The OpenWidget Set

Pete Ware

April 23, 1992

Contents

1 Goals 1

1.1 Copyrights 1

1.2 Types of Widgets 2

2 The Environment 3

2.1 Basic Tools 3

2.2 Naming Conventions 4

2.3 Documentation 5

3 The Widget Set 6

3.1 Conventions 6

3.2 Widget Overview 7

4 The Basic Widgets 11

4.1 Core Widget Class 12

4.2 Simple Widget Class 15

4.3 Label Widget Class 19

4.4 Button Widget Class 22

4.5 Scrollbar Widget Class 28

4.6 MenuButton Widget Class 35

4.7 MenuBarButton Widget Class 40

4.8 Grip Widget Class 45

5 The Output Gadgets **51**

5.1 Object Class 52
5.2 RectObj Class 53
5.3 ObjDraw Class 54
5.4 ObjArrow Class 56
5.5 ObjPixmap Class 58
5.6 ObjText Class 60
5.7 ObjBorder Class 64
5.8 Obj2dBorder Class 65
5.9 Obj3dBorder Class 68

6 The Manager Widgets **71**

6.1 Composite Widget Class 71
6.2 CompBase Widget Class 75
6.3 Box Widget Class 78
6.4 Row Widget Class 82
6.5 Column Widget Class 84
6.6 Menu Widget Class 88
6.7 MenuBar Widget Class 91
6.8 MenuBox Widget Class 93
6.9 Shell Widget Class 96
6.10 OverrideShell Widget Class 100
6.11 MenuShell Widget Class 103

7 The Manager + Data Widgets **108**

7.1 Constraint Widget Class 108
7.2 ConsBase Widget Class 110
7.3 List Widget Class 114
7.4 Vpane Widget Class 118

8 Utility functions **123**

8.1 Inheriting Resources 123

8.2 Converters 124
8.3 Creating Menus 124
8.4 Miscellaneous 125

9 The build Program **128**
9.1 Output Files 128
9.1.1 Public Include File 129
9.1.2 Private Include File 131
9.1.3 Initialization Include File 132
9.1.4 Documentation 134
9.2 Input Files 135
9.2.1 Variables 135
9.2.2 Class Record 137
9.2.3 Instance Record 137
9.2.4 Constraint Record 137
9.2.5 Actions 137
9.2.6 Translations 137
9.2.7 Functions 137

10 Testing **138**
10.1 Simple Widgets 138
10.2 Composite Widgets 139

Chapter 1

Goals

A basic assumption of this project is that a widget set, developed by a group of technically excellent people, widely available and distributed, will be a high quality product. This widget set must be publically available, freely distributed and very well documented. Except for the political intent of the Free Software Foundation, this project has similar goals as the FSF – software with high quality and wide availability produced by a group of people.

The goal of this project is to develop a widget set with functionality similar to that from the OSF/Motif widgets but implemented in a manner that allows others to augment the widgets. In particular, avoiding the many design errors that destroy the object oriented nature of Xt based toolkits.

1.1 Copyrights

The issue of copyrights is something that is very bothersome. The intent of this development effort is a body of code that is useful, available and continues to improve. There is little concern about an individual or company making money from this work – if they can compete against something that is free more power to 'em. Of greater concern is fragmentation and differing versions of the software. Another way of saying this is that we want to encourage people to make contributions, not to hoard their improvements. The need for 30 lines of copyright information is rather daunting especially when all it purports to do is give everything away. Personally, and this is something the entire group should decide, I'd rather not place any copyright information and let these lapse into the public domain. It should be our own efforts that keep the project together.

1.2 Types of Widgets

The widget set includes typical widgets such as labels, push buttons, menus, scrollbars, row manager, column manager, and viewports. Additional widgets such as xy plots, bar graphs, pie charts, and bitmap editor that are missing from most widget sets will also be included. Many more are actually planned, the above are mentioned to give an idea of the scope of the project[1].

An important feature of every widget will be the ability to provide conveniant hardcopy, save and restore state, provide help, and automatic class documentation.

This widget set will be comparable to the Motif, Athena, and OLIT widget sets in functionality. The widgets will be developed by a group of people with the primary goal being a high quality widget set. The typical widgets expected of a widget set will be provided plus some additional widgets. All widgets in the set will include special features.

[1] Please consider this document to be work in progress. Nothing is confirmed at this point. In particular, the set of widgets can certainly be greater than this and the order of implementation is completely up for grabs.

Chapter 2

The Environment

This defines the basic assumptions made while developing the Xo widget set and the assumptions a developer using the Xo widget set can make. This also defines the naming conventions to follow for writing new widgets and methods. Also, the expected locations for include files and libraries and what other widgets can be relied on when developing composite widgets.

2.1 Basic Tools

This development is oriented toward R4 with no attempt to provide compatability with R3.

Do not assume the presence any other widget sets. In particular, do not assume Athena widgets. If writing composite widgets that create other widgets, make the widget class a resource and possibly add class methods that manipulate that widget in the appropriate way. If possible, arrange it so that the application writer can subsitute any of his favorite widgets. In particular, when using scrollbars, make sure the type of scrollbar is settable.

Code should be as portable as Xt. Supporting varients of Unix is a must, VMS and other operating systems are optional.

Source should compile on a wide range of machines. In particular, try using *gcc -Wall*, this insures one's code is tight. Code should *lint* cleanly although this is sometimes difficult with Xt based code.

A tool called *build* exists to insure the widget set is consitently written and well documented. One describe's a widgets superclass, class variables (only those that are new), instance variable, new resources, new actions and translations. *build* then takes this description and generates appropriate documentation, public include files, private include files and the resource and class declarations that can be compiled. Using this insures that include files

and documentation match with the actual source code.

As such things go, *build* has fallen short of the original intention to provide a fully interactive editor for the creation of widget code. It also is not intended to be able to "interpret" existing C code and provide a more malleable environment – that is way, way beyond it's intended scope.

2.2 Naming Conventions

The goal of the following naming conventions is to allow applications to include multiple widget sets without collisions in naming structures, such as the instance part, and making it clear where particular methods, widget classes or variables are defined. The acid test is to insure that if some other widget set has a widget with the same name, they both can be used simultaneously.

We will follow the naming conventions used by Motif and formalized by Gene Dykes with some modifications as experience dictates.. A toolkit uses a special prefix with all its widgets. In the case of the Openwidgets, this prefix is "o". Using that prefix and a hypothetical Label widget as an example, the conventions are as follows:

1. The toolkit library name is *libXo.a*.
2. The widget class variable is xoLabelWidgetClass.
3. Include files are found in directives <X11/Xo/Label.h>[1].
4. Wrap include files in ifdef's to avoid multiple inclusion. In particular, use the name of the widget set to prefix the name. For example,
   ```
   #ifndef _XO_LABEL_H_
   #define _XO_LABEL_H_
   ...
   #endif /* _XO_LABEL_H_ */
   ```
5. Keep include file names to nine characters or less. To do so strip the lower case letters a word at a time until the basename is 9 characters or fewer (but strip as few letters as possible from the initial word).

   For example:
   `<X11/Xo/VeryLongWidgetName.h>`
   → `<X11/Xo/VeryLWN.h>`
   `<X11/Xo/Verylongwidgetname.h>`
   → `<X11/Xo/Vlongw.h>`

[1] I took the liberty of changing this to <X11/Xo/file> instead of the original <Xo/file> since this matches the Athena widgets.

6. The `class_name` field in the core structure is "Label"[2].
7. Enumerated resource values are called XoCAPITALIZED_WORDS.
8. Public functions are named XoLabelFunctionName(). Use leading underscores, '_', to indicate it is private to the widget set.
9. Global variables are named xoGlobalVariable. Use leading underscores, '_', to indicate it is private to the widget set.
10. Typedefs, structures and enums are named XoRestOfTypedef or struct XoStructName. Use leading underscores, '_', to indicate it is private to the widget set.

2.3 Documentation

This document, like the entire project, is intended to be worked on by multiple people. Ideally (I know it's not really possible but we can try) this document should address three audiences:

1. Widget writers who need to know the the details of how a widget is going to work.
2. Application developers who need to know the capabilities of the widget set.
3. Users that need to set resources for a widget and possibly (although this may be better left to the application developer to document) understand it's behaviour.

A description of each widget's resources is kept in a separate file called *res_<Widget>.tex*.

Originally, the entire documentation for each widget was in two files, one describing the resources, just as we currently do, and one file filling in the details. This was meant to make it easier to allow multiple people to document widgets independently. The reality was a bunch of fairly small files that provided more clutter than clarity.

[2] Actually, in Motif it is "XmLabel"

Chapter 3

The Widget Set

The following chapters describe the set of widgets to be developed in the detail necessary for implementing, programming with and using the widgets. This chapter introduces you to the widget set and provided a guide for interpreting the volumes of data to follow.

As mentioned earlier, the goal of this widget set is to provide a consistent, full featured widget set with many worthwhile widgets such as plots, pie charts, etc that are missing from other widget sets. This will be realized by providing the frame work so that other's contribute such widgets and the become a part of this effort.

Presenting a widget set offers many challanges. In particular ordering the information so that it can meet the requirement of presenting a cohesive picture and allowing easy access as the material becomes familiar is almost impossible – at least with a single manual. At this point, the decision is to present things in a manner that a widget writer will appreciate and probably no other audience. This is something that can be changed at a later date.

