columns cannot be individually scrolled. If an application needs to have columnar scrolling, it may instantiate multiple List widgets, each having only one column.

By default, each column is wide enough to display the longest item in the data. A resource is available to allow each column to be a fixed width, with the excess characters being clipped. When the List widget is shrunk by a Resize call, columns that are beyond the right edge of the new size will be clipped. List elements are also adjusted to force a common height, with each element being set to the height of the tallest member of the column. This automatic sizing can be turned off through a resource by forcing each element to an arbitrary height. If a constant height is selected, any element that will not fit in the specified space will be clipped.

The List widget provides management and layout functions for its elements, as well as a means for the user to choose elements, and allows an application to be notified when those elements are selected. However, it is the responsibility of the application to create the actual widgets that are to be inserted into the list. The widgets may be of any type, but only Primitive class widgets will highlight correctly.

To construct a list, the application must create each element as a child of the List widget. The row and column position of the element can be specified by means of a constraint resource. If the row and column are not given, the List widget will fill in the position of the element and store it in the constraint record so that it may be examined later. Position assignments are made from left to right, filling all columns on a row before moving down to create a new row.

The List widget supports two methods of choosing an item from its displayed list: single and multiple. A resource controls which mode is currently active.

In single choice mode, the user may move the cursor onto any element in the list and click the mouse button defined as "Element Select." By default, this is the left button. When the button is pressed, the list item is highlighted. If the user drags the mouse with the button held down, the highlighted selection will track the pointer. If the pointer moves off the currently highlighted item, it will become unselected, returning to its original state, and the item that the pointer has moved onto becomes highlighted. When the user releases the button, the currently selected item becomes the "choice," and the List widget invokes the select callback associated with the chosen item. The application must take over the widget's select callback in order to be notified that the item has been selected.

Multiple item selection is designed to allow the user to select several elements from the displayed list. When the user presses the mouse button bound to "Element Select," the item currently under the pointer is highlighted to indicate that it is included in the selection set. As the user drags the mouse with the button down, the original choice remains highlighted, and any new items that the pointer touches also becomes highlighted. At any time, the user can "back up" the

XWLIST(3X)

selection by leaving an item on the same side as it was entered. When the user finally releases the button, all highlighted elements are marked as chosen, and the selection callback is invoked for each item.

Selections can be either "sticky" or "instant." The selection mode is set through a resource. If set to sticky, the selection will remain highlighted after the user releases the mouse button, and will not be cleared until the next button press. In instant mode, the highlight will disappear when the button is released.

The selection mechanism can be affected by a "bias" that is controlled through a widget resource. The allowable bias types are row, column, and none (default). In this mode only list items that are actually touched with the pointer are included in the selection. In Row Bias mode. entire rows of items may be selected by moving the pointer vertically within a column. For example, consider the following case:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

If the user pressed the mouse button when the pointer was over item 3 and then moved to item 9, items 4 thru 8 would also be highlighted. In Column Bias mode, entire columns can be selected by movement between rows. Using the above diagram, and assuming column bias, if the user clicked on item 2 and moved to item 3, elements 8 and 14 would also be selected.

Additional selections can be made without disturbing the original by following the above procedures, by depressing the button bound to the "Append Select" function (which is defined as SHIFT + Left button in the default case). Extended selection can be disabled through a resource.

The List widget also allows an element to be selected by a double click. This feature can be enabled or disabled, and the interval between clicks can be configured through the XtNenableDoubleClick and XtNclickInterval resources.

The visual effect of highlighting can be accomplished in two ways: simple border highlighting, and inverse video. This may be configured through a List widget resource. Both styles are necessary--the inverse style of highlighting is by far the most common and natural interface, but could possibly conflict with an application or window manager that uses inverse to indicate the X11 "selection." The default highlighting style is inverse.

A user can select items that are not currently visible by simply extending the selection out of the visible window in the desired direction. The list will automatically scroll under the selection as needed, until there are no more list elements available in the given direction. For example, in single-selection mode, if the user were to begin the selection on a visible element, and then drag the cursor down the column past the last visible item, the window would scroll up to display further choices.

It is important to note that the List overrides any other selection method which may be defined by its elements. For example, if toggle buttons are inserted into the list, they will highlight when they are selected, but they will not "toggle."

When a list element is destroyed, the list will be re-ordered according to the value of the XtNdestroyMode resource. When it is XwSHRINK_COLUMN (the default), all list elements below the affected widget and in the same column will be moved up one row, and their row constraint resources will be updated to reflect the new positioning. When this resource is set to XwSHRINK_ALL, the elements will be moved in a row-wise fashion to fill the spot left by the affected element. The widget to the right of the affected one will be moved to the left, and so on to the last column. The first element of the next row will be moved into the last spot on the current column. This process will continue for all remaining rows in the list. If the value of this resource is XwNO_SHRINK, the list will not change its ordering and a "hole " will appear in the place of the affected element.

NEW RESOURCES

The List widget defines a unique set of resource types which can be used by the programmer to control the appearance and behavior of the list. The programmer can also set the values for the

Core, Composite, Constraint, Manager and ScrolledWindow widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, drop the XtN from the resource name. The following table contains the set of resources defined by List.

List Resource Set			
Name	**Class**	**Type**	**Default**
XtNclearList	XtCClearList	Boolean	FALSE
XtNclickInterval	XtCClickInterval	int	350
XtNcolumnWidth	XtCColumnWidth	int	0
XtNdestroyMode	XtCDestroyMode	int	XwSHRINK_COLUMN
XtNdoubleClick	XtCCallback	Pointer	NULL
XtNelementHeight	XtCElementHeight	int	0
XtNelementHighlight	XtCElementHighlight	int	XwINVERT
XtNenableDoubleClick	XtCEnableDoubleClick	Boolean	FALSE
XtNnoExtSelection	XtCNoExtSelection	Boolean	FALSE
XtNnumColumns	XtCNumColumns	int	1
XtNnumSelectedElements	XtCNumSelectedElements	int	0
XtNselectedElements	XtCSelectedElements	WidgetList*	NULL
XtNselectionBias	XtCSelectionBias	int	XwNO_BIAS
XtNselectionMethod	XtCSelectionMethod	int	XwSINGLE
XtNselectionStyle	XtCSelectionStyle	int	XwINSTANT
XtNsmartClient	XtCSmartClient	Boolean	FALSE
XtNunmanageList	XtCUnmanageList	Boolean	FALSE

XtNclearList

If a list is created without enough elements to form a perfect rectangle, or individual elements are deleted, "dummy" widgets will be created to maintain the proper visual ordering of the list. At a later time, if an application wishes to delete all elements of the list, it should set this resource to TRUE. This causes the List manager to call XtDestroyWidget for every widget in the list, including the invisible placeholder items.

XtNclickInterval

This specifies the number of milliseconds that two button presses must occur in to be considered a doubleclick. The default is 350.

XtNcolumnWidth

The width of each column. If the value is 0, the width defaults to the width of the largest element.

XtNdestroyMode

Controls the visual appearance of the list when an element is deleted. One of XwSHRINK_COLUMN, XwSHRINK_ALL or XwNO_SHRINK.

XtNdoubleClick

If doubleclick is enabled, the application should add a callback routine to this resource. When a double click occurs, the callback list will be invoked with the call_data parameter set to a pointer to the widget (element) that was selected.

XtNelementHeight

The height of each element. Zero implies that each element is resized to the height of the tallest element.

XtNelementHighlight

This controls the highlight mode on selection - either border highlighting (XwBORDER) or inversion (XwINVERT).

XtNenableDoubleClick

When this resource is TRUE, the doubleclick feature is enabled. When it is set to false, only single click selections will occur.

XWLIST(3X)

XtNnoExtSelection
Extended selection can be disabled by setting this resource to TRUE.

XtNnumColumns
The number of columns in the list.

XtNnumSelectedElements
The number of widgets currently selected (in the list pointed to by XtNselectedElements).

XtNSelectedElements
This is a list of the widgets currently marked as selected. An application program can issue a call to XtGetValues on this resource at any time to query the selected elements.

XtNselectionBias
Bias mode - either XwNO_BIAS, XwROW_BIAS or XwCOL_BIAS.

XtNselectionMethod
Controls the selection mode - either one element at a time (XwSINGLE) or multiple (XwMULTIPLE).

XtNselectionStyle
Controls the type of selection - either XwINSTANT or XwSTICKY.

XtNsmartClient
The overhead associated with the managemant of the list and dummy elements can be reduced substantially for an application that only needs a simple list. When XtNsmartClient is TRUE, the list makes the following assumptions:
1) The elements are inserted to the list in the proper order. The list will assign the proper locations if none are specified, but if locations are given they are assumed to be correct. If they are not, certain operations may perform incorrectly, such as row and column bias in multiple selection mode.
2) When an item is deleted, no special action is taken. The list simply shrinks by one space, and the row and column resources attached to each element are not renumbered.

XtNunmanageList
To unmanage all elements of the list, set this resource to TRUE. To manage the entire set of items set it to FALSE.

CONSTRAINT RESOURCES

The following resources are attached to every widget inserted into List. Refer to CONSTRAINT(3X) for a general discussion of constraint resources.

Constraint Resource Set -- Children of XWLIST(3X)			
Name	**Class**	**Type**	**Default**
XtNcolumnPosition	XtCColumnPosition	int	-1
XtNrowPosition	XtCRowPosition	int	-1
XtNselected	XtCSelected	Boolean	FALSE

XtNrowPosition,XtNcolumnPosition
This is the row,column location of the element in the list. If these values are greater than or equal to zero, the widget is inserted into the list at that position. If the values are left at -1, the List widget will create a list with XtNnumColumns number of columns, assigning row and column positions as needed.

XtNselected
If this resource is set to TRUE, the element will highlight as if it were selected by the user, but the XtNselect callback will not be invoked.

INCORPORATED RESOURCES

No incorporated resources are currently exported by the List widget.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

ScrolledWindow Resource Set - XWSCROLLEDWINDOW(3X)			
Name	**Class**	**Type**	**Default**
XtNforceHorizontalSB	XtCForceHorizontalSB	Boolean	FALSE
XtNforceVerticalSB	XtCForceVerticalSB	Boolean	FALSE
XtNhsbHeight	XtCHsbHeight	int	20
XtNhsbWidth	XtCHsbWidth	int	285
XtNhsbX	XtCHsbX	int	-1
XtNhsbY	XtCHsbY	int	-1
XtNhScrollEvent	XtCCallBack	Pointer	NULL
XtNinitialX	XtCInitialX	int	0
XtNinitialY	XtCInitialY	int	0
XtNvsbHeight	XtCVsbHeight	int	285
XtNvsbWidth	XtCVsbWidth	int	20
XtNvsbX	XtCVsbX	int	-1
XtNvsbY	XtCVsbY	int	-1
XtNvScrollEvent	XtCCallBack	Pointer	NULL

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

XWLIST(3X)

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the list widget.

TRANSLATIONS

The translations used for List are as follows:

```
<EnterWindow>:      enter()
<LeaveWindow>:      leave()
```

ACTIONS

enter: Enter window events occurring on the list window are handled by this action.

leave: Leave window events occurring on the list window are handled by this action.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), COMPOSITE(3X), CONSTRAINT(3X), XWMANAGERCLASS(3X), XWSCROLLEDWINDOW(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwmanagerWidgetClass - X Widget Manager MetaClass

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

CLASSES

The Manager class is built from the Core, Composite and Constraint classes.

DESCRIPTION

The manager class is an X Widget meta class. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other widget classes. It provides methods (procedures) which handle keyboard traversal and border highlighting for other manager widgets.

NEW RESOURCES

The manager class defines a set of resources used by the programmer to specify data for widgets that are subclasses of Manager. The string to be used when setting any of these resources in an application defaults file (.Xdefaults for example) can be obtained by stripping the preface "XtN" off the resource name. For instance, XtNtraversalOn becomes traversalOn.

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

XtNbackgroundTile

This resource defines the tile to be used for the background of the widget. It defines a particular tile to be combined with the foreground and background pixel colors. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNbottomShadowColor

This resource defines the color that is combined with the bottom shadow tile and foreground color to create a pixmap used to draw the bottom and right sides of the border.

XtNbottomShadowTile

This resource defines the tile used in creating the pixmap used for drawing the bottom and right border shadow for the widget. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNforeground

This resource defines the foreground color for the widget. Widgets built upon this class can use the foreground for their drawing.

XtNhighlightThickness

This resource specifies an amount of border spacing around the border of the widget. It

is typically used by managers to have padding space around their children and to draw special borders. This highlight thickness is and an integer value representing the width, in pixels, of the border area. This value must be greater than or equal to 0.

XtNlayout

This flag controls how the manager widget's geometry deals with too little or too much space. The valid settings for this field are XwMINIMIZE, XwMAXIMIZE and XwIGNORE. When setting this field in a resource file such as the .Xdefaults file, use "minimize", "maximize" or "ignore". Typically, the XwMINIMIZE means to request the minimum amount of space necessary to display all children. The XwMAXIMIZE means that if additional space is given to the widget (i.e., at create time or set values time) then use the additional space as padding between children widgets. The XwIGNORE settings means, maintain the size set at create time or at set value time and never change size in response to a child widget's request (i.e., added/deleted/modified a child widget). Look at the description of the individual manager widgets to see if this feature is supported.

XtNnextTop

This callback procedure is used by the applications programmer to move the focus from one toplevel widget to another toplevel widget.

XtNshadowOn

Manager, like Primitive, implements a drawing routine to draw a two pixel wide shadowing border around the widget inside of the highlight used for traversal. This resource indicates whether to take into account the additional space needed for drawing the shadow border. If set to TRUE the highlight thickness value of the widget will be automatically incremented by two.

The shadowed border is drawn with two pixmaps created using the following four resources. The border is split into two areas: the top-left shadow and the bottom-right shadow.

XtNtopShadowColor

This resource defines the color that is combined with the top shadow tile and foreground color to create a pixmap used to draw the top and left sides of the border.

XtNtopShadowTile

This resource defines the tile used in creating the pixmap used for drawing the top and left border shadow for the widget. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNtraversalOn

The application can define whether keyboard traversal is active or not. The default for this resource is typically FALSE.

KEYBOARD TRAVERSAL

If the traversalOn resource is TRUE (either when the widget is created or during a call to XtSetValues) the manager widget's translation table is augmented with the following translations:

<EnterWindow>:	enter()
<LeaveWindow>:	leave()
<Visible>:	visible()
<FocusIn>:	focusIn()
<FocusOut>:	focusOut()

ACTIONS

enter: If the widget is a top level manager and traversal is on, then begin or resume traversal.

leave: If the widget is a top level manager and traversal is on, then suspend traversal.

focusIn:

If the widget is a top level manager and traversal is on, then initiate traversal within this widget hierarchy.

focusOut:

If traversal is on for this widget, then remove the keyboard focus from the Primitive

XWMANAGER(3X)

widget to which it had previously been set.

visible: If traversal is on for a widget of this class and the widget that is focused becomes hidden (e.g. another window obscures its visibility), then the focus moves to another visible widget.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwmenuButtonWidgetClass - the X Widgets MenuButton widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/MenuBtn.h>
```

CLASSES

The MenuButton widget is built from the Core, Primitive, and Button classes.

The widget class to use when creating a MenuButton is **XwMenuButtonWidgetClass**. The class name is **MenuButton.**

DESCRIPTION

The MenuButton widget is commonly used with MenuPane and MenuMgr widgets to build a menu system. The MenuButton consists of a single label, a mark and a cascade indicator. The MenuButton is broken into three areas. Starting from the left border of the MenuButton the areas are: the mark area, the label area and the cascade area. By default, the mark area contains a checkmark image, the label area contains the name of the MenuButton widget and the cascade area contains an arrow image. The label can be set to any string or image and the label area attempts to grow or shrink to accommodate it. The mark and cascade can be set to an image, although the width of the these areas remains fixed.

The default semantic for this button is that button 1 down causes the select callbacks to be invoked. When a MenuButton is used in a MenuMgr, this may be overridden by the MenuMgr. The select callbacks may also be invoked by a keyboard accelerator or mnemonic, although it is up to the MenuMgr to determine whether the accelerator or mnemonic is active.

The MenuButton is often used with a MenuPane and MenuMgr widget, although it is not necessary to do so. The MenuButton could simply be used as another button widget.

NEW RESOURCES

The MenuButton widget defines a set of resource types used by the programmer to specify the data for the MenuButton. The programmer can also set the values for the Core, Primitive and Button widget classes to set attributes for this widget. The following table contains the set of resources defined by MenuButton. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name.

MenuButton Resource Set			
Name	**Class**	**Type**	**Default**
XtNcascadeImage	XtCCascadeImage	XImage *	NULL
XtNcascadeOn	XtCCascadeOn	Widget	NULL
XtNcascadeSelect	XtCCallback	Pointer	NULL
XtNcascadeUnselect	XtCCallback	Pointer	NULL
XtNinvertOnEnter	XtCInvertOnEnter	Boolean	TRUE
XtNkbdAccelerator	XtCKbdAccelerator	String	NULL
XtNlabelImage	XtCLabelImage	XImage *	NULL
XtNlabelType	XtCLabelType	int	XwSTRING
XtNmarkImage	XtCMarkImage	XImage *	NULL
XtNmenuMgrId	XtCMenuMgrId	Widget	NULL
XtNmgrOverrideMnemonic	XtCMgrOverrideMnemonic	Boolean	FALSE
XtNmnemonic	XtCMnemonic	String	NULL
XtNsetMark	XtCSetMark	Boolean	FALSE

XtNcascadeImage

This resource points to an XImage structure that describes the cascade image data. The cascade area is a fixed size (16x16). If this resource is set to NULL, then the default

cascade image (an arrow) is used. The cascade indicator is not displayed if the XtNcascadeOn resource is set to NULL. If the image is defined with XYBitmap data, then the image is nicely inverted when the MenuButton is highlighted.

XtNcascadeOn

This resource determines if the cascade indicator is displayed. It is typically set only by the MenuMgr and contains the widget ID of the MenuPane which cascades as a submenu from this MenuButton. This resource is set to NULL to disable the display of the cascade indicator.

XtNcascadeSelect

This resource provides the means for registering callback routines which are invoked if a cascade indicator is displayed and the pointer moves into the cascade area. In some cases, the MenuMgr suppresses the calling of these callback routines. The MenuButton does not pass any data in the call_data field of the callback.

XtNcascadeUnselect

This resource provides the means for registering callback routines that are invoked if a cascade indicator is displayed and the pointer moves out of the cascade area. These callbacks are only invoked if the XtNcascadeSelect callbacks have been previously invoked. The MenuButton passes data in the call_data field of the callback. It is a pointer to the XwunselectParams data structure shown below:

```
typedef struct
{
  Position              rootX;
  Position              rootY;
  Boolean               remainHighlighted;
} XwunselectParams;
```

The rootX and rootY parameters have the position of the pointer relative to the root window when the event occurred that caused the XtNcascadeUnselect call backs to be called. The remainHighlighted parameter is used by cascading submenus. It is set by the MenuMgr's call back routine to indicate that the pointer traversed from a cascade into the submenu. If the boolean is set TRUE, then the MenuButton does not unhighlight on exit. It also sets up an event handler on its parent MenuPane so that it is notified if the pointer enters another MenuButton, in which case the MenuButton should then unhighlight.

XtNinvertOnEnter

This boolean resource determines whether the menu button will invert when the cursor is moved into it.

XtNkbdAccelerator

This resource is a string that describes a set of modifiers and the key to be used to select this MenuButton widget. The format for this string is identical to that used by the translations manager, with the exception that only a single event may be specified and only KeyPress events are allowed. If the MenuButton does not have a MenuMgr associated with it, then this resource is ignored. The MenuMgr determines when, and if, this accelerator is available.

XtNlabelImage

If XtNlabelType indicates that a label image should be displayed, then this resource contains the image used. This is a pointer to an XImage structure which describes the label image data. If the image is defined with XYBitmap data, then the image is nicely inverted when the MenuButton is highlighted.

XtNlabelType

Two styles of labels are supported by the MenuButton widget: text string labels and image labels. The text string label is defined by the Button resource XtNlabel and the image label is defined by the XtNlabelImage resource. To programmatically set this resource, use either the XwSTRING define or the XwIMAGE define. To set this resource using the .Xdefaults files, use one of the strings "string" or "image".

XWMENUBUTTON(3X)

XtNmarkImage

This resource points to an XImage structure which describes the mark image data. The mark area is a fixed size (16x16). If this resource is set to NULL, then the default mark image is used. The mark is not displayed if the XtNsetMark resource is set to FALSE. If the image is defined with XYBitmap data, then the image is nicely inverted when the MenuButton is highlighted.

XtNmenuMgrId

This resource is used only by MenuMgr widgets to indicate to the MenuButton widget its MenuMgr. If this is set to NULL, then the MenuButton checks if it has a MenuMgr at the appropriate level in its parentage. This resource should not be set by users.

XtNmgrOverrideMnemonic

This boolean resource determines if the mnemonic character is underlined in the label string. If it is set to TRUE, then the mnemonic character is not underlined. This resource is typically set only by MenuMgr widgets.

XtNmnemonic

Certain MenuMgr widgets allow the MenuButtons to have a mnemonic. Mnemonics provide the user with another means for selecting a menu button. This resource is a NULL terminated string, containing a single character. The MenuMgr determines if this mnemonic is available. If the XtNmgrOverrideMnemonic resource is false and the mnemonic is found in the label string, then that character is underlined when the MenuButton is displayed. Refer to XwPullDown(3X) man page for further discussion of traversal.

XtNsetMark

This boolean resource determines whether the mark is displayed.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

Button Resource Set -- XWBUTTON(3X)			
Name	**Class**	**Type**	**Default**
XtNfont	XtCFont	XFontStruct *	Fixed
XtNhSpace	XtCHSpace	int	2
XtNlabel	XtCLabel	caddr_t	widget name
XtNlabelLocation	XtCLabelLocation	int	XwRIGHT
XtNsensitiveTile	XtCSensitiveTile	int	75_foreground
XtNvSpace	XtCVSpace	int	2

TRANSLATIONS

The default translations set by the MenuButton widget are as follows:

<Btn1Down>:	select()
<EnterWindow>	enter()
<LeaveWindow>:	leave()
<Key>Select:	select()
<Key>Left:	traverseLeft()
<Key>Up:	traverseUp()
<Key>Right:	traverseRight()
<Key>Down:	traverseDown()
<Key>Prior:	traversePrev()
<Key>Next:	traverseNext()
<Key>KP_Enter:	traverseNextTop()
<Key>Home:	traverseHome()

The following translations are added to the menu button widget when traversal is enabled within a menu hierarchy:

<Visible>:	visibility()
<Unmap>:	unmap()

ACTIONS

enter: If a MenuMgr is present, then it is informed of the enter event. The MenuMgr indicates whether this enter event should be processed or ignored. If there is no MenuMgr present, or if the MenuMgr indicates the event is to be processed, then the MenuButton is highlighted. A processed enter action also calls the moved action to determine if the

pointer is in the cascade indicator area.

leave: If a MenuMgr is present, then it is informed of the leave event. The MenuMgr indicates whether this leave event should be processed or ignored. If there is no MenuMgr present, or if the MenuMgr indicates that the leave event is to be processed, then the MenuButton is unhighlighted. If the XtNcascadeSelect callbacks have been called, the XtNcascadeUnselect callbacks are called.

moved: *Note: This action routine is provided only for backward compatibility and its use is discouraged.* If the MenuButton is *not* under the control of a MenuMgr widget, this action routine will determine if the pointer is in the cascade area and will invoke either the XtNcascadeSelect or XtNcascadeUnselect callbacks, as necessary. If the MenuButton *is* under the control of a MenuMgr widget, this action routine will not perform any action.

select: If the MenuButton is *not* under the control of a MenuMgr widget, this action routine will invoke each of the XtNselect callbacks associated with the MenuButton. If the MenuButton *is* under the control of a MenuMgr widget, this action routine will not perform any action.

traverseDown:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the menu button positioned below the current traversal item; wrap to the top, if necessary.

traverseHome:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the first MenuPane in the menu hierarchy.

traverseLeft:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the MenuPane cascading from this MenuButton, if one is present.

traverseNext:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the next menu tree, if one is present.

traverseNextTop:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the next top level MenuPane.

traversePrev:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the previous menu tree, if one is present.

traverseRight:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the MenuPane from which the current one has cascaded.

traverseUp:
Inform the MenuMgr controlling this widget that it should transfer the keyboard focus to the menu button positioned above the current traversal item; wrap to the bottom, if necessary.

visibility:
This action routine overrides the visibility action routine provided by the XwPrimitive meta class.

unmap:
This action overrides the unmap action routine provided by the XwPrimitive meta class.

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. See the XwPrimitive man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

XWMENUBUTTON(3X)

SEE ALSO

CORE(3), XWPRIMITIVE(3X), XWBUTTON(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWMENUMGR(3X)

NAME

XwmenuMgrWidgetClass - the X Widgets menu manager widget metaclass.

SYNOPSIS

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>

CLASSES

The menu manager class is built the Core, Composite, Constraint and Manager classes.

DESCRIPTION

The MenuMgr class is an X Widget meta class. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other menu manager widget classes.

NEW RESOURCES

The menu manager defines a set of resource types which may be used by the programmer to specify the data for widgets which are a subclass of MenuMgr. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by MenuMgr.

MenuMgr Resource Set			
Name	**Class**	**Type**	**Default**
XtNassociateChildren	XtCAssociateChildren	Boolean	TRUE
XtNkbdSelect	XtCKbdSelect	String	"<Key>Select"
XtNmenuPost	XtCMenuPost	String	"<Btn1Down>"
XtNmenuSelect	XtCMenuSelect	String	"<Btn1Up>"
XtNmenuUnpost	XtCMenuUnpost	String	NULL

XtNassociateChildren

This resource indicates whether the menu hierarchy controlled by the menu manager is accessible only from within the associated widget, or from within the widget and any of the widget's children.

XtNkbdSelect

This string resource describes the key event and any required modifiers needed to select the currently highlighted menu button. This provides the user with the means for selecting a menu item from the keyboard, without being required to use the mouse. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a key event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the button event occurs.

XtNmenuPost

This string resource describes the button event and any required modifiers needed to post one of the top level menupanes controlled by the menu manager. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a ButtonPress or ButtonRelease event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the button event occurs.

XtNmenuSelect

This string resource describes the button event and any required modifiers needed to select a menu button within any of the menupanes controlled by the menu manager. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a ButtonPress or ButtonRelease event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present

when the button event occurs.

XtNmenuUnpost

This string resource describes the key event and any required modifiers needed to unpost the currently viewable set of menupanes controlled by the menu manager. This provides the user with the means for unposting a menu hierarchy from the keyboard, without selecting a menu button. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a key event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the button event occurs.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwmenuPaneWidgetClass - the X Widgets menupane widget metaclass.

SYNOPSIS

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>

CLASSES

The MenuPane widget class is built from the Core, Composite, Constraint and Manager classes.

DESCRIPTION

The menupane class is an X Widget meta class. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other menupane widget classes. It provides a collection of resources which will be needed by most menupane subclasses.

NEW RESOURCES

The MenuPane defines a set of resource types used by the programmer to specify the data for widgets which are subclasses of MenuPane. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name.

MenuPane Resource Set			
Name	**Class**	**Type**	**Default**
XtNattachTo	XtCAttachTo	String	NULL
XtNfont	XtCFont	XFontStruct *	fixed
XtNmgrTitleOverride	XtCTitleOverride	Boolean	FALSE
XtNmnemonic	XtCMnemonic	String	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNtitleImage	XtCTitleImage	XImage *	NULL
XtNtitleShowing	XtCTitleShowing	Boolean	TRUE
XtNtitleString	XtCTitleString	String	widget name
XtNtitleType	XtCTitleType	int	XwSTRING

XtNattachTo

When used in conjunction with a menu manager, this resource provides the means by which the menupane may be attached as a cascade to a menubutton. The string which is specified represents the name of the menubutton to which the menupane is to be attached; this provides the means by which the menu manager is able to construct the menu tree. To specify that this menupane should be treated as the top level menupane within the menu tree, this string should contain the name of the menu manager widget, instead of a menubutton widget. Specifying a NULL string indicates that the menupane will not be presently attached to anything. If the menupane does not have a menu manager associated with it, then this resource is unused.

XtNfont

If the title type resource indicates that a title string should be displayed, then this resource will describe the font used to draw the title string.

XtNmgrTitleOverride

This resource is not intended to be used by applications; it should only be used by a menu manager widget, for overriding the application, and forcing off the menupane title. This is useful for those menu managers whose style dictates that certain menupane should not have a title displayed.

XtNmnemonic

Certain menu managers allow some of their menupanes to have a mnemonic. Mnemonics may be used to post a menupane using the keyboard, instead of using the pointer device. This resource is a NULL terminated string, containing a single character. Typically, the character is the same as one present in the menupane title.

XtNselect
This resource provides the means for registering callback routines which will be invoked when the menupane receives a select action.

XtNtitleImage
If the title type resource indicates that a title image should be displayed, then this resource will contain a pointer to an XImage structure; this structure describes the title image data.

XtNtitleShowing
This resource may be used by the application to control the displaying of a title within the menupane. This may be overridden, however, by a menu manager using the XtNmgrTitleOverride resource.

XtNtitleString
If the title type resource indicates that a title string should be displayed, then this resource will contain the title string which is to be used. In the case where the application does not specify a title string, the name of the menupane widget will be used. The title is displayed using the foreground color.

XtNtitleType
Two styles of titles are supported by the MenuPane widget. They include text string titles and image titles. To programmatically set this resource, use either the XwSTRING define or the XwIMAGE define. To set this resource using the .Xdefaults file, use one of the strings "string" or "image".

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWMENUSEP(3X)

NAME

XwmenuSepWidgetClass - the X Widget's menu item separator widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/MenuSep.h>
```

CLASSES

MenuSep is built from the Core, Primitive, Button, and MenuButton classes.

The widget class to use when creating a menu separator widget is **XwmenuSepWidgetClass.**

The class name for this widget is **MenuSep.**

DESCRIPTION

The MenuSep widget is a primitive widget to be used as an item separator placed between items in a menu. Several different line drawing styles are provided.

NEW RESOURCES

The MenuSep widget defines a one additional resource type. The programmer can also set the values for the Core and Primitive resources to set attributes for this widget. The Button and MenuButton resources are unused for this widget.

MenuSep Resource Set			
Name	**Class**	**Type**	**Default**
XtNseparatorType	XtCseparatorType	int	XwSINGLE_LINE

XtNseparatorType

This resource defines the type of line drawing to be done in the menu separator widget. Five different line drawing styles are provided. They are single, double, single dashed, double dashed and no line. The separator type can be set through an argument list by using one of the defines: XwSINGLE_LINE, XwDOUBLE_LINE, XwSINGLE_DASHED_LINE, XwDOUBLE_DASHED_LINE, and XwNO_LINE. The separator type can be set through the .Xdefaults file by using one of the following strings: single_line, double_line single_dashed_line, double_dashed_line and no_line.

Menu separator widgets will draw single pixel wide shadow border as opposed to the normal double pixel wide border.

The line drawing done within the menu separator will be automatically centered within the height of the widget.

The separator type of no_line is provided as an escape to the application programmer who needs a different style of drawing or who just wants the shadowing border for the separator. To create an alternate style, a pixmap the height of the widget can be created. After the separator widget has been created, this pixmap can be used as the background pixmap by building an argument list using the XtNbackgroundPixmap argument type as defined by Core and setting the widgets background through XtSetValues. Whenever the widget is redrawn its background will be displayed which contains the desired separator drawing. Note that the pixmap can only be set after the widget is created. If set when created, it will be overridden by the normal background pixmap created by the Primitive class.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

TRANSLATIONS

The menu separator widget defines no translations.

ACTIONS

The menu separator widget defines no actions.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWMOVEFOCUS(3X)

NAME

XwMoveFocus - move the keyboard focus (and the pointer) to a new toplevel widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

void XwMoveFocus (w)
Widget w;
```

ARGUMENTS

w	This is the ID of the widget to which the application wishes to move the focus. It must be the toplevel widget in a widget hierarchy and it must be a subclass of the shell widget class.

DESCRIPTION

XwMoveFocus is a very specialized function which can be used to move the keyboard and pointer focus to another toplevel widget hierarchy. It is useful when an application using keyboard traversal has multiple toplevel widget hierarchies and wishes to be able to move between these hierarchies without using the pointer device. Specifically, this function will warp the pointer to (1,1) in the specified widget and will also make a call to XSetInputFocus (this is necessary for use with window managers using an explicit listener mode).

ORIGIN

Hewlett-Packard Company.

SEE ALSO

Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwPanelWidgetClass - An X Widget for creating panels.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Panel.h>
```

CLASSES

Panel is built from the Core, Composite, Constraint and Manager classes.

The widget class to use when creating a Panel widget is **XwpanelWidgetClass**.

The class name of Panel is **Panel**.

DESCRIPTION

Panel provides a simple creation mechanism for the creation of application windows and associated menus. The panel widget is also appropriate for application sub-windows.

Panel will manage its children in three areas, and may have no more than one child in each area. The areas are called title, menu, and work space. Children are associated with these areas by means of the XtNwidgetType resource (see below). Panel ignores all other children.

Panel lays out its children such that the child in the title area is placed at the top, the child in the menu area is placed next, and the child in the work space area is placed at the bottom. Display of the title child can be optionally inhibited if the panel is under the control of a window manager that provides titlebars.

When Panel has its width changed by its parent, the child in the menu area, if displayed, is allowed to pick its own height, the child in the title area remains the same height and the child in the work space area is diminished or enlarged to fill the remaining available space. When Panel has its height reduced by its parent, space is taken from the work space area until that area is completely hidden. Further reductions in the height of Panel are shared between the title and menu areas. When Panel has its height increased by its parent, if either the title or the menu are less than their optimum height they are given the space until they reach their optimum height for the given width. If both the title and the menu are at their optimum height, all space is given to the work space.

The initial width of Panel is the widest of all its children (padding is taken into account). The initial height of Panel is the sum of the heights of all its children and their padding.

When an application is running in a Panel with a titling window manager, there is a possibility of double titling. Unfortunately, the application writer cannot know at the time of development whether or not the user will have a titling window manager. Panel has two resources that together allow runtime decisions about titling. The first, XtNtopLevel, indicates whether the Panel is a candidate for double titling. The application must always set this variable appropriately. The second resource, XtNdisplayTitle, indicates whether or not the Panel should display a title.

XWPANEL(3X)

NEW RESOURCES

To specify any of these resources within a resource defaults file, simply drop the XtN prefix from the resource name. Panel defines the following new resources:

Panel Resource Set			
Name	**Class**	**Type**	**Default**
XtNdisplayTitle	XtCDisplayTitle	Boolean	TRUE
XtNhSpace	XtCHSpace	int	0
XtNtitleToMenuPad	XtCTitleToMenuPad	int	0
XtNtopLevel	XtCTopLevel	Boolean	TRUE
XtNvSpace	XtCVSpace	int	0
XtNworkSpaceToSiblingPad	XtCWorkSpaceToSiblingPad	int	0

XtNdisplayTitle

Ignored if XtNtopLevel is FALSE.

XtNhSpace

Padding between the sides of the Panel and the sides of the displayed children.

XtNtitleToMenuPad

If both a title and a menu child are being displayed, the padding between them in pixels.

XtNtopLevel

Indicates whether not the panel is a candidate for management by a window manager. This should always be set by the application. Otherwise, if XtNdisplayTitle is TRUE, the TitleBar child will be displayed. If XtNdisplayTitle is FALSE, the TitleBar child will not be displayed.

This resource should be set by the user in the resource defaults file. If the user runs the application without a window manager or with a non-titling window manager, this resource should be set to TRUE. If the user runs with a titling window manager this resource should be set to FALSE.

XtNvSpace

Padding between the top of the Panel and the top child in pixels, and between the bottom of the Panel and the bottom child in pixels.

XtNworkSpaceToSiblingPad

The padding between the work space child and the sibling above it. If there is no title nor menu being displayed this resource is ignored.

CONSTRAINT RESOURCES

The following resources will be attached to every widget inserted into Panel. Refer to CONSTRAINT(3X) for a general discussion of constraint resources.

Constraint Resource Set -- Children of PANEL(3X)			
Name	**Class**	**Type**	**Default**
XtNcausesResize	XtCCausesResize	Boolean	TRUE
XtNwidgetType	XtCWidgetType	XwWidgetType	XwUNKNOWN

XtNcausesResize

Controls whether changes in the child geometry can cause the Panel to make a geometry request of its parent. If TRUE for only one child, Panel will request changes whenever that child requests changes. If TRUE for multiple children, Panel will request changes whenever any of that set of children grow, and when all of that set of children have shrunk.

The behavior of this resource can be nullified by setting XwNLayout to XwIGNORE.

XtNwidgetType
Indicates to Panel which of the three areas (title, menu or workspace) the child should occupy. The possible values are, XwWORK_SPACE (specified in a resource defaults file as "work space"), XwTITLE (specified in a resource defaults file as "title"), XwPULLDOWN (specified in a resource defaults file as "pulldown"), and XwUNKNOWN (specified in a resource defaults file as "unknown"). XwUNKNOWN is the default.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNforeground	XtCForeground	Pixel	Black
XtNlayout	XtClayout	int	minimize
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

TRANSLATIONS

The default translation set defining is as follows:

```
<EnterWindow>:      enter()
<LeaveWindow>:      leave()
<FocusIn>:                                  focusIn()
```

ACTIONS

enter: If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE), initiate keyboard traversal.

focusIn: If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE), accept the keyboard focus and visually indicate it by highlighting the widget.

leave: If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE), terminate keyboard traversal.

ORIGIN

Hewlett-Packard Company.

XWPANEL(3X)

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwpopupMgrWidgetClass - the X Widgets PopupMgr widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/PopupMgr.h>
```

CLASSES

The PopupMgr widget is built from the Core, Composite, Constraint, Manager and MenuMgr classes. Note that the Constraint fields are not used in this widget and so are not listed in the resource tables below. Also, since the Composite class contains no user settable resources, there is no table for Composite class resources.

The widget class to use when creating an instance of the PopupMgr is **XwpopupMgrWidgetClass.** The class name is **PopupMgr**.

DESCRIPTION

The PopupMgr widget is a Composite widget which is used by an application to manage a collection of menupanes. Even though the PopupMgr is a Composite widget, it should never have any normal widget children. Instead, all of its children are popup shell children; the child of each of the popup shell widgets is a MenuPane. In addition, the parent of the PopupMgr must be a popup shell widget, whose parent is the widget to which the menu tree is being associated.

The PopupMgr manages a collection of MenuPane widgets, which have been organized into a hierarchical tree structure; the root of the tree is the top level menupane. When the user requests that the menu be posted, by generating a post event within the widget (or possibly one of the widget's children), the top level menupane is posted.

Once the PopupMgr has posted the top level MenuPane, it will remain posted until the user generates a select action; at that point, the MenuPanes will be removed from the display, and the selected menu button will perform any required actions. If the select occurs outside of a menu button, or if the user issues the menu unpost event, then the MenuPanes are simply unposted.

The PopupMgr supports a mode by which the menu hierarchy may be associated only with the specified widget, or it may be associated with the widget and all of its children (both present and future children). If the menu is associated with the widget and its children, then a menu post event that occurs in either the widget or one of its children will cause the menu to be posted.

The PopupMgr also supports a commonly used menu feature known as "sticky" menus. When operating in sticky menu mode, the PopupMgr will remember the last menu button selected by the user. The next time the user requests that the menu system be posted, all of the MenuPanes, up to the one containing the previously selected menu button, will be posted.

The PopupMgr provides a keyboard interface to the menus, through the use of keyboard accelerators, for posting the menu and for selecting a MenuButton from within one of the MenuPanes. This manager does not support keyboard mnemonics. When traversal is enabled, the standard keyboard traversal keys are also operational. Using the mouse while traversal is enabled may produce confusing results for the user. Operating in this fashion is therefore discouraged.

The PopupMgr provides the application writer with a global function that may be used to programmatically post a top level menupane at a particular position relative to a specified widget. The calling sequence and parameters are shown below:

```
XwPostPopup (menuMgr, relativeTo, x, y) XwPopupMgrWidget menuMgr;
Widget relativeTo;
Position x,y;
```

XwPostPopup() posts the top level menupane associated with the specified menu manager at the requested (x,y) position, relative to the specified widget. If the relativeTo parameter is set to NULL, then the position is assumed to be relative to the root window.

NEW RESOURCES

The PopupMgr defines a set of resource types used by the programmer to specify the data for the menu manager. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by PopupMgr.

PopupMgr Resource Set			
Name	**Class**	**Type**	**Default**
XtNpostAccelerator	XtCPostAccelerator	String	NULL
XtNstickyMenus	XtCStickyMenus	Boolean	FALSE

XtNpostAccelerator

This resource indicates the keyboard event that can be used to post the top level menupane. The string is specified using the syntax supported by the translation manager, with three exceptions. First, only a single event may be specified. Second, the event must be a KeyPress or KeyRelease event. Third, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the key event occurs.

XtNstickyMenus

This resource controls whether the menu manager operates in sticky menu mode.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

Menu Manager Resource Set -- XWMENUMGR(3X)			
Name	**Class**	**Type**	**Default**
XtNassociateChildren	XtCAssociateChildren	Boolean	TRUE
XtNkbdSelect	XtCKbdSelect	String	"<Key>Select"
XtNmenuPost	XtCMenuPost	String	"<Btn1Down>"
XtNmenuSelect	XtCMenuSelect	String	"<Btn1Up>"
XtNmenuUnpost	XtCMenuUnpost	String	NULL

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XwMENUMGR(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwprimitiveWidgetClass - the X Widget's primitive widget meta class

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

CLASSES

The Primitive widget class is built from the Core class.

DESCRIPTION

The Primitive class is an X Widget metaclass. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other widget classes. It handles border drawing and highlighting, traversal activation and deactivation and various callback lists needed by primitive widgets.

NEW RESOURCES

Primitive defines a set of resource types used by the programmer to specify the data for widgets which are subclasses of Primitive.

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

XtNbackgroundTile

This resource defines the tile to be used for the background of the widget. It defines a particular tile to be combined with the foreground and background pixel colors. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNbottomShadowColor

This resource defines the color that is combined with the bottom shadow tile and foreground color to create a pixmap used to draw the bottom and right sides of the border.

XtNbottomShadowTile

This resource defines the tile used in creating the pixmap used for drawing the bottom and right border shadow for the widget. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNforeground

This resource defines the foreground color for the widget. Widgets built upon this class can use the foreground for their drawing.

XtNhighlightColor

This resource defines the color to be used in the highlighting drawn by Primitive around the exterior of the widget.

XtNhighlightStyle

Two styles of border highlighting are supported by Primitive. They include drawing the highlighting with a pattern and widget specific border drawing. To set the highlight style through an arg list, use the #define XwPATTERN_BORDER. To set the highlight style through the .Xdefaults file, use the string "pattern_border".

For Widget Writers: The highlighting style of XwWIDGET_DEFINED is used exclusively by widgets with special highlighting requirements that need to override the normal highlighting types. To use, the widget inserts a highlight and unhighlight procedure into its primitive class and forces the highlightStyle field in the primitive instance to the define XwWIDGET_DEFINED. The primitive class will then make the appropriate calls to the highlight and dehighlight functions.

XtNhighlightThickness

The width of the highlight can be set using this resource. It is specified as an integer value representing the width, in pixels, of the highlight to be drawn. This value must be greater than or equal to 0. Note that highlighting takes place within the window created for a widget and is separate from the window border.

XtNhighlightTile

When the highlight style is XwPATTERN_BORDER, one of several tiles can be used for the drawing. The #defines for setting the values through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNrecomputeSize

This boolean resource indicates to a primitive widget whether it should recalculate its size when an application makes a XtSetValues call to it. If set to TRUE, the widget will perform its normal size calculations will may cause its geometry to change. If set to FALSE, the widget will not recalculate its size.

XtNrelease

This resource provides the means for registering callback routines that will be invoked when the widget built upon the Primitive class receives an event bound to the unselect action.

XtNselect

This resource provides the means for registering callback routines that will be invoked when the widget built upon the Primitive class receives an event bound to the select action.

XtNshadowOn

Primitive implements a drawing routine to draw a two pixel wide shadowing border around the widget inside of the highlight used for traversal. This resource indicates whether to take into account the additional space needed for drawing the shadow border. If set to TRUE the highlight thickness value of the widget will be automatically incremented by two.

The shadowed border is drawn with two pixmaps created using the following four resources. The border is split into two areas: the top-left shadow and the bottom-right shadow.

XtNtopShadowColor

This resource defines the color that is combined with the top shadow tile and foreground color to create a pixmap used to draw the top and left sides of the border.

XtNtopShadowTile

This resource defines the tile used in creating the pixmap used for drawing the top and left border shadow for the widget. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNtraversalType

Three modes of border highlighting activation are supported by Primitive. They are, no highlighting, highlight on the cursor entering the widget's window and highlight for keyboard traversal. The last mode is used by the keyboard traversal mechanism to indicate the widget that is to receive all input occurring within the widget hierarchy. To set the traversal type through an arg list, one of three defines can be used. They are XwHIGHLIGHT_OFF, XwHIGHLIGHT_ENTER and XwHIGHLIGHT_TRAVERSAL. The strings that can be used to set this resource through the .Xdefaults file are "highlight_off", "highlight_enter", and "highlight_traversal".

KEYBOARD TRAVERSAL

If the traversalType resource is set to highlight_traversal (either when the widget is created or during a call to XtSetValues) the Primitive widget's translation table is augmented with the following translations:

<FocusIn>:	focusIn()	
<FocusOut>:	focusOut()	
<Visible>:	visibility()	
<Unmap>:	unmap()	
<Key>Up:	traverseUp()	HP Up arrow key.
<Key>Down:	traverseDown()	HP Down arrow key.
<Key>Left:	traverseLeft()	HP Left arrow key.
<Key>Right:	traverseRight()	HP Right arrow key.
<Key>Next:	traverseNext()	HP "Next" key.
<Key>Prior:	traversePrev()	HP "Prev" key.
<Key>Home:	traverseHome()	HP Home arrow key.
<Key>KP_Enter:	traverseNextTop()	HP "Enter" key.

ACTIONS

focusIn:

If traversal is on for a widget of this class then accept the keyboard focus and visually indicate it by highlighting the widget.

focusOut:

If traversal is on for a widget of this class then indicate that the widget no longer has the focus by unhighlighting the widget.

traverseDown:

Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget below this one.

traverseHome:

Inform the parent of a widget of this class that it should transfer keyboard focus to the child which is closest to the upper left hand corner of the parent. If that child already has the keyboard focus, then ask the grandparent of the widget to give the keyboard focus to whichever of its children which is closest to the upper left hand corner.

traverseLeft:

Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget to the left of this one.

traverseNext:

Inform the parent of a widget of this class that it should transfer keyboard focus to the next child in the parent's list of children.

traverseNextTop:

Find the topmost parent in a widget of this class hierarch which is a subclass of XwManager and tell it to issues any XtNnextTop callbacks that have been registered with it. The purpose of this callback is to allow applications to move the keyboard focus between top level widget hierarchies of the same application.

traversePrev:
Inform the parent of a widget of this class that it should transfer keyboard focus to the previous child in the parent's list of children.

traverseRight:
Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget to the right of this one.

traverseUp:
Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget above this one.

unmap:
If traversal is on for a widget of this class and the widget that is focused becomes unmapped, then the focus moves to another mapped widget.

visibility:
If traversal is on for a widget of this class and the widget that is focused becomes hidden (e.g. another window obscures its visibility), then the focus moves to another visible widget.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWCREATETILE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwpulldownWidgetClass - the X Widgets pulldown menu manager widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/Pulldown.h>
```

CLASSES

The Pulldown menu manager widget is built from the Core, Composite, Constraint, Manager and MenuMgr classes. Note that the Constraint fields are not used in this widget and so are not listed in the resource tables below. Also, since the Composite class contains no user settable resources, there is no table for Composite class resources.

The widget class to use when creating an instance of the pulldown menu manager is **XwpulldownWidgetClass**. The class name is **Pulldown**.

DESCRIPTION

The Pulldown menu manager widget is a composite widget which is used by an application to manage a collection of MenuPanes. Even though the Pulldown menu manager is a Composite widget, it should never have any normal widget children. Instead, all of its children are popup shell children; the child of each of the popup shell widgets is a MenuPane. In addition, the parent of the Pulldown menu manager must be a popup shell widget, whose parent is the widget to which the menu tree is being associated.

The Pulldown menu manager manages a collection of MenuPane widgets, which have been organized into a hierarchical tree structure; the root of the tree is referred to as the top level menupane. The Pulldown menu manager creates a Pulldown widget as a child of the widget to which the menu tree is associated; as the menu tree is constructed, titlebuttons will be added to the Pulldown widget, thus providing the user with a means for posting a particular portion of the menu tree. As MenuPanes are added to the menu tree, if cascading submenus are allowed, then only those MenuPanes which cascade from the top level MenuPane will be folded up as a first level MenuPane with a new titlebutton within the PullDown widget. If cascading submenus are not allowed, then all cascading MenuPanes will be folded up into a first level MenuPane with a new titlebutton.

When the user requests that the menu be posted, by generating a post event within one of the titlebuttons, the MenuPane associated with the indicated titlebutton is posted. As soon as a select event or an unpost event is generated, the MenuPanes are unposted.

Once the Pulldown menu manager has posted a first level MenuPane it will remain posted until either the user generates a select action, the user generates an unpost action, or the user moves the cursor into a different titlebutton. If the select action occurs, then the MenuPanes will be removed from the display, and the appropriate menubutton will perform any required actions. If the select action occurs outside of a MenuButton, or if the unpost action is generated, then the MenuPanes are simply unposted. If the cursor was moved into a different titlebutton, then the menupanes associated with the previous titlebutton will be unposted, and the first level MenuPane for the new titlebutton will be posted.

The Pulldown menu manager supports a mode by which the menu hierarchy may be associated only with the specified widget, or it may be associated with the widget and all of its children (both present and future children). If the menu is associated with the widget and its children, then a keyboard accelerator which occurs in either the widget or one of its children, will cause the appropriate action to occur.

The Pulldown menu manager provides a keyboard interface to the menus, through the use of mnemonics and keyboard accelerators. A mnemonic may be used to post any of the first level menupanes; a posting mnemonic is issued by typing the appropriate mnemonic character in the presence of the modifiers specified by the postAccelerator resource. Keyboard accelerators are supported for selecting a MenuButton from within any of the MenuPanes; accelerators are always active, even if the corresponding MenuButton is not currently displayed. Keyboard mnemonics

may also be used for selecting a MenuButton; however, a MenuButton's mnemonic is only active if the MenuPane in which it resides in is currently displayed. The Pulldown menu manager only allows the first level Pulldown MenuPanes to have keyboard mnemonics for posting.

The Pulldown menu bar that is made visible to the user is actually a Frame widget that contains a RowCol widget. The RowCol widget handles the layout of the Pulldown menu buttons. If the user wishes to specify any of the attributes associated with the Frame, RowCol, or any of the Pulldown menu buttons from a defaults file, then the following naming conventions for these widgets should be used:

Frame :Pulldown manager name + "_frame"
RowCol:Pulldown manager name + "_bar"
Pulldown menu button :MenuPane name + "_pdbtn"

NEW RESOURCES

The Pulldown menu manager defines a set of resource types that may be used by the programmer to specify the data for the Pulldown menu manager. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by Pulldown.

Pulldown Resource Set			
Name	**Class**	**Type**	**Default**
XtNallowCascades	XtCAllowCascades	Boolean	TRUE
XtNpostAccelerator	XtCPostAccelerator	String	"Meta"
XtNpulldownBarId	XtCPulldownBarId	Widget	NULL

XtNallowCascades

This resource is used to control whether any of the top level pulldown menupanes may have other menupanes cascading from them. This resource must be set to the desired value when the menu manager widget is first created; it cannot be modified after the widget has been created.

XtNpostAccelerator

This resource is used to specify the keyboard modifiers which must be present when one of the post mnemonics is issued by the user. This resource must be set to the desired value when the menu manager widget is first created; it cannot be modified after the widget has been created.

XtNpulldownBarId

This resource is a read-only resource, and provides the application with the means for obtaining the widget Id for the frame widget which encloses the pulldown menubar widget. Applications should not use this to modify the attributes of the pulldown menubar. This resource is made available to allow applications to obtain the pulldown menubar Id, which is needed when attempting to add a pulldown menu to a widget which is not menu smart.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

Menu Manager Resource Set -- XWMENUMGR(3X)			
Name	**Class**	**Type**	**Default**
XtNassociateChildren	XtCAssociateChildren	Boolean	TRUE
XtNkbdSelect	XtCKbdSelect	String	"<Key>Select"
XtNmenuPost	XtCMenuPost	String	"<Btn1Down>"
XtNmenuSelect	XtCMenuSelect	String	"<Btn1Up>"
XtNmenuUnpost	XtCMenuUnpost	String	NULL

PULLDOWN BUTTON RESOURCES

The pulldown menu manager is responsible for managing the set of menupanes specified by the application, and for creating pulldown buttons within the pulldown menu bar, as needed. When creating the pulldown buttons, certain resources are inherited from the menupane from which the pulldown button is derived, while other resources are inherited from the menu manager. When an application modifies one of these resources within the menupane or the menu manager, the attribute will also be passed on to the associated pulldown button. The following tables outline those resources which are inherited from the menupane and those which are inherited from the menu manager:

Inherited MenuPane Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNfont	XtCFont	XFontStruct *	"fixed"
XtNforeground	XtCForeground	Pixel	Black
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground

Inherited Menu Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNshadowOn	XtCShadowOn	Boolean	TRUE

BUGS

The pulldown menu manager currently does not support keyboard traversal.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWMENUMGR(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwpushButtonWidgetClass - the X Widgets PushButton widget.

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/PushButton.h>

CLASSES

The PushButton widget is built from the Core, Primitive and Button classes.

The widget class to use when creating a PushButton is **XwpushButtonWidgetClass**. The class name is **PushButton**.

DESCRIPTION

The PushButton widget consists of a text label surrounded by a button border.

By default, button 1 down will invert the interior of the button: the background will be filled with the foreground color and the text will be written in the background color. Button 1 down also sets the button state to TRUE and issues any XtNselect callbacks that have been registered. Button 1 up will repaint the button in the normal state, set the button state to FALSE and issue any XtNrelease callbacks that have been registered. It is also possible (by setting the XtNinvertOnSelect resource to FALSE) to make the Button 1 down and up sequence invert only the top and bottom shadows (instead of inverting the entire button). This mode is very useful on color systems, and can give the button a three-dimensional appearance.

As mentioned above, the XtNselect and XtNrelease callbacks can be attached to this widget. This widget can also be set to respond to Enter and Leave window events by highlighting and unhighlighting the button. This widget is also capable of handling keyboard traversal. See the translations below for the default traversal keycodes.

A final feature is that by setting the XtNtoggle resource to TRUE the PushButton can be made to act like a toggle button.

NEW RESOURCES

The PushButton widget class defines a set of resource types that can be used by the programmer to specify data for widgets of this class. Recall that the string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNfont becomes font.

PushButton Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNinvertOnSelect	XtCInvertOnSelect	Boolean	TRUE
XtNtoggle	XtCToggle	Boolean	FALSE

XtNinvertOnSelect

Forces the button to invert the foreground/background on selection if set to TRUE, otherwise switch only the top and bottom shadow colors.

XtNtoggle

If set to TRUE makes the PushButton act like a toggle button.