3.1 Conventions

Since this widget set is under development, included in each section is the current status of the widget. This status falls into several categories:

name Only the name and a few words describing what this widget may one day do exists.

spec Only the specification exists.

work Work is under progress on this widget.

done This widget is useable.

3.2 Widget Overview

The following chapters present several different types of widgets. The ones that a user typically interacts with and generally recieve most of the user's attention are covered first in the chapter on basic widgets. Following that is the set of widgets[1] that implement most of the outward appearance of the basic widgets. Following these two chapters are two more chapters on manager widgets that are responsible for laying out other widgets appropriatly.

Experience has shown that most application writer's spend more time working[2] with manager widgets to affect the correct layout and behaviour. Most user's are primarily interested in the basic widgets. The advanced user may be interested in the widgets that implement the looks of the basic widgets.

In any case, the following diagram shows the entire widget hierarchy with the page number of each widget description included. Widgets defined by the Xt Intrinsics are written in **boldface**. All of these widgets are implemented.

Core (page 12)
- Simple (page 15)
  - Label (page 19)
    - Button (page 22)
      - MenuButton (page 35)
        - MenuBarButton (page 40)
      - Grip (page 45)
  - Scrollbar (page 28)
- **Composite** (page 71)
  - **Shell** (page 96)
    - **OverrideShell** (page 100)
      - MenuShell (page 103)
  - CompBase (page 75)
    - Viewport (page **??**)
    - Box (page 78)
      - MenuBox (page 93)
    - Row (page 82)
      - MenuBar (page 91)
  - **Constraint** (page 108)
    - ConsBase (page 110)
      - Column (page 84)
        - List (page 114)
        - Vpane (page 118)
        - Menu (page 88)

Object (page 52)
- **RectObj** (page 53)

[1] Actually gadgets, to use the exact terminolgy, or windowless widgets
[2] Fighting is more accurate.

ObjDraw (page 54)
- ObjText (page 60)
- ObjArrow (page 56)
- ObjPixmap (page 58)
- ObjBorder (page 64)
  - Obj2dBorder (page 65)
  - Obj3dBorder (page 68)

The following is the set of widgets we'd ideally like implemented. Each includes a brief description of the functionality. Widget names in **boldface** are planned for immediate implementation. Widget names in *slanted* text will be implemented later.

2dBorder A gadget to draw simple, two dimensional borders.

Border A gadget to draw borders that give a raised or indented appearance

Ball/Socket A gadget used to indicate the state of something. Contains a hollow hemisphere that is empty when the state is off and filled with a sphere when the state is on.

BarGraph A widget that displays multiple data values in the typical bar graph format used by many businesses.

Box A manager widget that places widgets in a rectangular arrangment.

Button A widget that allows the user to "press" a button to cause something to happen.

Checkmark A gadget that displays a checkmark in a box when the state is on and leaves the box empty when the state is off.

Column A manager widget that places other widgets in a vertical column.

Diamond A gadget that displays a diamond ($\diamond$) with a three dimensional border. When the state is on the diamond appears lowered and when the state is off the diamond appears raised.

Down Arrow A gadget that displays a down arrow ($\bigtriangledown$). When the state is on, the arrow appears pressed and when the state is off the arrow appears raised.

Horizontal Slider
A gadget that draws a raised box with a tick mark to indicate the center.

InterMgr A manager widget that allows the user to interactively place and size other widgets.

Label A widget that displays either text or a pixmap.

Left Arrow A gadget that displays a left arrow (◁). When the state is on, the arrow appears pressed and when the state is off the arrow appears raised.

MenuBar A manager widget for holding pulldown menus.

MenuBox A manage widget that uses the same layout policy as the Box widget but serves the same function as a MenuBar.

MenuButton A widget that is placed inside of a menu. Allows the display of either text or a pixmap. Submenus can be attached to a menu item.

Menu A manager widget for holding the widgets in a menu.

MultiViewport
A manager widget that can display one or more children that are scrolled within a window but can keep several other widgets stationary. This is useful for forms that require the titles to be properly placed with respect to a large number of items.

PieChart A widget that displays a series of data as a pie.

Pixmap A gadget that displays a pixmap.

Plot A manager widget that plots (x, y) data as a series of lines and provides axis and various labels.

Pull right A gadget that displays a right pointing arrow (→)[3]. Useful for menu items with pull right submenus.

RadioButton Much like a button except it includes a symbol to indicate if the state is on or off. It uses a diamond gadget to indicate the current state.

Right Arrow A gadget that displays a right arrow (▷). When the state is on, the arrow appears pressed and when the state is off the arrow appears raised.

Row A manager widget that arranges other widgets horizontally.

Scale A widget that allows a slider to select a value from a range of values.

Seperator A gadget that draws a chiseled edge. Typically used in a *Label* widget to distuinguish between sections in a menu.

Simple A widget that provided common functionality for many of the widgets. Never directly used.

Square A widget that arranges it children in a square. Either so the number of children on each row and column are the same of so the width is the same as the height.

[3]Guess the <direction>Arrow gadgets should have been called triangle gadgets

Text A gadget that displays lines of text[4].

Toggle A widget much like the radio button widget except it uses a checkmark to indicate if the current state.

Up Arrow A gadget that displays an up arrow (△). When the state is on, the arrow appears pressed and when the state is off the arrow appears raised.

Valuator A widget that displays a see saw like bar that allows the user to continuously increase or decrease a value.

Vertical See Saw A widget similar to the valuator except it displays it vertically.

Vertical Slider A gadget similar to the horizontal slider except it displays it horizontally.

ScrollBar A widget that allows the user scroll other objects forward or backward using either arrows or a slider.

Viewport A manager widget that allows a single child to grow and places scrollbars to let the user see portions that are currently out of view because the child is larger than the parent.

Vpane A manager widget that arranges children vertically with each child seperated by a grip that allows the user to resize the children.

By providing the basic framework and sample implementations of a few of the above should make it easy for others to contribute further widgets.

[4]Should probably call this something else since Text seems to imply much more

Chapter 4

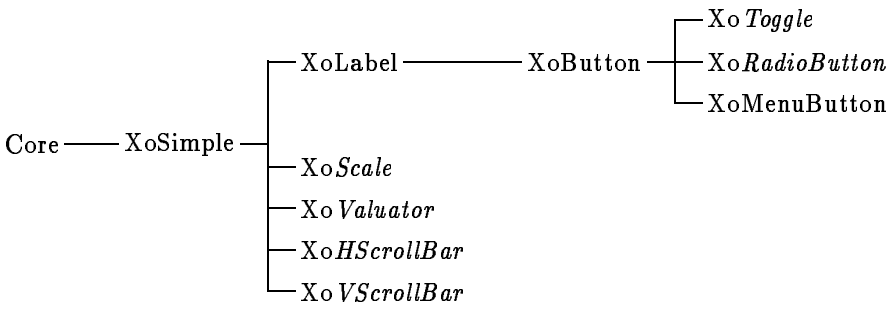
The Basic Widgets

These are the widgets that most users interact with an application through. The output widgets are intended primarily for use of widget writers and those brave user's who want the ultimate in customizability. The manager widgets are primarily of interest to the application developer that needs to deal with the ugly realities of screen layout. In many ways, the basic widgets are what define a widget set.

As defined by the Xt Intrinsics, all widgets inherit behaviour from the Core widget class. Accordingly that is described first although one will rarely see a Core widget created. Understanding how the Core widget works means you understand most of the Xt Intrinsics – no small task. It is only described this one time even though the manager widgets inherit behaviour from Core, it's assumed one already familiar with the Core widget at that stage.

After the Core widget is presented, the *Simple* widget class is described. This class implements most of the common behaviour and look of all the other widgets in the OpenWidget set.

The following diagram indicates the rest of chapters. Starting at the left most edge and visiting all the widgets from left to right and top to bottom (an inorder traversal with upper branches visited first, for computer science people) is the layout of the following chapters.