INHERITED RESOURCES

The following resources are inherited from the named superclasses. The defaults used for the PushButton when being created are as follows:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

Button Resource Set -- XWBUTTON(3X)			
Name	**Class**	**Type**	**Default**
XtNfont	XtCFont	XFontStruct *	Fixed
XtNhSpace	XtCHSpace	int	2
XtNlabel	XtCLabel	caddr_t	NULL
XtNlabelLocation	XtCLabelLocation	int	right
XtNsensitiveTile	XtCSensitiveTile	int	75_foreground
XtNset	XtCSet	Boolean	FALSE
XtNvSpace	XtCVSpace	int	2

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHTLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard

traversal. See the XwPrimitive man page for a complete description of these translations. See the TRANSLATIONS section in this man page for a description of the translations local to the PushButton widget.

TRANSLATIONS

The input to the PushButton is driven by the mouse buttons. The default translation set defining this button is as follows:

<Btn1Down>:	select()	
<Btn1Up>:	unselect()	
<EnterWindow>:	enter()	
<LeaveWindow>:	leave()	
<KeyDown>Select:	select()	HP "Select" key
<KeyUp>Select:	unselect()	HP "Select" key

ACTIONS

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the button will be highlighted. Otherwise no action is taken.

leave: If the XtNtraversalType resources has been set to XwHIGHLIGHT_ENTER then the button will be unhighlighted. Otherwise no action is taken. Note that this widget contains some actions which are not bound to any events by the default translations. The purpose of these additional actions are to allow advanced users to alter the button semantics to their liking.

select: Select sets the state of the button to TRUE. It also redraws the PushButton to reflect the current setting. It then issues any XtNselect callbacks which have been registered. No additional data beyond the widget id and the specified closure is sent with these callbacks.

toggle: Toggle the set state of the button (make it TRUE if it was FALSE, FALSE if it was TRUE). Redraw the PushButton to reflect the current button setting (if set, invert the button, otherwise draw normally). If the current state of the button is set (TRUE) issue the XtNselect callbacks, if not set (FALSE) issue the XtNrelease callbacks. No additional data beyond the widget id and the specified closure is sent with these callbacks.

unselect:
Release sets the state of the button to FALSE. It also redraws the PushButton to reflect the current setting. It then issues any XtNrelease callbacks which have been registered. No additional data beyond the widget id and the specified closure is sent with these callbacks.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWBUTTON(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWREGISTERCONVERTERS(3X)

NAME

XwRegisterConverters - register all of the resource converters used by the X Widgets.

SYNOPSIS

```
#include <X11/Atoms.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

void XwRegisterConverters ()
```

DESCRIPTION

XwRegisterConverters is used by widget writers to register all of the resource type converters used by the X Widgets. The call to this routine is made within a widget's ClassInitialize procedure that has added a resource converter to the source file containing this function. XwRegisterConverters ensures that resource converters it is registering are only registered once.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwrowColWidgetClass - the X Widgets row/column manager widget.

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/RCManager.h>

CLASSES

The row column manager widget is built from the Core, Constraint and Manager classes. Since the Composite class contains no resources that the user can set, there is no table for Composite class resources.

The widget class to use when creating a row column manager is **XwrowColWidgetClass**. The class name is **RowCol**.

DESCRIPTION

The row column widget is a composite widget that supports three types of row column layouts for its children: requested columns, maximum columns, and maximum unaligned. With requested columns, the application specifies the number of columns (the default is one) to be used in laying out the data. The children are laid out in rows. Columns are as wide as the widest element in the column and all elements are left justified. Row height is determined by the largest element in the row and all elements are centered in the row. The second layout type, maximum columns, automatically calculates the maximum number of columns that can fit within the manager and lays the children out accordingly. The last layout type, maximum unaligned, does not force any columnar alignment. A child being positioned is placed immediately to the right of the previous child until a row is full, then a new row is started at the left edge of the manager immediately below the previous row.

In addition to the row column ordering, this manager widget supports three different layout policies: minimize (the default), maximize and ignore. These policies are specified by the XtNlayout resource that is inherited from the Manager resource set. When the layout policy is set to minimize, the manager will create a box that is just large enough to contain all of its children, regardless of any provided width and height values. When the given width and height values would create a box larger than needed, the maximize setting will use this additional space as padding between elements. When using the maximize setting, if one or both of the height width and values are too small, the manager will grow to accommodate all of the children. When using the ignore policy, the row column manager will not grow or shrink in response to the addition, deletion or altering of its children.

The row column widget also implements two selection policies. The default is "n_of_many", and the alternative is "one_of_many." The "n_of_many" policy does not require any action on the part of the manager widget. It allows any or all of its children to be selected and performs no action in response to their selection. The "one_of_many" policy *only* applies to child widgets that are subclasses of the XwPrimitive class. When "one_of_many" is the active policy, a callback of type XtNselect is added to each child widget. Then, when a child is selected, the manager is informed. The manager keeps track of the previously active child and directly invokes a release procedure in that child so that it becomes unselected. The "one_of_many" mode will not activate a child if none are active and will not disallow the selection of an active child causing it to become deactive. Thus, if a strict "one_of_many" mode is desired, the application will have to enforce it.

NEW RESOURCES

The row column manager defines a set of resource types used by the programmer to specify data for the manager widget. The programmer can also set the values for the Core, Composite and XwManager widget classes to set attributes for this widget. The following table contains the settable resources defined by the row column manager. The string to be used when setting any of these resources in an application defaults file (such as .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNvSpace becomes vSpace.

Row Column Resource Set			
Name	**Class**	**Type**	**Default**
XtNcolumns	XtCColumns	int	1
XtNforceSize	XtCForceSize	Boolean	FALSE
XtNhSpace	XtCHSpace	int	4
XtNlayoutType	XtCLayoutType	int	requested_columns
XtNmode	XtCMode	int	n_of_many
XtNsingleRow	XtCSingleRow	Boolean	FALSE
XtNvSpace	XtCVSpace	int	4

XtNcolumns

The application can specify the number of columns to be used when laying out the widgets children.

XtNforceSize

The application has the option of forcing the widths of each widget in a column and the heights of each widget in a row to be the same. This can be used, for example to enforce an orderly layout for a group of buttons. For the layout type of maximum unaligned, only the heights of the widgets in a row are forced to the same size.

XtNhSpace

The application may determine the number of pixels of space left between each element within a given row. This defines a minimum spacing.

XtNlayoutType

The application can specify the type of layout the row column manager is to perform. Allowable argument list settings are XwREQUESTED_COLUMNS, XwMAXIMUM_COLUMNS and XwMAXIMUM_UNALIGNED. To set this value in .Xdefaults or another resource file use the strings requested_columns, maximum_columns and maximum_unaligned.

XtNmode

The application can specify whether the selection policy is n_of_many or one_of_many. Allowable argument list settings are XwONE_OF_MANY and XwN_OF_MANY. To set this value in .Xdefaults or another resource file use the strings one_of_many and n_of_many.

NOTE: The RowCol class provides a specialized "insert child" procedure. This procedure allows an application to provide a special argument list type XtNchildPosition with an integer value. This value specifies the position within the child list the new widget will be inserted.

XtNsingleRow

For layout types of maximum columns and maximum unaligned, the application has the option of having the row column manager to try to lay itself out in a single row whenever one of its children makes a geometry request.

XtNvSpace

The application may determine the number of pixels of space left between each column. This defines a minimum spacing.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwsashWidgetClass - an X Widgets utility widget

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Sash.h>

CLASSES

The sash widget is built from the Core and Primitive classes.

The widget class to use when creating a sash is **XwsashWidgetClass**. The class name is **Sash**.

DESCRIPTION

The sash widget is a utility widget used by the vertical paned manager XwVPaned to control the sizes of the individual panes. In its realized form it appears as a square box of its background color. When the pointer is moved into the sash the cursor is changed to the crosshair cursor.

Callbacks can be attached to the widget to report selection (XtNselect) and unselection (XtNrelease). This widget can be set to respond to Enter and Leave window events by highlighting and unhighlighting the sash. This widget is also capable of handling keyboard traversal. (While you can traverse to the Sash in the current widget library, Sash does not handle keyboard input.) See the translations below for the default traversal keycodes.

NEW RESOURCES

The sash widget class defines one additional resource which is detailed in the table below. The programmer should refer to the man pages for the sash's superclasses to determine available resources and their defaults.

Sash Resource Set			
Name	**Class**	**Type**	**Default**
XtNcallback	XtCCallback	caddr_t	NULL

XtNcallback

This is used by the paned window widget to be informed of button presses and mouse movement associated with the sash.

INHERITED RESOURCES

The following resources are inherited from the named superclasses: The defaults used for the sash when being created are as follows:

XWSASH(3X)

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNtraversalType	XtCTraversalType	int	highlight_off

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHTLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the Primitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. See the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the sash widget.

TRANSLATIONS

The input to the sash is driven by the mouse buttons. The default translation set defining this button is listed below. Note that for the specific key symbols used in traversal, the HP Key Cap which corresponds to this key symbol appears to the right of the definition.

```
<Btn1Down>:      SashAction(Start, UpperPane)
<Btn2Down>:      SashAction(Start, ThisBorderOnly)
<Btn3Down>:      SashAction(Start, LowerPane)
<Btn1Motion>:    SashAction(Move, Upper)
<Btn2Motion>:    SashAction(Move, ThisBorder)
<Btn3Motion>:    SashAction(Move, Lower)
Any<BtnUp>:      SashAction(Commit)
<EnterWindow>:   enter()
<LeaveWindow>:   leave()
```

ACTIONS

SashAction(Start, UpperPane):

Change the cursor from the crosshair to an upward pointing arrow. Determine the upper pane which will be adjusted (usually the pane to which the sash is attached).

SashAction(Start, ThisBorderOnly):
Change the cursor from the crosshair to a double headed arrow. The panes that will be adjusted are the pane to which the sash is attached and the first pane below it that can be adjusted. Unlike the UpperPane and LowerPane mode, only 2 panes will be affected. If one of the panes reaches its minimum or maximum, adjustment will stop, instead of finding the next adjustable pane.

SashAction(Start, LowerPane):
Change the cursor from the crosshair to a downward pointing arrow. Determine the lower pane which will be adjusted (usually the pane below the pane to which the sash is attached).

SashAction(Move, Upper):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget below the upper pane can be adjusted and make the appropriate adjustments.

SashAction(Move, ThisBorder):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Adjust as needed (and as possible) the upper and lower panes selected when the SashAction(Start, ThisBorderOnly) action was invoked.

SashAction(Move, Lower):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget above the lower pane can be adjusted and make the appropriate adjustments.

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the button will be highlighted. Otherwise no action is taken.

leave: If the XtNtraversalType resources has been set to XwHIGHLIGHT_ENTER then the button will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWVPANED(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWSCROLLBAR(3X)

NAME

XwscrollBarWidgetClass - the X Widget's scrollbar widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Valuator.h>
#include <Xw/Arrow.h>
#include <Xw/ScrollBar.h>
```

CLASSES

The ScrollBar is built from the Core, Composite, and Manager classes.

The widget class to use when creating a scrollbar is **XwscrollBarWidgetClass**. The class name for scrollbar is **ScrollBar**.

DESCRIPTION

The ScrollBar widget combines the Valuator and Arrow widgets to implement a horizontal or vertical scrolling widget containing a valuator and an arrow on each end of the valuator.

As with the Valuator, input is supported through interactive slider movement and selections on the slide area not occupied by the slider. Both types of input have a separate callback list for communicating with the application. The arrows on each end of the valuator control additional input to the valuator. When an arrow is selected, the slider within the valuator will be moved in the direction of the arrow by an application supplied amount. If the button is held down, the slider will continue to move at a constant rate.

The ScrollBar can be used by the application to attach to objects scrolled under application control, or used by composite widgets to implement predefined scrolled objects.

NEW RESOURCES

The ScrollBar defines a set of resource types used by the programmer to specify the data for the scrollbar. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by ScrollBar.

ScrollBar Resource Set			
Name	**Class**	**Type**	**Default**
XtNgranularity	XtCGranularity	int	2
XtNinitialDelay	XtCInitialDelay	int	500
XtNrepeatRate	XtCRepeatRate	int	100

XtNgranularity

This resource defines the increment in the valuator's coordinate system to move the slider while continuous scrolling.

XtNinitialDelay

The ScrollBar supports smooth time sequenced movement of the slider when a selection occurs on the arrows. This resource defines the amount of delay to wait between the initial selection and the slider starting its repetitive movement. The value is defined in milliseconds.

XtNrepeatRate

This resource defines the continuous repeat rate to use to move the slider while the button is being held down on an arrow. The value is also defined in milliseconds.

INCORPORATED RESOURCES

The ScrollBar creates itself by internally creating two Arrow widgets and a Valuator. As such, it uses a large number of the resources defined by these widgets. Many of the attributes for these

widgets can be set through the .Xdefaults file or by use of XtSetValues() when communicating with the ScrollBar.

It should be noted, that only the resources within the following tables will have any effect on the valuator or arrows. The other resource types defined by the Valuator and Arrow widgets are either overridden or unused by ScrollBar.

The following tables list the resources incorporated by ScrollBar. For a complete description of these resources, refer to the manual page listed in the table heading.

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

Valuator Resource Set -- XWVALUATOR(3X)			
Name	**Class**	**Type**	**Default**
XtNareaSelected	XtCCallback	Pointer	NULL
XtNsliderExtent	XtCSliderExtent	int	10
XtNsliderMax	XtCSliderMax	int	100
XtNsliderMin	XtCSliderMin	int	0
XtNsliderMoved	XtCCallback	Pointer	NULL
XtNslideOrientation	XtCSlideOrientation	int	vertical
XtNsliderOrigin	XtCSliderOrigin	int	0
XtNsliderReleased	XtCCallback	Pointer	NULL

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

XWSCROLLBAR(3X)

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to True at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWPRIMITIVE(3X), XWCREATETILE(3X), XWVALUATOR(3X), XWARROW(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwscrolledWindowWidgetClass - the X Widget's scrolled window widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Valuator.h>
#include <Xw/Arrow.h>
#include <Xw/ScrollBar.h>
#include <Xw/SWindow.h>
```

CLASSES

The ScrolledWindow is built from the Core, Composite, and Manager classes.

The widget class to use when creating a scrolled window is **XwscrolledWindowWidgetClass.** The class name is **ScrolledWindow.**

DESCRIPTION

The ScrolledWindow widget combines the ScrollBar and BulletinBoard widgets 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 scroll bars.

To use the scrolled window, an application first creates a ScrolledWindow widget and then creates a widget capable of displaying the desired data as a child of the ScrolledWindow. ScrolledWindow will position the child widget within its BulletinBoard manager instance and create scroll bars for the horizontal and vertical dimensions. When the user performs some action on the scroll bars, the child widget will be repositioned accordingly within the bulletin board.

NEW RESOURCES

The ScrolledWindow widget defines a unique set of resource types that can be used by the programmer to control the appearance and behavior of the scrolled window. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by ScrolledWindow.

ScrolledWindow Resource Set			
Name	**Class**	**Type**	**Default**
XtNborderPad	XtCBorderPad	int	4
XtNforceHorizontalSB	XtCForceHorizontalSB	Boolean	FALSE
XtNforceVerticalSB	XtCForceVerticalSB	Boolean	FALSE
XtNhsbHeight	XtCHsbHeight	int	20
XtNhsbY	XtCHsbY	int	0
XtNhScrollEvent	XtCCallback	Pointer	NULL
XtNhsliderExtent	XtCHsliderExtent	int	0
XtNhsliderMax	XtCHsliderMax	int	0
XtNhsliderMin	XtCHsliderMin	int	0
XtNhsliderOrigin	XtCHsliderOrigin	int	0
XtNinitialX	XtCInitialX	int	0
XtNinitialY	XtCInitialY	int	0
XtNvScrollEvent	XtCCallback	Pointer	NULL
XtNvsliderExtent	XtCVsliderExtent	int	0
XtNvsliderMax	XtCVsliderMax	int	0
XtNvsliderMin	XtCVsliderMin	int	0
XtNvsliderOrigin	XtCVsliderOrigin	int	0
XtNvsbWidth	XtCVsbWidth	int	20
XtNvsbX	XtCVsbX	int	0

XWSCROLLEDWINDOW(3X)

XtNborderPad
This is an integer that defines the number of pixels between the scrollbars and the viewable area of the scrolled window. The default padding is four pixels.

XtNforceHorizontalSB
When the child widget is created and positioned within the scrolled window, its width and height are examined. If the entire child widget will fit within the width of the scrolled window, the horizontal scrollbar will not be created, since there is no need to scroll in that direction. Setting this resource to TRUE disables this checking and will force a horizontal scrollbar to be attached to the window regardless of the dimension of the child widget.

XtNforceVerticalSB
This resource controls the existence of the vertical scrollbar. As described above, if this is set to TRUE a vertical scrollbar will always be created.

XtNhsbHeight
This is the height in pixels of the horizontal scroll bar.

XtNhsbY
This is a read-only resource that returns the position of the horizontal scrollbar. If this value is less than the current height of the scrolled window, the scrollbar will be visible to the user.

XtNhScrollEvent
An application program may track the position of the child within the scrolled window by linking into these callbacks. Whenever the user moves the valuator in either scroll bar, ScrolledWindow moves the child accordingly and then calls the appropriate callback. The call_data parameter is set to the new valuator origin for the scrollbar.

XtNhSliderExtent, XtNhSliderMax, XtNhSliderMin, XtNhSliderOrigin
These values correspond to the min, max, extent, and origin resources for the horizontal scrollbar. They are designed to be set only on initialization. Any subsequent resize or other geometry activity will force the scrolled window to recompute these values. Writing new values into these resources after initialization could cause toolkit warnings to occur. The exception is the XtNhSliderOrigin resource. This resource can be set at any time to cause the child to be horizontally scrolled as though the user had moved the valuator to that position.

XtNinitialX and **XtNinitialY**
The child widget is initially positioned at (0,0) within the bulletin board. This positioning can be changed by specifying a new X and Y location. If a non-zero value is given, that becomes the initial location, and the valuators inside the scrollbars are adjusted to give a visual indication of the new offset. **This value should be negative to assure proper operation of the scrolled window.** These resources are only used at initialization time; they cannot be set through a call to XtSetValues.

XtNvScrollEvent
An application program may track the position of the child within the scrolled window by linking into these callbacks. Whenever the user moves the valuator in either scroll bar, ScrolledWindow moves the child accordingly and then calls the appropriate callback. The call_data parameter is set to the new valuator origin for the scrollbar.

XtNvSliderMin, XtNvSliderMax, XtNvSlideExtent, XtNvSliderOrigin
These values correspond to the min, max, extent, and origin resources for the vertical scrollbar. They are designed to be set only on initialization. Any subsequent resize or other geometry activity will force the scrolled window to recompute these values. Writing new values into these resources after initialization could cause toolkit warnings to occur. The exception is the XtNvSliderOrigin resource. This resource can be set at any time to cause the child to be vertically scrolled as though the user had moved the valuator to that position.

XtNvsbWidth
This is the width in pixels of the vertical scroll bar.

XtNvsbX

This is a read-only resource that returns the position of the vertical scrollbar. If this value is less than the current width of the scrolled window, the scrollbar will be visible to the user.

INCORPORATED RESOURCES

The ScrolledWindow widget is built from two ScrollBar widgets and a BulletinBoard widget. As such, it uses a large number of the resources defined by these widgets. Many of the attributes for these widgets can be set through the .Xdefaults file or by use of XtSetValues() when communicating with the ScrolledWindow widget.

Only the resources within the following tables will have any effect on the scroll bars. The other resource types defined by the ScrollBar widget are either overridden or unused by ScrolledWindow.

The following tables list the resources incorporated by ScrolledWindow. For a complete description of these resources, refer to the manual page listed in the table heading.

ScrollBar Resource Set -- XWSCROLLBAR(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNgranularity	XtCGranularity	int	10
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNinitialDelay	XtCinitialDelay	int	500
XtNrepeatRate	XtCRepeatRate	int	100
XtNtraversalType	XtCTraversalType	int	highlight_off

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X) XWPRIMITIVE(3X),
XWSCROLLBAR(3X),XWBULLETINBOARD(3X),XWVALUATOR(3X), XWARROW(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XtNsliderMin	XtCSliderMin	int	0		
XtNsliderMax	XtCSliderMax	int	100		
XtNsliderExtent	XtCSliderExtent	int	10		
XtNsliderOrigin	XtCSliderOrigin	int	0		

NAME

XwstaticRasterWidgetClass - The HP X Widget's static image widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/SRaster.h>
```

CLASSES

The static raster widget is built from the Core and Primitive classes.

The widget class to use when creating a static raster is **XwstaticRasterWidgetClass.** The class name is **StaticRaster.**

DESCRIPTION

The static raster widget provides an uneditable raster image. As a default, the image is placed in a window that is exactly the size of the raster (plus the border width). The image can be dynamically resized. If the window is enlarged from its original size, the image will be redrawn in the center of the new window. If the window shrinks below the size of the raster, the image is clipped on the right and bottom sides as needed to fit within the new boundries.

The raster image is provided to the widget in the form of an XImage data structure. New data can be displayed by specifying a new XImage structure, or by changing the pointer to the bitmap data within that structure.

Callbacks can be attached to the widget to report selection (XtNselect) and unselection (XtNrelease). This widget can be set to respond to Enter and Leave window events by highlighting and unhighlighting the border.

NEW RESOURCES

StaticRaster defines several new resources. (To reference a resource in a .Xdefaults file, strip off the XtN from the resource string.)

StaticRaster Resource Set			
Resource	**Class**	**Type**	**Default**
XtNinvertOnSelect	XtCInvertOnSelect	Boolean	TRUE
XtNset	XtCSet	Boolean	FALSE
XtNshowSelected	XtCIShowSelected	Boolean	TRUE
XtNsRimage	XtCSRimage	XImage *	NULL

XtNinvertOnSelect

If this resource is TRUE, the raster image will invert its foreground and background colors when selected, and return to normal when unselected.

XtNset This is a Boolean resource which indicates whether the raster is currently selected (TRUE) or not (FALSE).

XtNshowSelected

If TRUE, this will cause the image to appear to be indented when selected, and raised when unselected.

XtNsRimage

This is a pointer to an XImage data structure.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

XWSTATICRASTER(3X)

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. See the TRANSLATIONS section in this man page for a description of the translations local to the static raster widget.

TRANSLATIONS

The static raster is affected by the mouse buttons and cursor motion. The default translation set is as follows:

```
<Btn1Down>:     select(),
<Btn1Up>:       release(),
<KeyDown>:      select(),
<KeyUp>:        release(),
<EnterWindow>:  enter(),
<LeaveWindow>:  leave(),
```

ACTIONS

enter: Causes the border to be highlighted if enabled.

leave: Causes the border to be highlighted if enabled.

release:
Allows an application to be notified of the event via the callback structure.

select: Allows an application to be notified of the event via the callback structure.

NOTES

Error checking on the XImage structure is minimal, so weird rasters can result from incorrect or incomplete data.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwstaticTextWidgetClass - An X Widget for displaying StaticText.

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/SText.h>

CLASSES

The StaticText widget is built from the Core and Primitive classes.

The widget class to use when creating a StaticText widget is **XwstaticTextWidgetClass**.

The class name for StaticText is **StaticText**.

DESCRIPTION

StaticText provides an uneditable block of text. Optionally, StaticText will provide simple heuristics to fit the text into arbitrarily sized windows. Imbedded new-line characters in the string are always honored. Stripping of leading and trailing spaces is optional. Alignment of text is provided by the XtNalignment resource. If the window is larger than necessary for the given text, the text can be positioned at various points within the window by using the XtNgravity resource. The alignment specified by XtNalignment is kept regardless of the setting of XtNgravity. For example, if XtNalignment is specified as "left" in a defaults file and XtNgravity is specified as "SouthEastGravity," the text will be positioned in the southeast corner of the box and will be aligned on the left edge of the text.

If the StaticText widget is directed to become larger than is needed for the text, the text will be centered in the window. The text will retain the specified alignment.

If the StaticText widget is directed to become narrower than is necessary for the text, the text may be wrapped (depending on XtNWrap) or clipped to the right and/or left (depending on the XtNalignment).

If the StaticText widget is directed to become shorter than is necessary for the text, the text will be clipped on the bottom.

When the text is wrapped, StaticText will try to break lines on spaces. The space on which the line is broken is temporarily converted to a newline.

NEW RESOURCES

To specify any of these resources within a resource defaults file, simply drop the *XtN* prefix from the resource name. StaticText defines the following new resources:

StaticText Resource Set			
Name	**Class**	**Type**	**Default**
XtNalignment	XtCAlignment	XwAlignment	XwALIGN_LEFT
XtNfont	XtCFont	XFontStruct *	Fixed
XtNgravity	XtCGravity	int	CenterGravity
XtNhSpace	XtCHSpace	int	2
XtNlineSpace	XtCLineSpace	int	0
XtNrecomputeHeight	XtCrecomputeHeight	Boolean	FALSE
XtNrecomputeWidth	XtCrecomputeWidth	Boolean	FALSE
XtNstring	XtCString	char *	NULL
XtNstrip	XtCStrip	Boolean	TRUE
XtNvSpace	XtCVSpace	int	2
XtNwrap	XtCWrap	Boolean	TRUE

XtNalignment

This specifies the alignment to be applied when drawing the text. The alignment resource is interpreted without regard to case.

Alignment never causes leading or trailing spaces to be stripped.

Alignment may have the following values and effects:

XwALIGN_LEFT will cause the left sides of the lines to be vertically aligned. Specified in resource default file as "Left".

XwALIGN_CENTER will cause the centers of the lines to be vertically aligned. Specified in resource default file as "Center".

XwALIGN_RIGHT will cause the right sides of the lines to be vertically aligned. Specified in resource default file as "Right".

XtNfont

This resource controls which font the text will be drawn in.

XtNgravity

This resource controls the use of extra space within the widget. No matter how the text is aligned, the gravity resource determines where the text will be placed within the window when there is more space than is needed to display the text.

CenterGravity will cause the string to be centered in the extra space. Specified in the resource defaults file as "CenterGravity". This is the default setting.

NorthGravity will cause the string to always to be at the top of the window centered in any extra width. Specified in the resource defaults file as "NorthGravity".

SouthGravity will cause the string to always to be at the bottom of the window centered in any extra width. Specified in the resource defaults file as "SouthGravity".

EastGravity will cause the string to always be at the right of the window centered in any extra height. Specified in the resource defaults file as "EastGravity".

WestGravity will cause the string to always be at the left of the window centered in any extra height. Specified in the resource defaults file as "WestGravity".

NorthWestGravity will cause the string to always be in the upper left corner of the window. Specified in the resource defaults file as "NorthWestGravity".

NorthEastGravity will cause the string to always be in the upper right corner of the window. Specified in the resource defaults file as "NorthEastGravity".

SouthWestGravity will cause the string to always be in the lower left corner of the window. Specified in the resource defaults file as "SouthWestGravity".

SouthEastGravity will cause the string to always be in the lower right corner of the window. Specified in the resource defaults file as "SouthEastGravity".

XtNhSpace

This specifies the number of pixels to maintain between the text and the highlight area to the right and left of the text.

XtNlineSpace

This resource controls the amount of space between lines. It is specified as a percentage of the font height. This space is added between each line of text. XtNlineSpace may be negative to a maximum of -100 (which causes all lines to overwrite each other).

XtNrecomputeHeight

If set to TRUE, this resource tells the widget to compute the optimum height for the data that is to be displayed within the widget. The widget then makes a request to its parent widget to grow to this height. This resource is ignored if XtNrecomputeSize is TRUE.

XtNrecomputeWidth

If set to TRUE, this resource tells the widget to compute the optimum width for the data that is to be displayed within the widget. The widget then makes a request to its parent widget to grow to this width. This resource is ignored if XtNrecomputeSize is TRUE.

XtNstring

This resource is the string which will be drawn. The string must be null terminated. If the string is given in a resource defaults file, newlines may be specified by "\n" within the string.

XtNstrip

This resource controls the stripping of leading an trailing spaces during the layout of the text string. If XtNstrip is FALSE, spaces are not stripped. If XtNstrip is TRUE and XtNalignment is XwALIGN_LEFT, leading spaces are stripped from each line. If XtNstrip is TRUE and XtNalignment is XwALIGN_CENTER, both leading and trailing spaces are stripped from each line. If XtNstrip is TRUE and XtNalignment is XwALIGN_RIGHT, trailing spaces are stripped from each line.

XtNvSpace

This specifies the number of pixels to maintain between the text and the highlight area to the top and bottom of the text.

XtNwrap

This resource controls the wrapping of lines within the widget. If XtNwrap is TRUE, lines which are too long are broken on spaces. The spaces are converted to new-lines to break the line. Imbedded new-lines are honored. If there is too much text for the specified window size, it will be clipped at the bottom.

If XtNwrap is FALSE, lines which are too long will be clipped according to the alignment. An XtNalignment value of XwALIGN_LEFT will cause lines which are too long to be clipped to the right. An XtNalignment value of XwALIGN_RIGHT will cause lines which are too long to be clipped to the left. An XtNalignment value of XwALIGN_CENTER will cause lines to be clipped equally on both the right and the left.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

TRANSLATIONS

The input to the toggle is driven by the mouse buttons. The default translation set defining this button is listed below. Note that for the specific key symbols used in traversal, the HP Key Cap which corresponds to this key symbol appears to the right of the definition.