4.1 Core Widget Class

This is the parent class of all widgets. It provides the information needed by the Intrinsics to manage windows, resources, display, etc. Understanding of this class is primarily needed by widget writers and not so much application writers or users.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Core.h>
extern WidgetClass coreWidgetClass;
```

The following resources are provided by the *Core* widget:

Core			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0

continued

continued			
Name	**Class**	**Type**	**Default**
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0

XtNaccelerators
Translations that can be added to other widgets and behave as if they occured in this widget. [*accelerators*]

XtNancestorSensitive
Set to False when an ancestor of this widget is no longer accepts input. If this is true, it may be appropriate to visually indicate this widget cannot accept input. Inherited from RectObj. [*ancestor_sensitive*]

XtNbackground
The background color of the window. [*background_pixel*]

XtNbackgroundPixmap
The pattern to use when drawing the background [*background_pixmap*]

XtNborderColor
Color for the border of the window. [*border_pixel*]

XtNborderPixmap
The pixmap used to tile the window border [*border_pixmap*]

XtNborderWidth
The width of the border in pixels. Inherited from RectObj. [*border_width*]

XtNcolormap
The colormap used when displaying this widget. [*colormap*]

XtNdepth The number of planes in the window of this widget [*depth*]

XtNdestroyCallback
List of callbacks executed when the widget is destroyed. Inherited from Object. [*destroy_callbacks*]

XtNheight The height of this widget in pixels. Inherited from RectObj. [*height*]

XtNmappedWhenManaged
If this is True, then the window is automatically mapped when it is managed. If it is False, then the window must be mapped by the application. [*mapped_when_managed*]

XtNscreen Which of the screens this widget is displayed on [*screen*]

XtNsensitive
Indicates if this widget accepts any input. Set by calls to XtSetSensitive(). Inherited from RectObj. [*sensitive*]

XtNtranslations
The translation table used by this widget to determine how it behaves. [*tm*]

XtNwidth The width of this widget in pixels. Inherited from RectObj. [*width*]

XtNx The x coordinate of this widget relative to the parent. Inherited from RectObj. [*x*]

XtNy The y coordinate of this widget relative to the parent. Inherited from RectObj. [*y*]

being_destroyed
marked for destroy.

constraints constraint record.

event_table private to event dispatcher.

managed is widget geometry managed?.

name widget resource name.

num_popups how many popups.

parent parent widget.

popup_list list of popups.

self pointer to widget itself.

visible is window mapped and not occluded?.

widget_class pointer to Widget's ClassRec.

window window ID.

xrm_name widget resource name quarkified.

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

4.2 Simple Widget Class

This provides the base class for non-manager and non-gadget widgets in this widget set. In plainer terms, most widgets in this widget set inherit appearance and behaviour from this class. It's function is similar to that provided by *Core* but is specific for this widget set.

The border gadget is sized to the same width and height as this widget. It is assumed that *border_width* space is used by to draw the border.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Simple.h>
extern WidgetClass xoSimpleWidgetClass;
```

The following resources are provided by the *Simple* widget:

Simple			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Simple (see page 15)			
backingStore	BackingStore	int	"NotUseful"
borderClass	Class	WidgetClass	"None"
borderWidget	BorderWidget	Widget	NULL
cursor	Cursor	Cursor	"cross"
foreground	Foreground	Pixel	XtDefaultForeground
inheritSet	InheritSet	XtCallbackList	NULL
insensitiveStipple	InsensitiveStipple	Pixmap	NULL

continued

continued			
Name	**Class**	**Type**	**Default**
traverseAccept	TraverseAccept	Boolean	(XtPointer) True

XtNbackingStore
Gives a hint to the server if it would be useful to maintain a copy of this widget to allow faster refreshes. The cost is additional resources consumed (which can add up to a great deal). Most widgets are simple enough that it is faster for the widget to just redraw – although there are plenty exceptions especially when bandwidth and transmission delays are included. Possible values are NotUseful, WhenMapped, Always. [*backing_store*]

XtNborderClass
Object class that implements drawing the border. [*border_class*]

XtNborderWidget
This is the instance of the widget that draws the border You have two choices, either set the *WidgetClass* at initialization time or create an object with this widget as it's parent and then use *XtSetValues()* to set this resource. The old widget is destroyed. [*border_widget*]

XtNcursor The cursor to use when in this window. There are several ways to create cursors: font, bitmap or glyph. We should provide access to all three ways in the converter. Font is the easiest, although we should provide a string to cursor name.

Cursor color should just be based on *foreground* color, although we could certainly add another resource but it hardly seems necessary. [*cursor*]

XtNforeground
The color for drawing primary objects such as text and lines. Changing this during execution causes the object to change color. [*fg*]

XtNinheritSet
This list of callbacks is executed to allow gadgets to update their instance variables when values in the parent would change. It allows gadgets to be transparent to the application and user. It can be viewed as a way of a child adding additional resources to the parent.

Internally, the function *_xoInheritSet()* implements this ability by getting the value of *XtNinheritResources* and scanning the argument list for corresponding values. The new values are collected and *XtSetValues()* executed on the gadget.

Similar capabilities are provided by *_xoInheritInit()* which queries the resource database pretending to be the parent widget. [*inherit_set*]

XtNinsensitiveStipple
: The pixmap to use as a stipple pattern to draw through when the widget is insensitive. Something like a 50 percent gray pattern should be used. [*insensitive_stipple*]

XtNtraverseAccept
: Indicates if this widget accepts the input focus. The core_class.accept_focus method should examine this value to determine if it should even try to accept keyboard input. If it is false, it should not. [*traverse_accept*]

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

```
typedef struct {
    WidgetClass      border_class;        /* Widget Class of border object */
    Widget           border_widget;       /* draws the border */
    Pixel            fg;                  /* foreground color */
    Cursor           cursor;              /* cursor pattern */
    int              backing_store;       /* backing store */
    Pixmap           insensitive_stipple; /* stipple when insensitive */
    XtCallbackList   inherit_set;         /* to set inherited values */
    Boolean          traverse_accept;     /* if accept focus */
} XoSimplePart,                  simple;
} XoSimpleRec,       * XoSimpleWidget;
```

4.3 Label Widget Class

This displays either text or a picture that is used to label other fields. It has no input behaviour. It is drawn with minimal decoration so as not to distract from other fields.

Much of the appearance of a *Label* widget is determined by the object class set in *drawClass*. A typical example is The class variable is *xoObjTextWidgetClass*. You will need to refer to the resources of that gadget to determine the possible variations. For example, The class variable is *xoObjTextWidgetClass* defines *label* and *font* for defining the text to display and how it should look. One is free to choose other gadgets such as a The class variable is *xoCheckmarkWidgetClass* or The class variable is *xoObjPixmapWidgetClass* to implement entirely different drawing characterstics.

The preferred size of the *Label* widget is set to the size of the *drawClass* widget plus the space used by the borderClass widget. In other words:

$$width = draw\_widget'swidth + 2 * border\_widget'sborder\_width$$

and the preferred height:

$$height = draw\_widget'sheight + 2 * border\_widget'sborder\_width$$

The size of the *drawWidget* is never changed. The size of the *borderWidget* is kept the same size as the *Label* widget.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Label.h>
extern WidgetClass xoLabelWidgetClass;
```

The following resources are provided by the *Label* widget:

Label			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Simple (see page 15)			
backingStore	BackingStore	int	"NotUseful"
borderClass	Class	WidgetClass	"None"
borderWidget	BorderWidget	Widget	NULL
cursor	Cursor	Cursor	"cross"
foreground	Foreground	Pixel	XtDefaultForeground
inheritSet	InheritSet	XtCallbackList	NULL
insensitiveStipple	InsensitiveStipple	Pixmap	NULL
traverseAccept	TraverseAccept	Boolean	(XtPointer) True
Label (see page 19)			
drawClass	Class	WidgetClass	"ObjText"
drawWidget	DrawWidget	Widget	NULL
gravity	Gravity	XoGravity	"Center"
resizeDraw	ResizeDraw	Boolean	False

XtNdrawClass

The class of the object that implements drawing the the label. The label widget creates one of these objects The width and the height are used to place the object according to the value of *gravity*. [*draw_class*]

XtNdrawWidget

This is the instance of the widget that actually does the drawing. You have two choices, either set the *WidgetClass* at initialization time or create an

object with this widget as it's parent and then use *XtSetValues()* to set this resource. The old widget is destroyed. [*draw_widget*]

XtNgravity Gravity controls the placement of the object within the widget. Values are specified in points of the compass with North being up [1] and West to the left. Possible values are (moving clockwise) North, NorthEast, East, SouthEast, South, SouthWest, West, NorthWest and Center. Center gives the intuitive result although may not always be what is desired.

To make text that is flush to the right side of the widget, set *justify* to right and *gravity* to West. [*gravity*]

XtNresizeDraw

If true, then the drawing widget is resized to be the same as the label widget. Otherwise, it is kept at whatever the QueryGeometry returns. [*resize_draw*]

```
typedef struct _XoLabelRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
```

[1] my apologies to those from down under

```
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetClass     border_class;       /* Widget Class of border object */
        Widget          border_widget;      /* draws the border */
        Pixel           fg;                 /* foreground color */
        Cursor          cursor;             /* cursor pattern */
        int             backing_store;      /* backing store */
        Pixmap          insensitive_stipple; /* stipple when insensitive */
        XtCallbackList  inherit_set;        /* to set inherited values */
        Boolean         traverse_accept;    /* if accept focus */
    } XoSimplePart,                 simple;
    typedef struct {
        Boolean         resize_draw;        /* if resize draw widget */
        XoGravity       gravity;            /* how to place the label */
        WidgetClass     draw_class;         /* Widget class that draws */
        Widget          draw_widget;        /* Widget for drawing text. */
    } XoLabelPart,                  label;
} XoLabelRec,               * XoLabelWidget;
```

4.4 Button Widget Class

This allows the user to click a button. A 3d border is provided to make the widget look like a button that one pushes.

Note that all the appearance is inherited from The class variable is *xoLabelWidgetClass* except *borderClass* is set to "ObjBorder" by default.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Button.h>
extern WidgetClass xoButtonWidgetClass;
```

The following resources are provided by the *Button* widget:

Button			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Simple (see page 15)			
backingStore	BackingStore	int	"NotUseful"
borderClass	Class	WidgetClass	"None"
borderWidget	BorderWidget	Widget	NULL
cursor	Cursor	Cursor	"cross"
foreground	Foreground	Pixel	XtDefaultForeground
inheritSet	InheritSet	XtCallbackList	NULL
insensitiveStipple	InsensitiveStipple	Pixmap	NULL
traverseAccept	TraverseAccept	Boolean	(XtPointer) True
Label (see page 19)			
drawClass	Class	WidgetClass	"ObjText"
drawWidget	DrawWidget	Widget	NULL
gravity	Gravity	XoGravity	"Center"
resizeDraw	ResizeDraw	Boolean	False
Button (see page 22)			
borderWidth	BorderWidth	Dimension	0*
borderClass	Class	WidgetClass	"DefaultBorder"*
callback	Callback	XtCallbackList	NULL
repeatDelay	RepeatDelay	int	(XtPointer) 300
repeatInterval	RepeatInterval	int	(XtPointer) 100

XtNcallback A list of functions to call when the button is activated. *call_data* is set to

set to indicate it is activated (this is done for compatibility with toggles and radiobuttons). [*callbacks*]

XtNrepeatDelay
Number of milliseconds to delay between each repeat action. When the button is initially set (activated) a timer is started. If the button has not been unset when this timer expires, the *callbacks* are executed. If this timer fires then the normal action that activates the callback is ignored (typically the button up). After the initialy delay, *repeatInterval* is used to install the next timer event. These two values allow an initial delay before starting the action which allows the user a chance to not start the repeating action but still have fast enough response once it starts. Setting this to a large number effectively eliminates the repeating. [*repeat_delay*]

XtNrepeatInterval
Number of milliseconds between each execution of the activate list. This takes effect after *repeatDelay* milliseconds and the first action is executed. [*repeat_interval*]

Core.border_width
No documentation.

Simple.border_class
No documentation.

set This private instance variable indicates the current state of the widget. Typicall 0 indicates it is off and 1 indicates it is on.

temp When the pointer temporarily leaves the window (i.e. it is initially pressed but the mouse moves out of the window), this is used to save the current state of the *set* flag.

timer_id This private instance variable contains the timer id used when starting the repeat action. When the user unsets the button, this timeout should be removed.

timer_ran This private instance variable indicates if a timer was executed. It should be cleared when the initial timer interval is installed and set when the first timer executes. The normal notify action should not execute if this is set. *Actually, it might be nice if the timer callback used the intrinsics to execute the notify action, this way subclasses could override the action and the appropriate functions would still get called. If that's the case, we should use some other convention such as a NULL event or a certain argument passed to the notify action.*

The following actions are defined by this widget.

Set	Indicates the button is currently selected. When a button is set, the background and foreground should be reversed.
Unset	Indicates the button is no longer selected. This should cause the background and foreground to be displayed normally.
UnsetTemp	Sets unset_temp if set is already True. Should cause the display to go back to the normal foreground and background. This allows the mouse to leave the window when it is depressed.
SetTemp	If unset_temp is true, sets set = True. This allows the mouse to reenter the window when the button is down.
Notify	Calls the XtNcallCallbacks if button is set. Passes the set flag as the call_data.
Highlight	Indicates focus is in this window. This should draw the 3d border so it looks like the button has been pushed.
Unhighlight	Indicates focus is not in window. This is done by drawing the 3d border it its normal condition to indicate the button protrudes.
RepeatStart	Set up timer for repeating.
RepeatEnd	Cancel the timer for repeating.
KeyIgnore	Skip keyboard events.
KeyAccept	Keyboard events are ok.
ClipStart	Starts making selection at current location
ClipEnd	Indicates the end of the selection point
ClipStuff	Asserts ownership of the current cut buffer. Name of the cut buffer can be specified. The current value can be returned either as a String or Pixmap. If displaying a pixmap, that is the only possibility. Allow a parameter to indicate if the entire item should be selected. This is for a "quick" cut.
ClipGet	Insert current selection from the specified cut buffer. If displaying text, requests it as a string. If displaying as a pixmap, requests it in that form.
ItemNext	Tells parent to move the location cursor to the next item. "Next" is defined by the parent. A simple implementationn would treat the next item as the next managed child. A more involved implementation (and far better) should move the location cursor in a left to right, top to bottom manner. When the location is at the end of the group, the parent should ask to move the focus to it's (this widgets grandparent) to the next field. (This, and the other traversals, should either use a callback list on the parent or use a XtGetValues() to get the function pointer to dereference. The traversals should not rely on the parent being a certain class.)

ItemPrev Tells parent to move the location cursor to the previous item. "Previous" is defined by the parent. A simple implementationn would treat the previous item as the prior managed child. A more involved implementation (and far better) should move the location cursor in a right to left, bottom to top manner. When the location is at the end of the group, the parent should ask to move the focus to it's (this widgets grandparent) to the previous widget.

ItemUp Tells parent to move the location cursor up an item. "Up" is defined by the parent. Behaves the same as ItemNext except it does wraps around to the last child of the parent when the first child is reached.

ItemDown Tells the parent to move the location cursor down an item. "Down" is defined by the parent. Behaves the same as ItemNext except it does wraps around to the first child of the parent when the last child is reached.

LineEnd Tells the parent to move the location cursor to the rightmost edge. The rightmost edge is defined by the parent.

LineBegin Tells the parent to move the location cursor to the leftmost edge. The leftmost edge is defined by the parent.

DataBegin Tells the parent to move the location cursor to the first item. "First" is determined by the parent.

DataEnd Tells the parent to move the locatin cursor to the last item. "Last" is determined by the parent.

GrabFocus Changes the keyboard focus to this widget, but only if traverseAccept resource is True.

The following translations are used by this widgets.

<FocusIn> Highlight()

Provides a visual feedback that the current input focus is set to this widget. The usual visual feedback is to place a black border around the widget.

<FocusOut> Unhighlight()

Should undo the visual effects from Highlight().

<Leave> UnsetTemp()

<Enter> SetTemp()

<Btn1Down> GrabFocus() Highlight() Set() RepeatStart() KeyIgnore()

<Btn1Up> Notify() Unset() RepeatEnd() KeyAccept()

<Key>Return Set() Notify() Unset()

!<Key>Tab ItemNext()

```
!Ctrl<Key>Tab          ItemNext()
!Shift<Key>Tab         ItemPrev()
!Shift Ctrl <Key>Tab   ItemPrev()
!<Key>Up               ItemUp()
!<Key>Left             ItemUp()
!<Key>Down             ItemDown()
!<Key>Right            ItemDown()
!<Key>Home             LineBegin()
!<Key>Begin            LineBegin()
!<Key>End              LineEnd()
!Ctrl<Key>Home         DataBegin()
!Ctrl<Key>Begin        DataBegin()
!Ctrl<Key>End          DataEnd()
```

```
typedef struct _XoButtonRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
```

```
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetClass     border_class;       /* Widget Class of border object */
        Widget          border_widget;      /* draws the border */
        Pixel           fg;                 /* foreground color */
        Cursor          cursor;             /* cursor pattern */
        int             backing_store;      /* backing store */
        Pixmap          insensitive_stipple; /* stipple when insensitive */
        XtCallbackList  inherit_set;        /* to set inherited values */
        Boolean         traverse_accept;    /* if accept focus */
    } XoSimplePart,             simple;
    typedef struct {
        Boolean         resize_draw;        /* if resize draw widget */
        XoGravity       gravity;            /* how to place the label */
        WidgetClass     draw_class;         /* Widget class that draws */
        Widget          draw_widget;        /* Widget for drawing text. */
    } XoLabelPart,              label;
    typedef struct {
        XtCallbackList  callbacks;          /* executed when activated */
        int             repeat_interval;    /* milli. sec. between executions */
        int             repeat_delay;       /* ms delay before executions */
        int             set;                /* current state of the button */
        XtIntervalId    timer_id;           /* current timer for repeating */
        Boolean         timer_ran;          /* indicates if a timer was actived */
        Boolean         temp;               /* temporary holder for set */
    } XoButtonPart,             button;
} XoButtonRec,                  * XoButtonWidget;
```

4.5 Scrollbar Widget Class

Implements a vertical or horizontal scrollbar. The scrollbar contains a Grip that indicates the amount of a working area visible and the relative location of the visible working area. The user can either grab the Grip directly to move the visible working area or cause the Grip to jump to a new working area.

There are also arrows at either end of the scrollbar to allow smooth scrolling by a fixed amount.