```
<EnterWindow>:      enter()
<LeaveWindow>:      leave()
<KeyDown>Select:    select()    HP "Select" key
<KeyUp>Select:      release()   HP "Select" key
```

ACTIONS

XWSTATICTEXT(3X)

enter If the XtNtraversalType resource has been set to XwHIGHLIGHT_OFF then the StaticText will be highlighted. Otherwise no action is taken.

leave If the XtNtraversalType resources has been set to XwHIGHLIGHT_OFF then the StaticText will be unhighlighted. Otherwise no action is taken.

release Invokes the release callbacks.

select Invokes the select callbacks.

NOTES

The forced new line is the '\n' character constant as defined by the C compiler. Fonts which do not use that character constant for the newline will not be handled correctly by StaticText.

StaticText will assume that the space is the ' ' character constant as defined by the C compiler. Fonts which do not use that character constant for spaces will not be handled correctly by StaticText.

Non-8-bit character representations have undefined effects on the operation of StaticText.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwtextEditWidgetClass - An X Widget for viewing and editing text.

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/TextEdit.h>

CLASSES

TextEdit is built from the Core and Primitive classes.

The widget class record to use when creating a text edit widget is **XwtextEditWidgetClass**.

The class name for TextEdit is **TextEdit**.

OVERVIEW

TextEdit provides a single and multi-line text editor which has both a customizable user interface and a programmatic interface. It can be used for single-line string entry, forms entry with verification procedures, multiple-page document viewing, and full-window editing. It provides an application with a consistent editing paradigm for entry of textual data.

The display of the textual data on the screen can be adjusted to the application writer's need based on four class resources, XtNwrap, XtNwrapBreak, XtNscroll, and XtNgrow. XtNwrapP controls automatic line breaking for lines that extend off the end of the screen. XtNscroll controls horizontal and vertical shifting of the text when the insertion cursor moves off the screen. XtNgrow controls attempts by the widget to resize its window to make more room for text that extends beyond the current screen size. These resources are explained in detail below.

TextEdit provides separate callback lists to verify insertion cursor movement, modification of the text, and leaving the TextEdit widget. Each of these callbacks provides the verification function with the widget instance, the event that caused the callback, and a data structure specific to the verification type. From this information the function can verify if the application considers this to be a legitimate state change and signal the widget whether to continue with the action. The verification function can also manipulate the widget through the class methods defined by the TextEdit class. The verification callback lists are explained in detail below.

The user interface can be tailored by providing a new set of translations. The default translations provide commands for movement, deletion, killing and selection with key bindings similar to an EMACS style editor.

TextEdit allows the user to select regions of text. By using TextEdit's selection mechanism, application writers can easily fit instances of TextEdit into X11's current selection mechanism.

The TextEdit class controls the data structures for drawing the text on the screen and defines the functions that manipulate that data. The storage of the text is provided by a separate component called the Source. The Source provides the storage of the textual data and a set of functions for querying and changing that data. The application writer can provide a new source for the TextEdit widget. The details are provided below.

NEW RESOURCES

TextEdit defines the following new resources:

NOTE: Resource values can be set in resource defaults files (such as the .Xdefaults) by stripping the "Xw" from the value and replacing any remaining upper case characters with lower case. For example, to set the value XwSoftWrap for the resource XtNwrap, use "softwrap."

TextEdit Resource Set			
Name	**Class**	**Type**	**Default**
XtNbottomMargin	XtCMargin	Dimension	2
XtNdisplayPosition	XtCTextPosition	XtTextPosition	0
XtNexecute	XtCallback	XtCallbackProc	NULL
XtNextendKey	XtCExtendKey	int	XwMeta
XtNgrow	XtCGrow	XwGrow	XwGrowOff
XtNinsertPosition	XtCTextPosition	XtTextPosition	0
XtNleaveVerification	XtCallbackProc	XtRCallback	NULL
XtNleftMargin	XtCMargin	Dimension	5 (See Note Below)
XtNmodifyVerification	XtCallbackProc	XtRCallback	NULL
XtNmotionVerification	XtCallbackProc	XtRCallback	NULL
XtNrightMargin	XtCMargin	Dimension	5 (See Note Below)
XtNscroll	XtCScroll	XwScroll	XwAutoScrollVertical
XtNselection	XtCSelection	XwTextSelection*	NULL
XtNsourceType	XtCSourceType	XwSourceType	"stringsrc"
XtNtextSource	XtCTextSource	Pointer	NULL
XtNtopMargin	XtCMargin	Dimension	2
XtNwrap	XtCWrap	XwWrap	XwSoftWrap
XtNwrapBreak	XtCWrapBreak	XwWrapBreak	XwWrapWhiteSpace

XtNbottomMargin

The number of pixels used for the bottom margin. Default is two or the highlight thickness, whichever is greater.

XtNdisplayPosition

The position in the text source that will be displayed at the top of the screen. The default is 0, or the start of the text source.

XtNexecute

This callback list is similar to a selection function on a button. When the user invokes an event that calls the "execute" function (see the translation table below), this callback list will be executed. In the default translation table, this is bound to the "enter" key.

XtNextendKey

This resource redefines the meta modifier key (**NOTE:** See your system administrator for the identity of the meta modifier key on your keyboard). The actions that use the meta modifier key (see the section entitled "DEFAULT KEY BINDINGS FOR TEXTEDIT" in this man page) can be redefined so that the "ESC" key becomes the meta modifier key. This is done by setting XtNextendKey to XwEsc. This is useful when the application intends to use the extended character set. The only other setting for this resource is XwMeta, the default setting.

XtNgrow

This resource controls if the widget will try to resize its window when it needs more height or width to display the text. When set to XwGrowOff it will not resize itself. When set to XwGrowHorizontal it will attempt to change its width when lines are too long for the current screen width. When set to XwGrowVertical it will attempt to resize its height when the number of text lines is greater than can be displayed with the current screen height. When set to XwGrowBoth, the widget will attempt resizes in both dimensions. Growth attempts have higher priority than either wrapping or scrolling. If enabled, the widget will always try to grow to display text before trying to wrap or scroll. The default is XwGrowOff. The success of a resize request is determined by the widget's parent.

XtNinsertPosition

The position in the text source of the insert cursor. The default is 0.

XtNleaveVerification

This verification callback list is called before the widget loses input focus. The default is

NULL. See the verification section below.

XtNleftMargin
The number of pixels used for the left margin.

NOTE: If TextEdit is embedded in a manager with keyboard traversal enabled, it will silently enforce the constraint that left and right margins must be at least 3 pixels wider than the highlight border width.

XtNmodifyVerification
This verification callback list is called before text is deleted from or inserted to the text source. The default is NULL. See the verification section below.

XtNmotionVerification
This verification callback list is called before the insertion cursor is moved to a new position. The default is NULL. See the verification section below.

XtNrightMargin
The number of pixels used for the right margin.

XtNscroll
This resource controls the horizontal and vertical scrolling of lines longer than the screen width. When set to XwAutoScrollOff the widget will not scroll. When set to XwAutoScrollVertical, the widget will scroll lines vertically. When set to XwAutoScrollHorizontal, the widget will scroll a single-line display horizontally. Horizontal scrolling is not currently supported for multi-line displays. Both horizontal and vertical scrolling can be set with XwAutoScrollBoth (again, subject to the single-line horizontal restriction). The default is XwAutoScrollVertical. XtNscroll has lower priority than XtNwrap, meaning if wrapping is enabled, the widget will attempt to wrap to the next line before it will attempt to scroll horizontally.

XtNselection
This resource specifies a pointer to a structure that contains the beginning and ending position of the initial selection, as well as the selection type. The possible values for this resource are XwselectPosition, XwselectWord, XwselectLine, XwselectParagraph, XwselectAll, and XwselectNull. XwselectNull is the default value for this resource.

XtNsourceType
This resource describes the text source type. Valid argument list settings are XwstringSrc, XwdiskSrc, and XwprogramDefinedSrc. to set this resource in a resource defaults file (such as .Xdefaults) use the strings "stringsrc", "disksrc", and "programdefinedsrc".

XtNtextSource
This resource specifies the new source. When XtNsourceType is of type XwprogramDefinedSrc, TextEdit uses the value in this resource to define the text source. The default value is NULL.

XtNtopMargin
The number of pixels used for the top margin.

XtNwrap
This resource specifies how the widget displays lines longer than the screen width. When set to XwWrapOff, the lines may extend off screen to the right. When set to XwSoftWrap, the lines will be wrapped at the right margin with the actual position determined by the resource XtNwrapBreak.

XtNwrapBreak
This resource specifies how the wrap position is determined. When set to XwWrapAny, the wrap will happen at the character position closest to the right margin. When set to XwWrapWhiteSpace, the wrap will happen at the last whitespace before the right margin. If the line does not have whitespace, it will be wrapped as XwWrapAny.

SUBCOMPONENT RESOURCES

TextEdit defines three groups of subcomponent resources: StringSrc, DiskSrc, and Display. These groups are described below.

StringSrc

StringSrc defines the following new resources. They can be specified in a resource file by the name "stringsrc" under the name of the TextEdit widget, or through the class "StringSrc."

StringSrc Resource Set			
Name	**Class**	**Type**	**Default**
XtNeditType	XtCEditType	XwEditType	XwtextEdit
XtNmaximumSize	XtCLength	int	NULL
XtNstring	XtCString	char *	NULL

XtNeditType.
This resource controls the edit state of the source. It can be XwtextRead, a read only source, XwtextAppend, a source that can only be appended to, and XwtextEdit, a fully editable source.

XtNmaximumSize.
The maximum number of characters that can be entered into the internal buffer. If this value is not set then the internal buffer will increase its size as needed limited only by the space limitations of the process.

XtNstring.
The initial string to be viewed and/or edited. The default is the empty string. An XtGetValues call on this resource will return a copy of the internal buffer. The application program is responsible for freeing the space allocated by the copy. An XtSetValues call will copy the given string into the internal buffer.

DiskSrc

DiskSrc defines the following new resources. They can be specified in a resource file by the name "disksrc" under the name of the TextEdit widget, or through the class "DiskSrc."

DiskSrc Resource Set			
Name	**Class**	**Type**	**Default**
XtNeditType	XtCEditType	XwEditType	XwtextRead
XtNfile	XtCFile	char *	NULL

XtNeditType.
This resource controls the edit state of the source. It can be XwtextRead, a read only source, and XwtextAppend, a source tha can only be appended to.

XtNfile.
The absolute pathname of a disk file to be viewed and/or appended to. When opening for an append, if no file is given a temporary file will be created.

Display

Display defines the following new resources. They can be specified in a resource file by the name "display" under the name of the TextEdit widget, or through the class "Display."

Display resource Set			
Name	**Class**	**Type**	**Default**
XtNfont	XtCFont	XFontStruct *	Fixed
XtNforeground	XtCForeground	XtRPixel	Black

XtNfont.

The font used to display the text. The default is fixed. There are currently several display bugs associated with proportional fonts.

XtNforeground.
The color for drawing the text. The default is black.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the

scrolled window widget.

TRANSLATIONS

Since TextEdit has full editing functionality, it supports an elaborate set of translations. The following table lists TextEdit's default translations, which are a subset of key bindings from an EMACS editor. (An EMACS editor refers to a set of editors based on the original design of R.M. Stallman at MIT for an extensible, customizable, self-documenting display editor.) TextEdit supports the concept of delete and kill. Both delete and kill remove a unit of text from the text source, but text that has been removed with a kill can be restored by an unkill action. Kills are stored in the X Cutbuffer 1, so that a kill in one instance of a TextEdit widget can be inserted into another instance of a TextEdit widget. TextEdit does not support a history of kills in a kill ring, nor the appending of kills made in sequence. TextEdit highlights the current selection by reversing the foreground and background color. Text that has been copied from TextEdit into the current selection storage can be inserted into the buffer with a stuff action.

Each of these functions can be rebound to a different key in the default translation file set in .Xdefaults. The string to identify the function is identical to the function name used below. For example, to bind Control-I to move the insertion point forward one word is:

Ctrl<Key>I: forward-word

See *Programming With the HP X Widgets and the Xt Intrinsics* for more information on the Xdefaults file and translations.

TextEdit works with keyboard traversal and defines the required actions.

DEFAULT KEY BINDINGS FOR TEXTEDIT

Movement	
Ctrl F	forward-character
Right Arrow	forward-character
Ctrl B	backward-character
Left Arrow	backward-character
Meta F	forward-word
Meta B	backward-word
Meta]	forward-paragraph
Ctrl [	backward-paragraph
Ctrl A	beginning-of-line
Ctrl E	end-of-line
Ctrl N	next-line
Down Arrow	next-line
Ctrl P	previous-line
Up Arrow	previous-line
Ctrl V	next-page
Next	next-page
Meta V	previous-page
Prev	previous-page
Home	beginning-of-file
Shift Home	end-of-file
Ctrl Z	scroll-one-line-up
Meta Z	scroll-one-line-down

Delete, Kill, and Stuff	
Ctrl D	delete-next-character
Ctrl H	delete-previous-character
Meta D:	delete-next-word
Meta H	delete-previous-word
Shift Meta D	kill-word
Shift Meta H	backward-kill-word
Ctrl W	kill-selection
Ctrl K	kill-to-end-of-line
Meta K	kill-to-end-of-paragraph
Ctrl Y	unkill
Meta Y	stuff

Miscellaneous	
Ctrl O	newline-and-backup
Ctrl M	newline
<Btn1Down>	select-start
Button1<PtrMoved>	extend-adjust
<Btn1Up>	extend-end
<Btn2Down>	stuff
<Btn3Down>	extend-start
Button3<PtrMoved>	extend-adjust
<Btn3Up>	extend-end
Ctrl L	redraw-display
<Key>	insert-char

KEYBOARD TRAVERSAL

The following table summarizes the keystrokes which (when keyboard traversal is active) will move the focus. The keys used elsewhere in the X Widgets library for keyboard traversal are used for other purposes in the text edit widget. Therefore, it was necessary to define other keystrokes to serve these functions. To minimize the incompatibility the decision was made to use the same keys with the addition of the Ctl modifier to implement keyboard traversal in this widget.

Keyboard Traversal	
Ctrl Down	traverse-down
Ctrl Home	traverse-home
Ctrl Left	traverse-left
Ctrl Next	traverse-next
Ctrl Prev	traverse-prev
Ctrl Right	traverse-right
Ctrl Up	traverse-up

traverse-down:

Inform the parent of this widget that it should transfer keyboard focus to the first widget below this one.

traverse-home:

Inform the parent of this widget that it should transfer keyboard focus to the child which is closest to the uppper left hand corner of the parent. If that child already has the keyboard focus, then ask the grandparent of this widget to give the keyboard focus to whichever of its children which is closest to the upper left hand corner.

traverse-left:

Inform the parent of this widget that it should transfer keyboard focus to the first widget to the left of this one.

traverse-next:
Inform the parent of this widget that it should transfer keyboard focus to the next child in the parent's list of children.

traverse-prev:
Inform the parent of this widget that it should transfer keyboard focus to the previous child in the parent's list of children.

traverse-right:
Inform the parent of this widget that it should transfer keyboard focus to the first widget to the right of this one.

traverse-up:
Inform the parent of this widget that it should transfer keyboard focus to the first widget above this one.

DISPLAYING TEXT, WORD WRAP AND ACTIONS

Text is considered to be hierarchically composed of white space, words, lines and paragraphs. These component concepts are currently hard-coded, but we intend that future versions will support a more general version of the text composition hierarchy. White space is defined as any non-empty sequence of the ASCII characters space, tab, linefeed or carriage return (decimal values of 32, 9, 10, 13, respectively); a word is any non-empty sequence of characters bounded on both sides by whitespace. A source line is any (possibly empty) sequence of characters bounded by newline characters; a display line is any (possibly empty) sequence of characters appearing on a single screen display line. A source paragraph is any sequence of characters bounded by sets of two or more adjacent newline characters. a display paragraph is any (possibly empty) sequence of characters bounded by newline characters (NOTE: this is identical to the definition of a source line). In all cases, the beginning or end of the edit text is an acceptable bounding element in the previous definitions.

When making display decisions, TextEdit first determines whether all the text will fit in the current display. If it does not, and growing is enabled, the widget will make resize request of its parent. If the request is denied or only partially satisfied, no future growth requests will be made unless there is an intervening resize operation externally imposed. If any source line is still too long to fit in the display after growing is attempted, wrapping is checked. If wrap is off (XwWrapOff), one display line is drawn for each source line. If a source line is too long for the display, it is truncated at the right margin after the last full character which fits. If wrapping is enabled (XwSoftWrap), a new display line will be started with the first word which doesn't fit on the current line. If the wrap break option is XwWrapAny, as many characters from that word as will fit before the right margin are written to the current display line, then the next character starts at the left margin of the next display line. If the wrap break option is XwWrapWhiteSpace, the line break is instead made after the first whitespace character which follows the last full word which does fit on the current display line. If, however, under white space break, the first full word which does not fit is also the first word on the line, the wrap break is made as if XwWrapAny were selected.

VERIFICATION CALLBACKS

Three types of verification callbacks are supported by TextEdit There is one for motion operations, to verify a new insert position; there is one for modifying operations, to verify insertion, deletion or replacement of text; there is one for widget exit, to verify state consistency on loss of focus by the widget. Each verification callback procedure is of type XtCallbackProc, which defines the three arguments it will be invoked with. These are the id of the widget making the callback, the client data which was specified by the client application when the callback was registered (see XtAddCallback), and a pointer (type XwTextVerifyPtr) to the verification call_data structure. The C data types used here are:

```
typedef enum {motionVerify, modVerify, leaveVerify} XwVerifyopType ;

typedef struct {
    XEvent      *xevent ;
    XwVerifyopType  operation ;
```

```
boolean      doit ;
XtTextPosition  currInsert, newInsert ;
XtTextPosition  startPos, endPos ;
XtTextBlock   *text ;
} XwTextVerifyCD, *XwTextVerifyPtr ;
```

Before the chain of verification callbacks is activated for any given operation, a structure of type XtTextVerifyCD is initialized. The initial values are:

xevent:	for a leave operation, the current event pointer
operation:	element of opType signifying the type of verification operation
doit:	TRUE
currInsert:	current position of the insert point
newInsert:	for a motion operation, the position the user is attempting to move the insert point to, otherwise, the same value as currInsert
startPos:	for a modify operation, the beginning position in the current source of the text about to be deleted or replaced, or where new text will be inserted. If not a modify operation, the same value as currInsert.
endPos:	for a modify operation, the ending position in the current source of the text about to be deleted or replaced. If no text is being removed, it will have the same value as startPos. If not a modify operation, the same value as currInsert.
text:	for a modify operation with new text to be inserted, a pointer to a structure of type XtTextBlock, which references the text to be inserted. Otherwise, NULL.

It is possible for the client to register more than one callback procedure for any of these callback types. The order in which the callbacks will be invoked is described in the toolkit documentation. Since there can be more than one callback, each verification procedure should first check the "doit" field. If it is false, someone else has already rejected the operation, so there is no need for further evaluation. On return from invoking the chain of callbacks, the TextEdit widget will look at the "doit" member of the XtTextVerifyCD structure. If it is still true, TextEdit will proceed with operation, otherwise it will not. Any user feedback for the rejected operation is the responsibility of the verification procedure. Verification callbacks are permitted to modify some of the data in the XtTextVerifyCD structure. The TextEdit widget will only look at certain fields on return, though, according to the operation type. For a motion operation, only the newInsert position will be looked at. For a modify operation, only startPos, endPos and text will be examined for changes. For leave operation, no fields will be examined. There is no mechanism for preventing a verification callback from making other changes to the editing state through the documented interface, but such behind-the-back actions are discouraged.

APPLICATION WRITER'S INTERFACE

The state of TextEdit can be changed in through the normal functional interface to widgets (XtSetValues and XtGetValues) or by exported external functions.

TextEdit's resources can be queried and set through XtSetValues and XtGetValues. The widget will maintain its display consistent with the new values. In particular, this is the method for changing the display options.

The internal buffer should be manipulated through the external functions that follow.

This set of external functions is designed to allow the widget programmer to access the internal buffer that TextEdit manages. For example, if the widget is being used to enter a string, the program can get a copy of the string (i.e. the internal buffer) with the function XwTextCopyBuffer or XwTextReadSubString. All of the following functions that change the contents of the buffer, its selection, or insertion position, will update the display after they are called. If the programmer needs to make a sequence of these calls, the widget's screen updating function should be turned off with a call to XwTextUpdate(Off) to prevent screen flash. After the sequence of calls the programmer must remember to call XwTextUpdate(On) to update the window and resume normal updating. Note that it is not necessary to turn off the update function for functions that only get values from the widget. Neither is it necessary to use these calls if the programmer only makes one call that changes the widget.

Buffer Functions

```
void XwTextClearBuffer(w)
        XwTextEditWidget w;
```

Clear the internal buffer. After this call all characters in the buffer have been removed.

```
void *XwTextCopyBuffer(w)
        XwTextEditWidget w;
```

This function uses XtMalloc to create space to make a copy of the internal buffer and returns the pointer to that copy. The application writer is responsible for freeing the space.

Read a Substring

```
int XwTextReadSubString( w, startpos, endpos, target, targetsize, targetused )
 XwTextEditWidget  w;
 XwTextPosition    startpos, endpos;
 unsigned char     *target;
 int               targetsize,
                   *targetused;
```

This function will move characters from the buffer into the caller's space. The caller must provide the space to copy into and its size in bytes. The routine will return the number of positions moved. The value of targetused returns the number of bytes used in the target string by the move.

Selection

```
void *XwTextCopySelection(w)
  XwTextEditWidget w;
```

This function uses XtMalloc to create space to make a copy of the current selection and returns the pointer to that copy. The application writer is responsible for freeing the space.

```
void XwTextUnsetSelection(w)
   XwTextEditWidget w;
```

This function will clear the current selection.

```
void XwTextSetSelection(w, left, right)
 XwTextEditWidget  w;
 XwTextPosition    left, right;
```

This function sets the current selection to be between the character positions left to right.

```
void XwTextGetSelectionPos(w, left, right)
    XwTextEditWidget w;
    XwTextPosition *left, *right;
```

This function returns the character positions of the current selection.

```
void XwTextSetSelectionArray()
                    XwTextEditWidget w;
                    XwSelectType *sarray;
```

This function resets the selection types for multiple mouse button clicks. The array must be terminated with XwselectNull. The possible types are XwselectPosition, XwselectWord, XwselectLine, XwselectParagraph, XwselectAll, and XwselectNull.

Insertion and Deletion

```
void XwTextInsert(w, string)
    XwTextEditWidget w;
    unsigned char *string;
```

This function inserts the string at the current insertion position and advances the insertion position to the end of the string.

```
void XwTextReplace(w, startPos, endPos, text)
  XwTextEditWidget      w;
  XwTextPosition startPos,
            endPos;
  unsigned char  *text;
```

Remove text in the source from startPos to endPos and insert the string text starting at startPos. If startPos and endPos are the same the action is an insertion. If text is the empty string, the action is a deletion.

Drawing and Updating

```
void XwTextRedraw(w);
  XwTextEditWidget w;
```

Refresh the widget screen.

```
void XwTextUpdate( w, status )
    XwTextEditWidget w;
    Boolean status;
```

This function turns the widget's screen updating function on and off. Wrapping these calls around a sequence of calls that change the content of the internal buffer will prevent screen flash.

End of Buffer

```
XwTextPosition XwTextGetLastPos (w, lastPos)
    XwTextEditWidget w;
```

This function returns the last character position in the buffer.

Insertion Position

```
void XwTextSetInsertPos(w, position)
    XwTextEditWidget w;
    XwTextPosition position;

XwTextPosition XwTextGetInsertPos(w)
    XwTextEditWidget w;
```

These functions set and return the insertion position.

Setting the Source

```
void XwTextSetSource(w, source, startpos)
    XwTextEditWidget w;
    XwTextSourcePtr source;
    XwTextPosition startpos;
```

SOURCE DEFINITION

The source provides textual data space and functions for manipulating that data. The functions are defined below. An application can define its own source by reimplementing these functions.

Read

```
XwTextPosition DiskReadText(src, pos, text, maxread)
    XwTextsource   *src;
    XwTextPosition  pos;
    XwTextblock    *text;
    XwTextPosition  maxread;
or
XwTextPosition SourceReadText(src, pos, text, maxread)
    XwTextsource   *src;
    XwTextPosition  pos;
    XwTextblock    *text;
    XwTextPosition  maxread;
```

This function returns a read-only text block in the src with maxread number of characters starting from pos. The return value is the next character position following the block.

Replace

```
XwEditResult DiskReplaceText(src, startpos, endpos, textblk, delta)
    XwTextsource   *src;
    XwTextPosition  startpos,
                    endpos;
    XwTextBlock    *textblk;
    XwTextPosition          *delta;
or
XwEditResult StringReplaceText(src, startpos, endpos, textblk, delta)
    XwTextsource   *src;
    XwTextPosition  startpos,
                    endpos;
    XwTextBlock    *textblk;
    XwTextPosition          *delta;
```

This function removes existing text in src between startpos and endpos and inserts new text from textblk at startpos. delta is change in the size of the text source. It returns XweditDone for a successful operation, XweditPosError for positional errors when source is in XttextAppend mode, and XweditError when the operation could not be performed.

SetLastPosition

```
void DiskSetLastPos(src, lastpos)
  XwTextSource   *src;
  XwTextPosition lastpos;
or
void StringSetLastPos(src, lastpos)
  XwTextSource   *src;
  XwTextPosition lastpos;
```

This functions sets the last position in the source.

Scan

```
XwTextPosition DiskScan(src, pos, scantype, dir, count, include)
  XwTextsource   *src;
  XwTextPosition  pos;
  XwScanType      scantype;
  XwScanDirection dir;
  int             count;
  Boolean         include;
or
XwTextPosition StringScan(src, pos, scantype, dir, count, include)
  XwTextsource   *src;
  XwTextPosition  pos;
  XwScanType      scantype;
```

```
XwScanDirection  dir;
int              count;
Boolean          include;
```

SourceScan searches in dir direction (XwsdLeft XwsdRight) for XwScantype (XwstPositions, XwstWhiteSpace, XwstEOL, XwstParagraph, XwstLast). The variable "count" is the number of the given type it will scan over. The variable "include" indicates whether to count the item currently being pointed to. It returns the starting position of the item scanned for.

EditType

```
XtEdittype DiskGetEditType(src)
  XwTextsource  *src;
or
XtEdittype StringGetEditType(src)
  XwTextsource  *src;
```

Returns the edit type of source.

NATIONAL LANGUAGE I/O SUPPORT

TextEdit supports 16-bit National Language I/O (NLIO) if you have the NLIO subsystem package installed. In order to see and edit 16-bit characters, an appropriate font must be loaded and 16-bit text must be encoded in "HP-15." The language to be used is determined by the environment variable LANG or the physical keyboard in use. If LANG is set to a valid language, that language is used. If it is not set or is set to an invalid value, then TextEdit will determine the language of the physical keyboard in use and use that language. You should set the resource XtNextendKey to the value XwEsc when using 16-bit NLIO. Note that 16-bit NLIO is currently supported when using StringSrc but not when using DiskSrc.

CURRENT LIMITATIONS

The current default source is not optimized for large amounts of data. X11's current selection is not yet supported.

ORIGIN

Digital Equipment Corporation. Massachusetts Institute of Technology. Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwtitleBarWidgetClass - An X Widget for creating titlebars.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/TitleBar.h>
```

CLASSES

This class is built from the Core, Composite, Constraint and Manager classes.