It is the responsibility of either the application or other components to implement the semantics of any scrolling. The scrollbar executes a callback list to indicate the new position of the top of the Grip and the Grip's size – both expressed as a percentage of the Scrollbar's size.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Scrollbar.h>
extern WidgetClass xoScrollbarWidgetClass;
```

The following resources are provided by the *Scrollbar* widget:

Scrollbar			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Simple (see page 15)			
backingStore	BackingStore	int	"NotUseful"
borderClass	Class	WidgetClass	"None"
borderWidget	BorderWidget	Widget	NULL

continued

continued			
Name	**Class**	**Type**	**Default**
cursor	Cursor	Cursor	"cross"
foreground	Foreground	Pixel	XtDefaultForeground
inheritSet	InheritSet	XtCallbackList	NULL
insensitiveStipple	InsensitiveStipple	Pixmap	NULL
traverseAccept	TraverseAccept	Boolean	(XtPointer) True
Scrollbar (see page 28)			
height	Height	Dimension	(XtPointer) 15*
width	Width	Dimension	(XtPointer) 15*
callback	Callback	XtCallbackList	NULL
forwardScroller	Scroller	Widget	NULL
gripClass	Class	WidgetClass	"Grip"
gripWidget	GripWidget	Widget	NULL
increment	Increment	double	".05"
minGrip	MinGrip	Dimension	(XtPointer) 15
orientation	Orientation	XoOrientation	"Vertial"
reverseScroller	Scroller	Widget	NULL
scrollerClass	ScrollerClass	WidgetClass	"Button"
scrollerDraw	ScrollerDraw	WidgetClass	"ObjArrow"
size	Size	double	"0.2"
top	Top	double	"0.0"

XtNcallback A list of functions to call when the Grip is moved. *call_data* is set to the address of *top* to indicate the center of the grip. Do not change this value. [*callbacks*]

XtNforwardScroller

This is the instance of the widget that allows the user to interactively move the grip either down or to the right a small increment at a time. To change this you have two choices: either set the *WidgetClass* at initialization time or create a widget with the scrollbar as it's parent and then use *XtSetValues()* to set this resource. The old widget is destroyed. With the second method you must use *XtCreateWidget* and not *XtCreateManagedWidget* to create the scroller.

Note that this widget class must use certain callbacks to communicate with the scrollbar. See the description of the the scrollerClass resource for those details. [*forward_scroller*]

XtNgripClass

Widget class that implements the grip. This class must implement a callback, XtNmoveCallback, that passes a XoMoveCB structure when the user is moving the Grip. When the user is finished, it must execute an XtNcallback again passing an XoMoveCB structure. [*grip_class*]

XtNgripWidget

This is the instance of the widget that allows the user to interactively move the scrollbar up and down. You have two choices, either set the *WidgetClass* at initialization time or create an object with this widget as it's parent and then use *XtSetValues()* to set this resource. The old widget is destroyed.

Note that this widget class must use certain callbacks to communicate with the scrollbar. See the description of the the gripClass resource for those details. [*grip_widget*]

XtNincrement

When the XtNcallback is executed by one of the Scroller widgets, the grip is moved by this amount. [*increment*]

XtNminGrip

The minimum physical size of the Grip in Pixels. Even though the value of *XtNsize* would result in a Grip smaller than this, the grip will not become smaller. This insures the user is able to find the Grip.

If this value is set to 0, it is automatically set to 1 since it is not possible to have a window with a dimension of 0. [*min_grip*]

XtNorientation

Either XoVERTICAL or XoHORIZONTAL. Indicates if the scrollbar is displayed horizontally or vertically.

Actually, this could probably just be determined by the ration of the width to height but we use this instead. [*orientation*]

XtNreverseScroller

This is the instance of the widget that allows the user to interactively move the grip either up or to the left a small increment at a time. To change this you have two choices: either set the *WidgetClass* at initialization time or create a widget with the scrollbar as it's parent and then use *XtSetValues()* to set this resource. The old widget is destroyed. With the second method you must use *XtCreateWidget* and not *XtCreateManagedWidget* to create the scroller.

Note that this widget class must use certain callbacks to communicate with the scrollbar. See the description of the the scrollerClass resource for those details. [*reverse_scroller*]

XtNscrollerClass

Widget class that implments the Scroller buttons. At both ends of the scrollbar a widget is placed that allows the user to move the Grip forward or backward (down/up if the orientation is vertical, right/left is the orientation is horizontal) a fixed increment at a time.

This widget class must use a XtNcallback to communicate with the scrollbar. The data from the the callback is ignored.

There are two ways that Scroller widgets can be created: either the scrollbar creates them at initialization time or something external (the application or another widget) creates them and uses XtSetValues() to notify the scrollbar about the new scrollers. If the scrollbar creates the Scroller widgets, it passes in the value of left, right, top, bottom class as the value XtNdrawClass when creating the widget. If the Scroller is created outside the scrollbar, it is the responsibilty of that external code to set this appropriately.

Whichever way is used to create the Scroller widgets, a XtNcallback callback is added to the scroller. Each time the callback is executed the grip is moved forward or backward by *XtNincrement* percent. It is up to the external code to insure that *XtNincrement* is set to a reasonable value. Ideally, this value should change so that the data being scrolled always moves a constant number of pixels. [*scroller_class*]

XtNscrollerDraw
: The widget class passed to the forward and reverse scroller as the type to draw. On of XoLEFT, XoRIGHT, XoUP, XoDOWN is passed as the value of XtNdirection depending on the orientation of the scrollbar and whether it is a reverse or forward scroller. [*scroller_draw*]

XtNsize
: The size of the grip widget expressed as a percentage of the size of the scrollbar. This should indicate how much of the possible area is currently being viewed. Changing this causes the Grip to resize but it the top remains at *XtNcenter*.

  The actual physical size of the Grip is never allowed to be less then the value of *XtNminGrip* pixels or 1, whichever is larger. [*size*]

XtNtop
: The location of the grip expressed as a percentage of the scrollbar size. When the Grip is interactively moved by the user, this value is updated before the application is notified of the Grip's new position. Similarly, when this value is changed using XtSetValues(), the callbacks are executed to notify the application of the change.

  If changing the top position of the grip ever causes the grip to be not fully visible, the value is adjusted to keep the grip fully visible. [*top*]

Core.height
: No documentation.

Core.width
: No documentation.

direction
: Flag used to keep track of which side of the Grip the mouse was previously on. This is used so we know which way to move the grip when the mouse is over the grip.

orig_height The original height of the scrollbar. This is used when querying the widget for the preferred size. This is returned as the goal height.

Using XtSetValues() to change the height updates this value to the new dimension.

orig_width The original width of the scrollbar. This is used when querying the widget for the preferred size. This is returned as the goal width.

Using XtSetValues() to change the width updates this value to the new dimension.

The following actions are defined by this widget.

GripGoto Move the Grip to the location of the event. The XtNcallbacks are executed, just as for any other action that causes the grip to move.

The following translations are used by this widgets.

<Btn1Down> GripGoto()

Make the Grip jump to the initial mouse click position.

<Btn1Motion> GripGoto()

Moves the grip so it follows the mouse

<Btn1Up> GripGoto()

And when the mouse is released, make sure we are at that position.

```
typedef struct _XoScrollbarRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
```

```
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetClass     border_class;       /* Widget Class of border object */
        Widget          border_widget;      /* draws the border */
        Pixel           fg;                 /* foreground color */
        Cursor          cursor;             /* cursor pattern */
        int             backing_store;      /* backing store */
        Pixmap          insensitive_stipple; /* stipple when insensitive */
        XtCallbackList  inherit_set;        /* to set inherited values */
        Boolean         traverse_accept;    /* if accept focus */
    } XoSimplePart,             simple;
    typedef struct {
        Boolean         direction;          /* True if forward. */
        WidgetClass     grip_class;         /* widget class of Grip */
        Dimension       orig_width;         /* preferred width */
        Dimension       orig_height;        /* preferred height */
        Widget          grip_widget;        /* draws the grip */
        WidgetClass     scroller_class;     /* class for scrollers */
        WidgetClass     scroller_draw;      /* what scroller displays */
        Widget          reverse_scroller;   /* causes reverse scrolling */
        Widget          forward_scroller;   /* causes forward scrolling */
        double          increment;          /* amount to move grip */
        double          size;               /* size of Grip, as percent */
        Dimension       min_grip;           /* minimum size of Grip */
        double          top;                /* location of Grip's top */
        XoOrientation   orientation;        /* vertical or horizontal */
        XtCallbackList  callbacks;          /* executed on grip movement */
    } XoScrollbarPart,          scrollbar;
} XoScrollbarRec,               * XoScrollbarWidget;
```

4.6 MenuButton Widget Class

Should be used in a menu to allow an action to be executed. Understands about hierarchical menus.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/MenuButton.h>
extern WidgetClass xoMenuButtonWidgetClass;
```

The following resources are provided by the *MenuButton* widget:

MenuButton			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Simple (see page 15)			
backingStore	BackingStore	int	"NotUseful"
borderClass	Class	WidgetClass	"None"
borderWidget	BorderWidget	Widget	NULL
cursor	Cursor	Cursor	"cross"
foreground	Foreground	Pixel	XtDefaultForeground

continued

continued			
Name	**Class**	**Type**	**Default**
inheritSet	InheritSet	XtCallbackList	NULL
insensitiveStipple	InsensitiveStipple	Pixmap	NULL
traverseAccept	TraverseAccept	Boolean	(XtPointer) True
Label (see page 19)			
drawClass	Class	WidgetClass	"ObjText"
drawWidget	DrawWidget	Widget	NULL
gravity	Gravity	XoGravity	"Center"
resizeDraw	ResizeDraw	Boolean	False
Button (see page 22)			
borderWidth	BorderWidth	Dimension	0*
borderClass	Class	WidgetClass	"DefaultBorder"*
callback	Callback	XtCallbackList	NULL
repeatDelay	RepeatDelay	int	(XtPointer) 300
repeatInterval	RepeatInterval	int	(XtPointer) 100
MenuButton (see page 35)			
moveDelta	Delta	int	(caddr_t) 5
subMenu	SubMenu	Widget	(caddr_t) NULL

XtNmoveDelta
The amount of horizontal pointer motion necessary before displaying any associated walking menu. [*move_delta*]

XtNsubMenu
A menu widget to implement a submenu off of this widgets. The parent of this widget should be a subclass Shell. When the WalkMenu action executes, it calls the newPopup callbacks on this submenu widget. [*submenu*]

draw_border Set by Highlight() and cleared by Unhighlight(). Tells the expose procedure if the border should be drawn.

start_x The initial x coordinate for determining when any submenu should be displayed. This is set when the widget receieves the focus.

The following actions are defined by this widget.

WalkMenu Popup the one and only popup child of this widget. The widget is popped up spring loaded. It is an error if there is more than one popup child. If there are no popup children, this is a no-op.

The horizontal motion must be >= *move_delta* before the walking menu is moved up. Setting *moveDelta* to 0 causes the menu to be popped up as soon as mouse moves in the widget.

If the value of *moveDelta* is negative, the pointer must move to the left by that amount. If *moveDelta* is positive, the pointer must move right by that amount.

Highlight If this widget does not have a submenu or it has a submenu which is not popped up, then it executes the *newPopup* callback of the parent with the *call_data* set to *NULL*. This causes the parent to popdown any currently displayed menu but only if that menu is from another *menuButton*.

The *x* coordinate of the event is then used to determine when the user has moved enough to activate the menu. See *moveDelta* for the amount and the *WalkMenu()* action that implements this.

The *draw_border* flag is set to true and the widget is redrawn. This should cause a the border to become active and instead of being drawn flat, a raised appearance (assuming a 3d border object is used) should be used for the borders.

Notify If this widget has a submenu, it executes the *newPopup* callback of the parent with the *call_data* set to NULL. This should popdown any submenus currently displayed.

It then executes the *callback* callbacks with the *call_data* set to *True*.

Note that it is possible to execute MenuButtons that have submenus. This should probably be an optional feature.

Unhighlight Sets the *draw_border* flag to *False* and and causes the widget to redisplay itself. The widget should then have a "flat" apperance to indicate it is inactive

ItemNext If this menubutton has a submenu, pop that menu up. If there is not a submenu, then remain here.

ItemPrev If this menubutton has a submenu, pop that menu up. If there is not a submenu, then remain here.

The following translations are used by this widgets.

<FocusIn> Set() Highlight()

Provides a visual feedback that the current input focus is set to this widget. If there are any other submenus displayed, they should be popped down at this point.

<FocusOut> Unhighlight() Unset()

Undoes the visual effects from Highlight(). The display of submenus is not changed.

<Enter> Highlight() Set()

Same as FocusIn

<Leave> Unhighlight() Unset()

Same as FocusOut

<Btn1Down> Unhighlight()

Display any associated submenu if one exists. This is an experimental action to support MenuBars.

<Btn1Up> Notify() Unhighlight() Unset()

Executes the callbacks associated with *callback* and pop's down any other submenus.

<Btn1Motion> WalkMenu()

Causes a submenu to be displayed when *moveDelta* is exceeded.

<Key>Return Notify() Unhighlight() Unset()

!<Key>Up ItemUp()

!<Key>Down ItemDown()

!<Key>Left ItemPrev()

!<Key>Right ItemNext()

!<Key>Home LineBegin()

!<Key>Begin LineBegin()

!<Key>End LineEnd()

```
typedef struct _XoMenuButtonRec {
    typedef struct {
        Widget          self;              /* pointer to widget itself */
        WidgetClass     widget_class;      /* pointer to Widget's ClassRec */
        Widget          parent;            /* parent widget */
        XrmName         xrm_name;          /* widget resource name quarkified */
        Boolean         being_destroyed;   /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
        XtPointer       constraints;       /* constraint record */
        Position        x;                 /* window position */
        Position        y;                 /* window position */
        Dimension       width;             /* window dimensions */
        Dimension       height;            /* window dimensions */
        Dimension       border_width;      /* window border width */
        Boolean         managed;           /* is widget geometry managed? */
        Boolean         sensitive;         /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;       /* private to event dispatcher */
        XtTMRec         tm;                /* translation management */
        XtTranslations  accelerators;      /* accelerator translations */
        Pixel           border_pixel;      /* window border pixel */
        Pixmap          border_pixmap;     /* window border pixmap or NULL */
```

```
    WidgetList      popup_list;         /* list of popups */
    Cardinal        num_popups;         /* how many popups */
    String          name;               /* widget resource name */
    Screen *        screen;             /* window's screen */
    Colormap        colormap;           /* colormap */
    Window          window;             /* window ID */
    Cardinal        depth;              /* number of planes in window */
    Pixel           background_pixel;   /* window background pixel */
    Pixmap          background_pixmap;  /* window background pixmap or NULL */
    Boolean         visible;            /* is window mapped and not occluded? */
    Boolean         mapped_when_managed; /* map window if it's managed? */
} CorePart,                 core;
typedef struct {
    WidgetClass     border_class;       /* Widget Class of border object */
    Widget          border_widget;      /* draws the border */
    Pixel           fg;                 /* foreground color */
    Cursor          cursor;             /* cursor pattern */
    int             backing_store;      /* backing store */
    Pixmap          insensitive_stipple; /* stipple when insensitive */
    XtCallbackList  inherit_set;        /* to set inherited values */
    Boolean         traverse_accept;    /* if accept focus */
} XoSimplePart,             simple;
typedef struct {
    Boolean         resize_draw;        /* if resize draw widget */
    XoGravity       gravity;            /* how to place the label */
    WidgetClass     draw_class;         /* Widget class that draws */
    Widget          draw_widget;        /* Widget for drawing text. */
} XoLabelPart,              label;
typedef struct {
    XtCallbackList  callbacks;          /* executed when activated */
    int             repeat_interval;    /* milli. sec. between executions */
    int             repeat_delay;       /* ms delay before executions */
    int             set;                /* current state of the button */
    XtIntervalId    timer_id;           /* current timer for repeating */
    Boolean         timer_ran;          /* indicates if a timer was actived */
    Boolean         temp;               /* temporary holder for set */
} XoButtonPart,             button;
typedef struct {
    int             move_delta;         /* delta for popups */
    Widget          submenu;            /* submenu to popup */
    Position        start_x;            /* initial x location */
    Boolean         draw_border;        /* if draw the border */
} XoMenuButtonPart,         menu_button;
} XoMenuButtonRec,          * XoMenuButtonWidget;
```

4.7 MenuBarButton Widget Class

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/MenuBarB.h>
extern WidgetClass xoMenuBarButtonWidgetClass;
```

The following resources are provided by the *MenuBarButton* widget:

MenuBarButton			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Simple (see page 15)			
backingStore	BackingStore	int	"NotUseful"
borderClass	Class	WidgetClass	"None"
borderWidget	BorderWidget	Widget	NULL
cursor	Cursor	Cursor	"cross"
foreground	Foreground	Pixel	XtDefaultForeground
inheritSet	InheritSet	XtCallbackList	NULL
insensitiveStipple	InsensitiveStipple	Pixmap	NULL
traverseAccept	TraverseAccept	Boolean	(XtPointer) True

continued

continued			
Name	**Class**	**Type**	**Default**
Label (see page 19)			
drawClass	Class	WidgetClass	"ObjText"
drawWidget	DrawWidget	Widget	NULL
gravity	Gravity	XoGravity	"Center"
resizeDraw	ResizeDraw	Boolean	False
Button (see page 22)			
borderWidth	BorderWidth	Dimension	0*
borderClass	Class	WidgetClass	"DefaultBorder"*
callback	Callback	XtCallbackList	NULL
repeatDelay	RepeatDelay	int	(XtPointer) 300
repeatInterval	RepeatInterval	int	(XtPointer) 100
MenuButton (see page 35)			
moveDelta	Delta	int	(caddr_t) 5
subMenu	SubMenu	Widget	(caddr_t) NULL
MenuBarButton (see page 40)			
moveDelta	Delta	int	0*
enablePopup	EnablePopup	Boolean	False

XtNenablePopup
: True if it's ok to execute the new popup on the parent. This is used as a flag between the parent menubar and the other widgets to indicate when they can actually popup on highlight events. [*enable_popup*]

MenuButton.move_delta
: No documentation.

The following actions are defined by this widget.

WalkMenu Popup the one and only popup child of this widget. The widget is popped up spring loaded. It is an error if there is more than one popup child. If there are no popup children, this is a no-op. The enable_popup flag must be True, otherwise this is a no-op.

Highlight If this widget does not have a submenu or it has a submenu which is not popped up, then it executes the *newPopup* callback of the parent with the *call_data* set to *NULL*. This causes the parent to popdown any currently displayed menu but only if that menu is from another *menuButton*.

The *draw_border* flag is set to true and the widget is redrawn. This should cause the border to become active and instead of being drawn flat. A raised appearance (assuming a 3d border object is used) should be used for the borders.

Unhighlight Sets the *draw_border* flag to *False* and and causes the widget to redisplay itself. The widget should then have a "flat" apperance to indicate it is inactive

SubmenuPopup
Display the submenu. This is an experimental action to be used for MenuButtons that are a child of a MenuBar widget.

GrabFocus Changes the keyboard focus to this widget, but only if traverseAccept resource is True.

EnableMenus
Calls XoEnablePopups() on our parent so that menus can be popped up.

ItemUp Tells parent to move the location cursor up an item. "Up" is defined by the parent. Behaves the same as ItemNext except it does wraps around to the last child of the parent when the first child is reached.

ItemDown Tells the parent to move the location cursor down an item. "Down" is defined by the parent. Behaves the same as ItemNext except it does wraps around to the first child of the parent when the last child is reached.

MenuRight Move right to the next menu bar button. If this menu is popped up, then the next menu bar button's menu is popped up. If this menu is not popped up, then neither is the next one. If at the right hand edge we wrap to the next line. If there is only one line, then it is back to the beginning.

MenuLeft Move left to the next menu bar button. If this menu is popped up, then the next menu bar button's menu is popped up. If this menu is not popped up, then neither is the next one. If at the left hand edge we wrap to the previous line. If there is only one line, then it is back to the beginning.

DataBegin Tells the parent to move the location cursor to the first item. "First" is determined by the parent.

DataEnd Tells the parent to move the locatin cursor to the last item. "Last" is determined by the parent.

LineBegin Tells the parent to move the location cursor to the leftmost edge. The leftmost edge is defined by the parent.

LineEnd Tells the parent to move the location cursor to the rightmost edge. The rightmost edge is defined by the parent.

The following translations are used by this widgets.

<Btn1Motion> WalkMenu()

Causes a submenu to be displayed when *moveDelta* is exceeded.

<FocusIn> Highlight()

Provides a visual feedback that the current input focus is set to this widget. If there are any other submenus displayed, they should be popped down at this point.

<FocusOut> Unhighlight()

Undoes the visual effects from Highlight(). The display of submenus is not changed.

<Leave> Unhighlight()

Same as FocusOut

<Btn1Down> GrabFocus() EnableMenus(True) SubmenuPopup()

Display any associated submenu if one exists. This is an experimental action to support MenuBars.

<Btn1Up> Notify() Unhighlight()

Executes the callbacks associated with *callback* and pop's down any other submenus.

!<Key>Up ItemUp()

!<Key>Left MenuLeft()

!<Key>Down ItemDown()

!<Key>Right MenuRight()

!<Key>Home LineBegin()

!<Key>Begin LineBegin()

!<Key>End LineEnd()