The widget class to use when creating a TitleBar widget is **XwtitleBarWidgetClass**.

The class name of TitleBar is **TitleBar**.

DESCRIPTION

TitleBar provides a flexible mechanism for creating titlebars containing text and arbitrary widgets. Inputs are an optional text string and any number of widgets to manage. The title string will be displayed in a StaticText widget (refer to XWSTATICTEXT(3X)). Managed widgets may have optionally specified layout information (see CONSTRAINT RESOURCES below).

When TitleBar is directed to become narrower than is necessary to display all of its interior widgets, some widgets may be hidden. The XtNprecedence resource in each managed widget controls this feature.

As TitleBar is directed to become narrower and narrower, widgets whose sum of XtNrPadding and XtNlPadding is greater than zero will have their padding collapsed to one pixel. Widgets will have their padding stripped in order of decreasing values of XtNprecedence.

If, after collapsing all of the widgets' padding, TitleBar is still too narrow to display all of its children widgets, widgets will be hidden. Widgets will be hidden in order of decreasing values of XtNprecedence. TitleBar will try to always display a widget of the highest priority (lowest value of XtNprecedence, even if it must be clipped.

Users of TitleBar should note that when children widgets are hidden they are completely hidden. Additionally, users who wish to make extensive use of the obscurability rules should read carefully the section on XtNprececence in the CONSTRAINT RESOURCES section below.

NEW RESOURCES

To specify any of these resources within a resource defaults file, simply drop the XtN prefix from the resource name. TitleBar defines the following new resources:

TitleBar Resource Set			
Name	**Class**	**Type**	**Default**
XtNenter	XtCCallback	Pointer	NULL
XtNhSpace	XtCHSpace	int	2
XtNleave	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNtitleBackground	XtCBackground	Pixel	white
XtNtitleBorderWidth	XtCBorderWidth	int	0
XtNtitleForeground	XtCForeground	Pixel	black
XtNtitleHSpace	XtCHSpace	int	2
XtNtitleLPadding	XtCTitleLPadding	int	1
XtNtitlePosition	XtCTitlePosition	int	0
XtNtitlePrecedence	XtCTitlePrecedence	int	0
XtNtitleRegion	XtCTitleRegion	XwAlignment	XwALIGN_CENTER
XtNtitleRelease	XtCCallback	Pointer	NULL
XtNtitleRPadding	XtCTitleRPadding	int	1
XtNtitleSelect	XtCCallback	Pointer	NULL
XtNtitleVSpace	XtCVSpace	int	2
XtNvSpace	XtCVSpace	int	2

XtNenter, XtNleave, XtNselect, and XtNrelease
Callbacks provided for control of TitleBar. The data parameter is unused.

XtNhSpace
The amount of space to maintain between the right and left of the titlebar and the interior widgets.

XtNtitleBorderWidth
The value to loaded into the XtNborderWidth resource of the optional StaticText widget.

XtNtitleBackground
The value to be loaded into the XtNbackground resource of the optional StaticText widget's core part.

XtNtitleForeground
The value to be loaded into the XtNforeground resource of the optional StaticText widget's core part.

XtNtitleHSpace
The value to be loaded into the XtNhSpace resource of the optional StaticText widget.

XtNtitleLPadding
The value to be loaded into the XtNlPadding constraint resource of the optional StaticText widget.

XtNtitlePosition
The value to be loaded into the XtNposition constraint resource of the optional StaticText widget.

XtNtitlePrecedence
The value to be loaded into the constraint record of the optional StaticText widget.

XtNtitleRegion
The value to be loaded into the XtNregion constraint resource of the optional StaticText widget.

XWTITLEBAR(3X)

XtNtitleRelease
The value loaded into the XtNrelease resource of the optional StaticText widget.

XtNtitleRPadding
The value to be loaded into the XtNrPadding constraint resource of the optional StaticText widget.

XtNtitleSelect
The value loaded into the XtNselect resource of the optional StaticText widget.

XtNtitleVSpace
The value to be loaded into the XtNvSpace resource of the optional StaticText widget.

XtNvSpace
The amount of space to maintain between the top and bottom of the titlebar and the interior widgets.

INCORPORATED RESOURCES

The TitleBar creates an internal StaticText widget to handle the title string. In order to provide the user some control over the appearance of this internal widget, the following resources defined by StaticText are incorporated into TitleBar's resource list.

It must be noted that only the resources within the following tables will have any effect on the internal StaticText widget. The other resources defined for StaticText will be overridden by TitleBar.

For a complete description of the following resources, refer to the manual page given in the table heading.

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	XwPATTERN_BORDER
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	XwBACKGROUND
XtNtraversalType	XtCTraversalType	int	HIGHLIGHT_OFF

Static Text Resource Set -- STATICTEXT(3X)			
Name	**Class**	**Type**	**Default**
XtNalignment	XtCAlignment	XwAlignment	XwALIGN_CENTER
XtNfont	XtCFont	XFontStruct *	Fixed
XtNlineSpace	XtCLineSpace	int	0
XtNstring	XtCString	char *	NULL
XtNwrap	XwCWrap	Boolean	TRUE

CONSTRAINT RESOURCES

The following resources will be attached to every widget inserted into TitleBar. See CONSTRAINT(3X) for a general discussion of constraint resources.

TitleBar uses the constraint resources as hints during the layout of its managed children. Under certain conditions, any of these resources except XtNprecedence can (and will) be ignored by TitleBar.

Constraint Resource Set -- Children of TITLEBAR(3X)			
Name	**Class**	**Type**	**Default**
XtNlPadding	XtCLPadding	int	2
XtNposition	XtCPosition	int	0
XtNprecedence	XtCPrecedence	int	1
XtNregion	XtCRegion	XwAlignment	See below.
XtNrPadding	XtCRPadding	int	2

XtNlPadding

The number of pixels that TitleBar should try to maintain between the left of the widget and the right padding of the sibling widget to the left. For example, widget1 is to the left of widget2 within TitleBar. Widget1 has a XtNrPadding value of 5. Widget2 has a XtNlPadding value of 5. The borders of widget1 and widget2 will be 10 pixels apart.

If TitleBar is too narrow to honor all of its children's padding requests without hiding some children, some, possibly all, padding requests will be collapsed.

XtNposition

This resource gives the order of widgets within region. The left and the center region are laid out with XtNposition values increasing from left to right. The right region is laid out with XtNposition values increasing from right to left.

Position values are unique within a region. If two widgets are assigned the same position, the widget which was assigned first gets the position. The second widget gets the next available position. For example, widget1 and widget2 are the only widgets inserted in TitleBar. Widget1 is inserted before widget2. Widget1 and widget2 are both assigned a position of 4. Widget1 will be given the position of 4, and widget2 will be assigned a position of 5.

XtNprecedence

When TitleBar is too narrow to display all of its children, this resource is used to determine which children should be hidden. Widgets with high values of XtNprecedence are hidden first. Precedence values are relative to all other widgets within an instantiation of TitleBar. This means that all widgets, regardless of their region, with high values of XtNprecedence will be hidden before any widgets with the next lower values are hidden.

Values of XtNprecedence need not be unique. If values are unique, there is no question about which widget is first to lose its padding, nor about which widget is first to be hidden.

If values are not unique for all children of TitleBar, there need be no question about which widget is acted on first, but it is dependent on both insertion order and precedence. The last widget inserted in TitleBar of a given precedence is the first to lose its requested padding (of widgets with that priority). Widgets lose padding from last inserted to first inserted, within a given level of precedence. When hiding widgets, widgets within a given precedence level are hidden from last inserted to first inserted.

XtNregion

Associates a child with a region of the titlebar. The regions may be specified in the resource default file as "left" for XwALIGN_LEFT, "center" for XwALIGN_CENTER, and "right" for XwALIGN_RIGHT.

During layout widgets with XtNregion values of XwALIGN_LEFT are grouped to the left end of TitleBar. Widgets with XtNregion values of XwALIGN_LEFT are grouped to the right of TitleBar. Widgets with XtNregion values of XwALIGN_CENTER will be grouped between the left and right groups. Additionally, TitleBar tries to center the center group within the TitleBar.

Widgets for which XtNregion is unspecified or XwALIGN_NONE when XtNstring is non-null, will be assigned one of the two regions not equal to XtNtitleRegion in an alternating fashion.

XWTITLEBAR(3X)

Widgets for which XtNregion is unspecified or XwALIGN_NONE when XtNstring is null, will be assigned a region. The first such widget will be assigned to the left region, the next to the center region, the next to the right region, the next to the left region, and so forth.

XtNrPadding

The number of pixels that TitleBar should try to maintain between the right of the widget and the left padding of the sibling widget to the right. For example, widget1 is to the right of widget2 within TitleBar. Widget1 has a XtNlPadding value of 5. Widget2 has a XtNrPadding value of 5. The borders of widget1 and widget2 will be 10 pixels apart.

If TitleBar is too narrow to honor all of its children's padding requests without hiding some children, some, possibly all, padding requests will be collapsed.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNforeground	XtCForeground	Pixel	Black
XtNlayout	XtClayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

TRANSLATIONS

The input to the toggle is driven by the mouse buttons. The default translation set defining this button is listed below.

```
<EnterWindow>:    enter()"
<LeaveWindow>:    leave()"
<Btn1Down>:       select()"
<Btn1Up>:         release()"
```

ACTIONS

enter If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE) and the parent of this widget is not a subclass of XwManager, initiate keyboard traversal. After this, the callback list is invoked.

leave If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE) and the parent of this widget is not a subclass of XwManager, terminate keyboard traversal. After this, the callback list is invoked.

release Invokes the release callback list.

select Invokes the select callback list.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X), XWSTATICTEXT(3X), XWCREATETILE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

XWTOGGLE(3X)

NAME

XwtoggleWidgetClass - the X Widgets toggle button widget

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Toggle.h>

CLASSES

The toggle widget is built from the Core, Primitive and Button classes.

The widget class to use when creating a toggle is **XwtoggleWidgetClass**. The class name is **Toggle**.

DESCRIPTION

The toggle widget implements a button which consists of a graphic and a label. The label can be positioned either to the right (the default) or the left of the graphic. The size of the graphic is based on the height of the font used for the label. The space between the graphic and the label is equal to 1/3 the font height. The default graphic is a square box and this may be changed to a diamond shape. It is intended that application writers can put a group of square buttons into a Row Column manager with its mode set to the default n_of_many to get the checkbox, or N of Many, selection semantic and then put a group of diamond buttons into a Row Column manager with its mode set to one_of_many to get the radiobutton, or One of Many, selection semantic.

The default semantic for this button is that button 1 down will toggle the state of the toggle. When in a selected state, the interior of the graphic will be filled with the foreground color; when not selected the interior of the graphic will be filled with the background color; when insensitive, the label will be drawn with the patterned tile (the default is a 75/25 mix of the foreground and background colors).

Callbacks can be attached to the widget to report selection (XtNselect) and unselection (XtNrelease). This widget can be set to respond to Enter and Leave window events by highlighting and unhighlighting the button. This widget is also capable of handling keyboard traversal. See the translations below for the default traversal keycodes.

NEW RESOURCES

The toggle widget class defines a set of resource types that can be used by the programmer to specify data for widgets of this class. Recall that the string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNfont becomes font.

Toggle Resource Set			
Name	**Class**	**Type**	**Default**
XtNselectColor	XtCForeground	Pixel	Black
XtNsquare	XtCSquare	Boolean	True

XtNselectColor

Allows the application to specify what color should be used to fill in the center of the square (or the diamond) when it is set.

XtNsquare

If True, forces the button to draw a square box, otherwise it will draw a diamond shape box. One possible usage for this resource is to make the convention that row column managers containing diamond shaped toggles have their XtNmode resource set to one_of_many which will only allow one of the buttons to be set at any one time, while row column managers containing square buttons use the default mode setting of n_of_many which allows any or all of the buttons to be set.

INHERITED RESOURCES

The following resources are inherited from the named superclasses: The defaults used for the toggle when being created are as follows:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

Button Resource Set -- XWBUTTON(3X)			
Name	Class	Type	Default
XtNfont	XtCFont	XFontStruct *	Fixed
XtNhSpace	XtCHSpace	int	2
XtNlabel	XtCLabel	caddr_t	NULL
XtNlabelLocation	XtCLabelLocation	int	right
XtNsensitiveTile	XtCSensitiveTile	int	75_foreground
XtNset	XtCSet	Boolean	False
XtNvSpace	XtCVSpace	int	2

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer

XWTOGGLE(3X)

to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the toggle widget.

TRANSLATIONS

The input to the toggle is driven by the mouse buttons. The default translation set defining this button is listed below. Note that for the specific key symbols used in traversal, the HP Key Cap which corresponds to this key symbol appears to the right of the definition.

```
<Btn1Down>:       toggle()
<EnterWindow>:    enter()
<LeaveWindow>:    leave()
<Key>Select:      toggle()    HP "Select" key
```

ACTIONS

Note that this widget contains some actions which are not bound to any events by the default translations. The purpose of these additional actions are to allow advanced users to alter the button semantics to their liking.

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the button will be highlighted. Otherwise no action is taken.

leave: If the XtNtraversalType resources has been set to XwHIGHLIGHT_ENTER then the button will be unhighlighted. Otherwise no action is taken.

toggle: Toggle the set state of the button (make it TRUE if it was FALSE, FALSE if it was TRUE). Redraw only the toggle part (not the label) of the button. If the current state of the button is set (TRUE) issue the XtNselect callbacks, if not set (FALSE) issue the XtNrelease callbacks. No additional data beyond the widget id and the specified closure is sent with these callbacks.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWBUTTON(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwvaluatorWidgetClass - the X Widget's valuator widget

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Valuator.h>

CLASSES

The Valuator widget is built from the Core and Primitive classes.

The widget class to use when creating a valuator is **XwvaluatorWidgetClass**. The class name for Valuator is **Valuator**.

DESCRIPTION

The Valuator widget implements a horizontal or vertical scrolling widget as a rectangular bar containing a sliding box (slider). The Valuator widget supports input through interactive slider movement and selections on the slide area not occupied by the slider. Both types of input have a separate callback list for communicating with the application. The Valuator widget can be used by the application to attach to objects scrolled under application control, or used by composite widgets to implement predefined scrolled objects.

NEW RESOURCES

The Valuator widget defines a set of resource types used by the programmer to specify the data for the valuator. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by Valuator.

Valuator Resource Set			
Name	**Class**	**Type**	**Default**
XtNareaSelected	XtCCallback	Pointer	NULL
XtNsliderExtent	XtCSliderExtent	int	10
XtNsliderMax	XtCSliderMax	int	100
XtNsliderMin	XtCSliderMin	int	0
XtNsliderMoved	XtCCallback	Pointer	NULL
XtNslideOrientation	XtCSlideOrientation	int	vertical
XtNsliderOrigin	XtCSliderOrigin	int	0
XtNsliderReleased	XtCCallback	Pointer	NULL
XtNsliderShadowOn	XtCsliderShadowOn	Boolean	TRUE
XtNsliderTile	XtCSliderTile	int	foreground

XtNareaSelected, BXtNsliderMoved, XtNsliderReleased

The Valuator widget defines three types of callback lists that are invoked upon different event conditions when interacting with a valuator. All types use the data parameter to send the location of the slider to the callback functions.

The first callback type, sliderMoved, defines the callback list containing the callback functions called when the slider is interactively moved.

The second callback type, sliderReleased, defines a callback list containing callback functions called when the mouse button is released after moving the slider.

The third callback type, areaSelected, defines a callback list containing the callback functions called when an area in a valuator not containing the slider is selected. The slider is not moved to this position but if the application wants the slider moved, it can use the position value contained in the parameter call_data and perform a XtSetValues() to its valuator.

For the callback types, the call_data parameter of the callback function will be an integer containing the slider or selection position.

XtNsliderExtent
The size of the slider can be set by the application. The acceptable values are 0 < sliderExtent < (sliderMax - sliderMin).

XtNsliderMax, XtNsliderMin
The Valuator widget lets the application define its own coordinate system for the valuator. Any integer values with sliderMin less than sliderMax can be specified.

XtNslideOrientation
The Valuator widget supports both horizontal and vertical scrolling. This resource type is the means by which this is set. It can be defined through the .Xdefaults file by the strings "horizontal", and "vertical" or within an arg list for use in XtSetValues() by the defines XwHORIZONTAL and XwVERTICAL.

XtNsliderOrigin
The location of the slider can be set by the application. The acceptable values are between sliderMin and (sliderMax - sliderExtent).

XtNsliderShadowOn
The valuator will automatically draw shadows around both the valuator and slider. This resource can be used to control drawing the shadow around the slider.

XtNsliderTile
This resource is used to set the tile used to create the pixmap to use when drawing the slider. The #defines for setting the values through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. See the XwPrimitive man page for a complete description of these translations. See the TRANSLATIONS section in this man page for a description of the translations local this widget.

TRANSLATIONS

The input to the Valuator widget is driven by the mouse buttons. The default translation is defined as follows:

```
<Btn1Down>:          select(),
<Btn1Up>:            release(),
Button1<PtrMoved>:   moved(),
<EnterWindow>:       enter(),
<LeaveWindow>:       leave(),
Ctrl<Key>Left:       left(),          HP "Control Left Cursor" key
Ctrl<Key>Up:         up(),            HP "Control Up Cursor" key
Ctrl<Key>Right:      right(),         HP "Control Right Cursor" key
Ctrl<Key>Down:       down(),          HP "Control Down Cursor" key
Ctrl<Key>Left:       keyrelease(),    HP "Control Left Cursor" key
Ctrl<Key>Up:         keyrelease(),    HP "Control Up Cursor" key
Ctrl<Key>Right:      keyrelease(),    HP "Control Right Cursor" key
Ctrl<Key>Down:       keyrelease(),    HP "Control Down Cursor" key
```

ACTIONS

down: If the valuator's orientation is vertical, this action will cause its slider origin to be incremented by 1 unit and redisplayed.