```
typedef struct _XoMenuBarButtonRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
```

```
    XtTMRec          tm;                /* translation management */
    XtTranslations   accelerators;      /* accelerator translations */
    Pixel            border_pixel;      /* window border pixel */
    Pixmap           border_pixmap;     /* window border pixmap or NULL */
    WidgetList       popup_list;        /* list of popups */
    Cardinal         num_popups;        /* how many popups */
    String           name;              /* widget resource name */
    Screen *         screen;            /* window's screen */
    Colormap         colormap;          /* colormap */
    Window           window;            /* window ID */
    Cardinal         depth;             /* number of planes in window */
    Pixel            background_pixel;  /* window background pixel */
    Pixmap           background_pixmap; /* window background pixmap or NULL */
    Boolean          visible;           /* is window mapped and not occluded? */
    Boolean          mapped_when_managed; /* map window if it's managed? */
} CorePart,                  core;
typedef struct {
    WidgetClass      border_class;      /* Widget Class of border object */
    Widget           border_widget;     /* draws the border */
    Pixel            fg;                /* foreground color */
    Cursor           cursor;            /* cursor pattern */
    int              backing_store;     /* backing store */
    Pixmap           insensitive_stipple; /* stipple when insensitive */
    XtCallbackList   inherit_set;       /* to set inherited values */
    Boolean          traverse_accept;   /* if accept focus */
} XoSimplePart,              simple;
typedef struct {
    Boolean          resize_draw;       /* if resize draw widget */
    XoGravity        gravity;           /* how to place the label */
    WidgetClass      draw_class;        /* Widget class that draws */
    Widget           draw_widget;       /* Widget for drawing text. */
} XoLabelPart,               label;
typedef struct {
    XtCallbackList   callbacks;         /* executed when activated */
    int              repeat_interval;   /* milli. sec. between executions */
    int              repeat_delay;      /* ms delay before executions */
    int              set;               /* current state of the button */
    XtIntervalId     timer_id;          /* current timer for repeating */
    Boolean          timer_ran;         /* indicates if a timer was actived */
    Boolean          temp;              /* temporary holder for set */
} XoButtonPart,              button;
typedef struct {
    int              move_delta;        /* delta for popups */
    Widget           submenu;           /* submenu to popup */
```

```
        Position        start_x;        /* initial x location */
        Boolean         draw_border;    /* if draw the border */
    } XoMenuButtonPart,                   menu_button;
    typedef struct {
        Boolean         enable_popup;   /* if popup are enabled */
    } XoMenuBarButtonPart,                menu_bar_button;
} XoMenuBarButtonRec,    * XoMenuBarButtonWidget;
```

4.8 Grip Widget Class

Implements a small region the user can "grip" with the mouse to change the size of a window. This is intended to be used by other widgets, such as The class variable is *xoVpaneWidgetClass*.

The The class variable is *xoGripWidgetClass* executes the *moveCallback* callback list as the mouse moves with the left mouse button pressed. It passes as data the initial x and y coordinates and the current x and y coordinates both expressed relative to the the parent of the grip.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Grip.h>
extern WidgetClass xoGripWidgetClass;
```

The following resources are provided by the *Grip* widget:

Grip			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
			continued

continued			
Name	**Class**	**Type**	**Default**
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Simple (see page 15)			
backingStore	BackingStore	int	"NotUseful"
borderClass	Class	WidgetClass	"None"
borderWidget	BorderWidget	Widget	NULL
cursor	Cursor	Cursor	"cross"
foreground	Foreground	Pixel	XtDefaultForeground
inheritSet	InheritSet	XtCallbackList	NULL
insensitiveStipple	InsensitiveStipple	Pixmap	NULL
traverseAccept	TraverseAccept	Boolean	(XtPointer) True
Label (see page 19)			
drawClass	Class	WidgetClass	"ObjText"
drawWidget	DrawWidget	Widget	NULL
gravity	Gravity	XoGravity	"Center"
resizeDraw	ResizeDraw	Boolean	False
Button (see page 22)			
borderWidth	BorderWidth	Dimension	0*
borderClass	Class	WidgetClass	"DefaultBorder"*
callback	Callback	XtCallbackList	NULL
repeatDelay	RepeatDelay	int	(XtPointer) 300
repeatInterval	RepeatInterval	int	(XtPointer) 100
Grip (see page 45)			
borderWidth	BorderWidth	Dimension	0*
height	Height	Dimension	(XtPointer) 10*
width	Width	Dimension	(XtPointer) 10*
borderClass	Class	WidgetClass	"DefaultBorder"*
callback	Callback	XtCallbackList	NULL
moveCallback	MoveCallback	XtCallbackList	NULL

XtNcallback A list of function to executed when the mouse moves with a button down. The *call_data* is packaged as follows:

[*callbacks*]

XtNmoveCallback
A list of function to executed when the mouse moves with a button down. The *call_data* is packaged as follows:

[*move_callbacks*]

Core.border_width
No documentation.

Core.height No documentation.

Core.width No documentation.

Simple.border_class
No documentation.

x_first The initial position movement started from. This is relative to the parent widget. Should be initialized by the Set action on a button down. Passed in an XoMoveCBdata structure for the moveCallback.

y_first The initial position movement started from. This is relative to the parent widget. Should be initialized by the Set action on a button down. Passed in an XoMoveCBdata structure for the moveCallback.

The following actions are defined by this widget.

Set Initializes the starting position as used in MoveNotify.

Unset Indicates the button is no longer selected. This should cause the background and foreground to be displayed normally.

Notify Calls the XtNcallCallbacks to indicate no more motion events are going to happen.

MoveNotify Executes the *moveCallback* and passes the starting position as determined in *Set* and the current position – both are parent relative. Uses the XoMoveCB-data structure as the call_data part of the callback.

The following translations are used by this widgets.

<Btn1Down>	Highlight() Set()
<Btn1Motion>	MoveNotify()
<Btn1Up>	Notify() Unset()
<FocusIn>	Highlight()

Provides a visual feedback that the current input focus is set to this widget. The usual visual feedback is to place a black border around the widget.

<FocusOut>	Unhighlight()

Should undo the visual effects from Highlight().