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be highlighted. Otherwise no action is taken.

keyrelease:
This action will cause the current slider position to be reported through the XtNSliderPosition callback.

leave: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be unhighlighted. Otherwise no action is taken.

left: If the valuator's orientation is horizontal, this action will cause its slider origin to be decremented by 1 unit and redisplayed.

moved: Moved processes interactive movement of the slider following a selection upon the slider.

release:
Release handles the processing terminating conditions for selections on the valuator.

right: If the valuator's orientation is horizontal, this action will cause its slider origin to be incremented by 1 unit and redisplayed.

select: Select processes the activation conditions within the valuator, both for selections within the slider area and on the slider.

up: If the valuator's orientation is vertical, this action will cause its slider origin to be decremented by 1 unit and redisplayed.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWCREATETILE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwvPanedWidgetClass - the X Widgets vertical paned manager widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/VPW.h>
```

CLASSES

The vertical paned manager widget is built from the Core, Composite, Constraint and Manager classes. Note that since the Composite class contains no user settable resources, there is no table for Composite class resources.

The widget class to use when creating a vertical paned manager is **XwvPanedWidgetClass.** The class name is **VPanedWindow.**

DESCRIPTION

The vertical paned manager is a composite widget which lays children out in a vertically tiled format. Children appear in a top to bottom fashion, with the first child inserted appearing at the top of the paned widget and the last child inserted appearing at the bottom of the paned widget. The vertical paned manager will grow to match the width of its widest child and all other children are forced to this width. The height of the vertical paned manager will be equal to the sum of the heights of all its children and the (optional) padding surrounding them.

It is also possible for the end user to adjust the size of the panes. To facilitate this adjustment, a control widget (XwsashWidgetClass) is created for most children. The control widget appears as a square box positioned on the bottom of the pane which it controls. Using the mouse (see the description on translations below) a user can adjust the size of a pane.

The vertical paned manager is 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 vertical paned widget will not allow a pane to be adjusted below its minimum nor beyond its maximum. Also, when the minimum size of a pane is equal to its maximum then no control widget will be presented for that pane. Nor will a control widget be presented for the bottom-most pane.

The vertical paned manager supports 2 presentation modes: framed and unframed. When framed, each pane is offset from the edges of the vertical paned manager and from other panes by a specified (and settable) number of pixels. In this mode the entire borderwidth of each child is also visible. Note that the vertical paned manager enforces a particular (and settable) border width on each pane. The second mode is unframed where the edge of a pane exactly corresponds to the edge of the vertical paned manager so that only a border between panes is visible.

No callbacks are defined for this manager.

NEW RESOURCES

The vertical paned manager defines a set of resource types used by the programmer to specify data for the manager widget. The programmer can also set the values for the Core and XwManager widget classes to set attributes for this widget. The following table contains the settable resources defined by the vertical paned manager. Recall that the string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNmin becomes min.

Vertical Paned Resource Set			
Name	**Class**	**Type**	**Default**
XtNborderFrame	XtCBorderWidth	int	1
XtNframed	XtCBoolean	Boolean	TRUE
XtNpadding	XtCPadding	int	3
XtNrefigureMode	XtCBoolean	Boolean	TRUE
XtNsashIndent	XtCSashIndent	int	-10

XtNborderFrame

The application can specify the thickness of the borderwidth of all panes in the paned manager. The value must be greater than or equal to 0.

XtNframed

The application can specify whether the panes should be displayed with some padding surrounding each pane (TRUE) or whether the panes should be set flush with the paned manager (FALSE).

XtNpadding

The application can specify how many pixels of padding should surround each pane when it is being displayed in framed mode. This value must be greater than or equal to 0.

XtNrefigureMode

This setting is useful if a large number of programmatic manipulations are taking place. It will prevent the manager from recomputing and displaying new positions for the child panes (FALSE). Once the changes have been executed this flag should be set to TRUE to allow the vertical paned manager to show the correct positions of the current children.

XtNsashIndent

This controls where along the bottom of the pane the control widget (the pane's sash) will be placed. A positive number will cause the sash to be offset from the left side of the pane, a negative number will cause the sash to be offset from the right side of the pane. If the offset specified is greater than the width of the vertical paned manager, minus the width of the sash, the sash will be placed flush against the left hand side of the paned manager.

CONSTRAINT RESOURCES

The following resources are attached to every widget inserted into vertical paned manager. See *CONSTRAINT(3X)* for a general discussion of constraint resources.

Constraint Resource Set -- Children of VPANEDWINDOW(3X)			
Name	**Class**	**Type**	**Default**
XtNallowResize	XtCBoolean	Boolean	FALSE
XtNmax	XtCMax	int	10000
XtNmin	XtCMin	int	1
XtNskipAdjust	XtCBoolean	Boolean	FALSE

XtNallowResize

Allows an application to specify whether the vertical paned manager should allow a pane to request to be resized. This flag only has an effect after the paned manager and its children have been realized. If this flag is set to TRUE, the manager will try to honor requests to alter the height of the pane. If false, it will always deny pane requests to resize.

XtNmax

Allows an application to specify the maximum size to which a pane may be resized. This value must be greater than the specified minimum.

XtNmin

Allows an application to specify the minimum size to which a pane may be resized. This value must be greater than 0.

XtNskipAdjust

Allows an application to specify that the vertical paned manager should not automatically resize this pane (flag set to TRUE).

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

XWVPANED(3X)

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNforeground	XtCForeground	Pixel	Black
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

SASH TRANSLATIONS

The translations which control the sashes created for each adjustable pane are replicated here for convenience.

```
<Btn1Down>:       SashAction(Start, UpperPane)
<Btn2Down>:       SashAction(Start, ThisBorderOnly)
<Btn3Down>:       SashAction(Start, LowerPane)
<Btn1Motion>:     SashAction(Move, Upper)
<Btn2Motion>:     SashAction(Move, ThisBorder)
<Btn3Motion>:     SashAction(Move, Lower)
Any<BtnUp>:       SashAction(Commit)
<EnterWindow>:    enter()
<LeaveWindow>:    leave()
```

enter: Enter window events occurring on the scrolled window are handled by this action.

leave: Leave window events occurring on the scrolled window are handled by this action.

SashAction(Start, UpperPane):

Change the cursor from the crosshair to an upward pointing arrow. Determine the upper pane which will be adjusted (usually the pane to which the sash is attached).

SashAction(Start, ThisBorderOnly):

Change the cursor from the crosshair to a double headed arrow. The panes that will be adjusted are the pane to which the sash is attached and the first pane below it that can be adjusted. Unlike the UpperPane and LowerPane mode, only 2 panes will be effected. If one of the panes reaches its minimum or maximum, adjustment will stop, instead of finding the next adjustable pane.

SashAction(Start, LowerPane):

Change the cursor from the crosshair to a downward pointing arrow. Determine the lower pane which will be adjusted (usually the pane below the pane to which the sash is attached).

SashAction(Move, Upper):

Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget below the upper pane can be adjusted and make the appropriate adjustments.

SashAction(Move, ThisBorder):

Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Adjust as needed (and as possible) the upper and lower panes selected when the SashAction(Start, ThisBorderOnly) action was invoked.

SashAction(Move, Lower):

Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget above the lower pane can be adjusted and make the appropriate adjustments.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWPRIMITIVE(3X), XWSASH(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

NAME

XwworkSpaceWidgetClass - the X Widget's empty window widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/WorkSpace.h>
```

CLASSES

The WorkSpace widget is built from the Core and Primitive classes.

The widget class to use when creating a workspace is **XwworkSpaceWidgetClass.**

The class name for this widget is **WorkSpace.**

DESCRIPTION

The WorkSpace widget provides the application developer with an empty primitive widget. This widget can be used by the application as a non-widget graphics area. Callback types are defined for widget exposure and resize to allow the application to redraw or reposition its graphics. Keyboard, button press and button release callbacks are also defined to provide the application an easy means of getting normal input from the widget. Other types of input can be gathered from the widget by adding event handlers.

WorkSpace supports the highlighting and shadowed border drawing defined by the Primitive widget class. If the workspace widget has a highlight thickness, the application should take care not to draw on this area. This can be done by creating the graphics context to be used for drawing in the widget with a clipping rectangle set to the size of the widget's window inset by the highlight thickness.

NEW RESOURCES

The WorkSpace widget defines a set of resource types used by the programmer to specify the data for the workspace. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget.

WorkSpace Resource Set			
Name	**Class**	**Type**	**Default**
XtNexpose	XtCCallback	Pointer	Null
XtNkeyDown	XtCCallback	Pointer	Null
XtNresize	XtCCallback	Pointer	Null

XtNexpose

This resource defines a callback list which is invoked when an exposure event occurs on the widget. The call_data parameter for the callback will contain a Region structure containing the exposed region.

XtNkeyDown

This resource defines a callback list which is invoked when keyboard input occurs in the widget. The call_data parameter for the callback will contain the key pressed event.

XtNresize

This resource defines a callback list which is invoked when the widget is resized. The widget parameter can be accessed to obtain the new size of the widget.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	Dimension	1
XtNdepth	XtCDepth	int	0
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNheight	XtCHeight	Dimension	0
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNtranslations	XtCTranslations	XtTranslations	NULL
XtNwidth	XtCWidth	Dimension	0
XtNx	XtCPosition	Position	0
XtNy	XtCPosition	Position	0

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNrelease	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to this widget.

TRANSLATIONS

The following translations are defined for the WorkSpace widget.

```
<KeyDown>:      keydown()
<BtnDown>:      select()
<BtnUp>:        release()
<EnterWindow>:  enter()
<LeaveWindow>:  leave()
```

ACTIONS

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the workspace's border will be highlighted. Otherwise no action is taken.

keydown:
Keyboard input occurring on a workspace invokes the workspace's XtNkeyDown callback list.

leave: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the workspace's border will be unhighlighted. Otherwise no action is taken.

release:
Release invokes the workspace's primitive XtNrelease callback list.

select: Selections occurring on a workspace invokes the workspace's primitive XtNselect callback list.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X),
Programming With The HP X Widgets,
Programming With Xt Intrinsics,
Programming With Xlib.

This page left blank intentionally.

Index

All references to appendix A, "Reference Information," are indicated as page A-1.

A

Accelerator, 1-2
Accelerator, menu, 4-11
Action lists, 7-7
Adding callback procedures, 3-7
Adding callbacks, 3-9
Advanced program, 3-21
Advanced program, font selection, 3-31
Advanced program, source code, 3-25
Advanced programming techniques, 3-18
Application Control, 6-2
Application defaults files, 3-13
Argument lists for widgets, 3-6
Argument values, setting, 3-18
Arguments, setting, 3-24
Arrow, 2-14

B

Background tile, 2-15
BulletinBoard, 2-7

C

Callback, 1-2
Callback procedure, writing, 3-8
Callback procedures, adding, 3-7
Callback procedures, writing, 3-24
Callback resources, 3-10
Callbacks, 4-11
Callbacks, adding, 3-9
Cascade, 2-14
Cascade menu, 4-8
Categories of widgets, 2-3
Child widget, 1-2
Class, 1-3
Class initialization, 7-10
Class, meta, 1-3
Class record, 7-9
Classes, widget, 2-1
Classing, widget, 7-33
Color, 3-14
Common resources, 7-1
Composite manager widget, 1-3
Composite widget class, 1-3
Constraint, 1-3
Constraint resource set, 5-2
Constructing a widget, 7-2
Core, 1-3
CreateTile, 2-15
Creating a widget, 3-10

D

Defaults, files", 3-12
Defaults file, example, 3-14, 3-31
Defaults files, app-defaults, 3-13
Defaults files, application, 3-13
Defaults files, example, 3-45
Defaults files, user, 3-14
Defaults files, .Xdefaults, 3-13, 3-14
Destroy procedure, 7-13
Display widgets, 2-4
Documentation, 1-6
Double click, 1-3, 4-6
Drag, 1-3, 4-6

E

Editing widgets, 2-4
Efficient operation, 3-13
Event handler, 4-11

F

Form, 2-7
Form widget, code example, 5-2
Form widget, constraints, 5-2
Form widget, example, 5-3
Form widget, using, 5-1
Frame, 2-14

G

Grab, 1-3, 4-6

H

Header file, private, 7-2, 7-3
Header file, public, 7-2, 7-6, 7-9
Header files, 7-6
Header files, including, 3-4
Hierarchy, 3-22, 3-35

I

ImageEdit, 2-5
Include files, 7-6
Initialization, class, 7-10
Initializing, 3-5
Instance, 1-4, 2-1
Instantiate, 1-3
Interface, Keyboard, 6-1

K

Keyboard
  Input Processing, 6-1
  Traversal, 6-2
Keyboard interface, 4-11, 6-1

L

Layout widgets, 2-7
Linking libraries, 3-12
List, 2-8

M

manager widget, 1-3
Menu
  Cascade, 4-8
  Creation, 4-9
  Data Specification, 4-4
  Hierarchy, 4-1
  Popup, 4-2
  Pulldown, 4-2
Menu accelerator, 4-11
Menu components, 4-5
Menu manager, 4-5
Menu Manager Views, 4-2
Menu Pane Widget, 4-8
Menu, Popup, 2-14
Menu, Pulldown, 2-14
Menu, sample program, 4-13
Menu system description, 4-1
Menu widgets, 2-13
MenuButton, 2-14
MenuButton widget, 4-8
MenuMgr, 4-5
Menus, 4-1
Menus, using, 4-10
MenuSep, 2-14, 4-9
Meta class, 1-3
Miscellaneous widgets, 2-14
Mixing menu accelerators and traversal, 4-11
Mnemonic, 4-11

P

Panel, 2-9
Popup, 1-3
Popup menu, 4-2
PopupMgr, 2-14, 4-6
Post, 1-4, 4-6
Primitive widget, 1-4
Private header file, 7-2, 7-3
Programming, advanced, 3-18
Public header file, 7-2, 7-6, 7-9

Pulldown, 2-14
Pulldown manager, 4-7
Pulldown Menu, 4-2
PushButton, 2-6

R

Redisplay procedure, 7-11
Resize procedure, 7-13
Resources, common, 7-1
Resources, multi-state button, 7-9
RowCol, 2-10

S

Sample program, advanced, 3-21
Sample program, font selection, 3-31
Sample program, menu, 4-13
Sample program, simple, 3-2
Scroll bar, 2-15
ScrolledWindow, 2-11
Setting argument values, 3-18
Setting arguments, 3-24
SetValues procedure, 7-12
Source code, 7-13
Source code, demo2, 3-25
Source code file, 7-6
Source code, xfonts.c, 3-37
StaticRaster, 2-4
StaticText, 2-4
Sticky, 4-6
Subclass, 1-4

T

TextEdit, 2-5
Tile, 2-15
Title bar, 2-15
Toggle, 2-6
Translation, 1-4
Translations, 7-7
Traversal Activation, 6-3
Traversal Key Definitions, 6-3
Traversal Requirements, 6-6
Tree, widget, 3-35

U

User selection widgets, 2-6
Using menus, 4-10
Using Widgets, 3-1
Utilities, 2-15

V

Valuator, 2-15
Visual Attributes, 6-2
VPanedWindow, 2-12

W

Widget, 1-1, 1-4, 2-1
Widget, child, 1-2
Widget classes, 2-1
Widget, classing, 7-33
Widget classing, implementing, 7-34
Widget, composite class, 1-3
Widget, composite manager, 1-3
Widget, constructing, 7-2
Widget, creating, 3-10
Widget hierarchy, 3-22, 3-35
Widget instance, 1-4, 2-1
Widget, making visible, 3-11
Widget, manager, 1-3
Widget, MenuButton, 4-8
Widget, primitive, 1-4
Widget program, simple, 3-2
Widget tree, 1-4, 3-35
Widgets, argument lists for, 3-6
Widgets, categories of, 2-3
Widgets, display, 2-4
Widgets, editing, 2-4
Widgets, form, 5-1
Widgets, layout, 2-7
Widgets, menu, 2-13
Widgets, miscellaneous, 2-14

Widgets, user selection, 2-6
Widgets, using, 3-1
Widgets, writing, 7-1
Writing widgets, 7-1

X

xfonts, 3-31
XtAddCallback, Defined, 3-9
XtCallbackProc, Defined, 3-8
XtCreateManagedWidget, 3-10
XtCreateWidget, Defined, 3-11
XtInitialize, 3-5
  Defined, 3-5
XtNumber, 3-11
XtRealizeWidget, 3-11
  Defined, 3-12
XtSetArg, 3-6, 3-7

Please print or type your name and address.

Name: _____

Company: _____

Address: _____

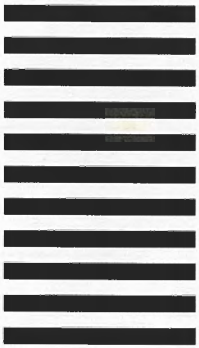
City, State, Zip: _____

Telephone: _____

Additional Comments: _____

Programming With the HP X Widgets and the Xt Intrinsics
HP Part Number 98794-90000
E0689

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 38 CORVALLIS, OR

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY
CWO PRODUCT MARKETING
1000 NE CIRCLE BLVD
CORVALLIS OR 97330-9988

MANUAL COMMENT CARD

HP Part Number 98794-90000 E0689

Your comments and suggestions help us determine how well we meet your needs.

Programming With the
HP X Widgets and the Xt Intrinsics

	Agree				Disagree
The manual is well organized.	○	○	○	○	○
It is easy to find information in the manual.	○	○	○	○	○
The manual explains features well.	○	○	○	○	○
The manual contains enough examples.	○	○	○	○	○
The examples are appropriate for my needs.	○	○	○	○	○
The manual covers enough topics.	○	○	○	○	○
Overall, the manual meets my expectations.	○	○	○	○	○

You have used this product:

__ Less than 1 week __ Less than 1 year __ More than 2 years
__ Less than 1 month __ 1 to 2 years

fold

Please write additional comments, particularly if you disagree with a statement above. Use additional pages if you wish. The more specific your comments, the more useful they are to us.

Comments: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

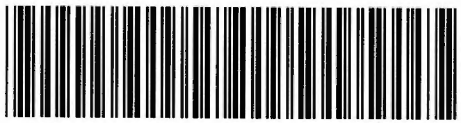
_____

_____

_____

Please Tape Here

HP Part Number
98794-90000
Microfiche No. 98794-99000
Printed in U.S.A. E0689



98794-90602
For Internal Use Only