<Leave>	UnsetTemp()
<Enter>	SetTemp()
<Key>Return	Set() Notify() Unset()
!<Key>Tab	ItemNext()
!Ctrl<Key>Tab	ItemNext()
!Shift<Key>Tab	ItemPrev()
!Shift Ctrl <Key>Tab	ItemPrev()
!<Key>Up	ItemUp()
!<Key>Left	ItemUp()
!<Key>Down	ItemDown()
!<Key>Right	ItemDown()
!<Key>Home	LineBegin()
!<Key>Begin	LineBegin()
!<Key>End	LineEnd()
!Ctrl<Key>Home	DataBegin()
!Ctrl<Key>Begin	DataBegin()
!Ctrl<Key>End	DataEnd()

```
typedef struct _XoGripRec {
    typedef struct {
        Widget          self;              /* pointer to widget itself */
        WidgetClass     widget_class;      /* pointer to Widget's ClassRec */
        Widget          parent;            /* parent widget */
        XrmName         xrm_name;          /* widget resource name quarkified */
        Boolean         being_destroyed;   /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
```

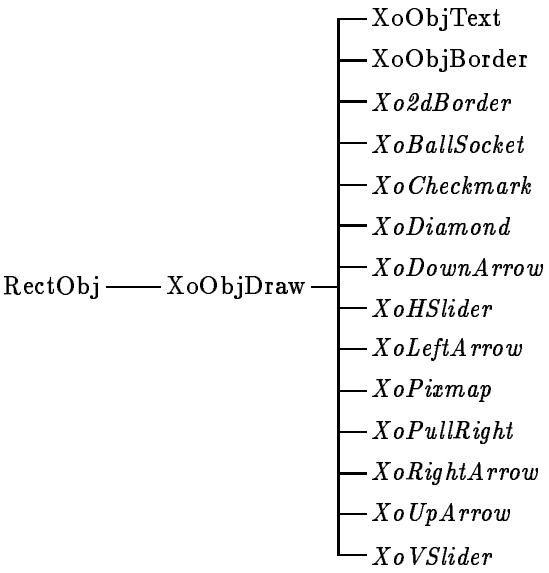
```
    XtPointer       constraints;        /* constraint record */
    Position        x;                  /* window position */
    Position        y;                  /* window position */
    Dimension       width;              /* window dimensions */
    Dimension       height;             /* window dimensions */
    Dimension       border_width;       /* window border width */
    Boolean         managed;            /* is widget geometry managed? */
    Boolean         sensitive;          /* is widget sensitive to user events */
    Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    XtEventTable    event_table;        /* private to event dispatcher */
    XtTMRec         tm;                 /* translation management */
    XtTranslations  accelerators;       /* accelerator translations */
    Pixel           border_pixel;       /* window border pixel */
    Pixmap          border_pixmap;      /* window border pixmap or NULL */
    WidgetList      popup_list;         /* list of popups */
    Cardinal        num_popups;         /* how many popups */
    String          name;               /* widget resource name */
    Screen *        screen;             /* window's screen */
    Colormap        colormap;           /* colormap */
    Window          window;             /* window ID */
    Cardinal        depth;              /* number of planes in window */
    Pixel           background_pixel;   /* window background pixel */
    Pixmap          background_pixmap;  /* window background pixmap or NULL */
    Boolean         visible;            /* is window mapped and not occluded? */
    Boolean         mapped_when_managed; /* map window if it's managed? */
} CorePart,                 core;
typedef struct {
    WidgetClass     border_class;       /* Widget Class of border object */
    Widget          border_widget;      /* draws the border */
    Pixel           fg;                 /* foreground color */
    Cursor          cursor;             /* cursor pattern */
    int             backing_store;      /* backing store */
    Pixmap          insensitive_stipple; /* stipple when insensitive */
    XtCallbackList  inherit_set;        /* to set inherited values */
    Boolean         traverse_accept;    /* if accept focus */
} XoSimplePart,             simple;
typedef struct {
    Boolean         resize_draw;        /* if resize draw widget */
    XoGravity       gravity;            /* how to place the label */
    WidgetClass     draw_class;         /* Widget class that draws */
    Widget          draw_widget;        /* Widget for drawing text. */
} XoLabelPart,              label;
typedef struct {
    XtCallbackList  callbacks;          /* executed when activated */
```

```
        int             repeat_interval;    /* milli. sec. between executions */
        int             repeat_delay;       /* ms delay before executions */
        int             set;                /* current state of the button */
        XtIntervalId    timer_id;           /* current timer for repeating */
        Boolean         timer_ran;          /* indicates if a timer was actived */
        Boolean         temp;               /* temporary holder for set */
    } XoButtonPart,                 button;
    typedef struct {
        Position        x_first;            /* movement started here */
        Position        y_first;            /* movement started here */
        XtCallbackList  callbacks;          /* executed when activated */
        XtCallbackList  move_callbacks;     /* executed when activated */
    } XoGripPart,                   grip;
} XoGripRec,                    * XoGripWidget;
```

Chapter 5

The Output Gadgets

The following diagram illustrates the set of output gadgets intended:



Using gadgets encapsulates the typical drawing used in the toolkit. It allows the rest of the toolkit to implement the layout and input actions needed. Anyone is free to write additional gadgets that can be used interchangeably with any of the others.

An alternative to the above is to wrap all the drawing into a set of function calls. This is possible but forces one to know all the possible calling sequences or to limit those to some subset. By using gadgets, however, one can continue to use the resource database,

initialization, and setting/getting of values.

It also makes it simpler to customize the appearance. The user and application writer are free to substitute different gadgets for the drawing.

5.1 Object Class

This is the parent of all objects used by the Xt Intrinsics. Unfortunately, it was grafted on after the Core widget class was implemented so things to do not exactly follow the usual Xt model of inheritance.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Object.h>
extern WidgetClass objectWidgetClass;
```

The following resources are provided by the *Object* widget:

Object			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL

XtNdestroyCallback
List of callbacks executed when the widget is destroyed. Inherited from Object. [*destroy_callbacks*]

being_destroyed
marked for destroy.

constraints constraint record.

parent parent widget.

self pointer to widget itself.

widget_class pointer to Widget's ClassRec.

xrm_name widget resource name quarkified.

```
typedef struct _ObjectRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
```

```
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
    } ObjectPart,                       object;
} ObjectRec,                        * ObjectWidget;
```

5.2 RectObj Class

The *RectObj* adds x, y, width, height and several display oriented methods to the Object class. Subclasses of this are typically referred to as gadgets. The signficance is that subclasses of RectObj are windowless widgets. This has several ramifications.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/RectObj.h>
extern WidgetClass rectObjWidgetClass;
```

The following resources are provided by the *RectObj* widget:

RectObj			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0

XtNancestorSensitive

Set to False when an ancestor of this widget is no longer accepts input. If this is true, it may be appropriate to visually indicate this widget cannot accept input. Inherited from RectObj. [*ancestor_sensitive*]

XtNborderWidth
The width of the border in pixels. Inherited from RectObj. [*border_width*]

XtNheight The height of this widget in pixels. Inherited from RectObj. [*height*]

XtNsensitive
Indicates if this widget accepts any input. Set by calls to XtSetSensitive(). Inherited from RectObj. [*sensitive*]

XtNwidth The width of this widget in pixels. Inherited from RectObj. [*width*]

XtNx The x coordinate of this widget relative to the parent. Inherited from RectObj. [*x*]

XtNy The y coordinate of this widget relative to the parent. Inherited from RectObj. [*y*]

managed is widget geometry managed?.

```
typedef struct _RectObjRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
    } ObjectPart,                       object;
    typedef struct {
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                      rectangle;
} RectObjRec,                       * RectObjWidget;
```

5.3 ObjDraw Class

Provides a basic framework for all subclasses of RectObj in the Xo widget set. Adds resources for foreground color, insensitive foreground color, a flag to indicate if the object

should not do any drawing and finally a list of resources that should be inherited by from the parent widget.

Such are the common features to all objects.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/ObjDraw.h>
extern WidgetClass xoObjDrawWidgetClass;
```

The following resources are provided by the *ObjDraw* widget:

ObjDraw			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
ObjDraw (see page 54)			
foreground	Foreground	Pixel	XtDefaultForeground
insensitiveColor	Foreground	Pixel	"gray90"
inheritResources	QuarkSortedList	XrmQuark *	NULL
noDraw	DoDraw	Boolean	"False"

XtNforeground
: The color for drawing primary objects such as text and lines. Changing this during execution causes the object to change color. [*fg*]

XtNinsensitiveColor
: The color for drawing primary objects when they are no longer sensitive. [*fg_insensitive*]

XtNinheritResources
: The list of resources that this widget should inherit from it's parent. [*inherit_resources*]

XtNnoDraw Use this flag to temporarily turn off drawing of the object. Actually, I'm not sure why not just destroy the object, but anyway. [*no_draw*]

```
typedef struct _XoObjDrawRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
    } ObjectPart,                   object;
    typedef struct {
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                  rectangle;
    typedef struct {
        Pixel           fg;                 /* foreground color */
        Pixel           fg_insensitive;     /* insensitive color */
        Boolean         no_draw;            /* should it not draw */
        XrmQuark *      inherit_resources;  /* resources to inherit */
    } XoObjDrawPart,                obj_draw;
} XoObjDrawRec,                 * XoObjDrawWidget;
```

5.4 ObjArrow Class

Draws an isoceles triangle.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/ObjArrow.h>
extern WidgetClass xoObjArrowWidgetClass;
```

The following resources are provided by the *ObjArrow* widget:

ObjArrow			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
ObjDraw (see page 54)			
foreground	Foreground	Pixel	XtDefaultForeground
insensitiveColor	Foreground	Pixel	"gray90"
inheritResources	QuarkSortedList	XrmQuark *	NULL
noDraw	DoDraw	Boolean	"False"
ObjArrow (see page 56)			
inheritResources	QuarkSortedList	XrmQuark *	"foreground, background, sensitive, ancestorSensitive, insensitiveColor, direction"*
direction	Direction	XoPointing	"Left"

XtNdirection
The direction the arrow points – the line drawn from the base, the longest dimension, to the apex goes in this direction. [*direction*]

ObjDraw.inherit_resources
No documentation.

gc This GC has the foreground color set to *foreground* and the line width set to border_width. It is used for the XDrawLine() of the triangle.

```
typedef struct _XoObjArrowRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
        Widget          parent;         /* parent widget */
        XrmName         xrm_name;       /* widget resource name quarkified */
        Boolean         being_destroyed; /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
        XtPointer       constraints;    /* constraint record */
    } ObjectPart,                   object;
    typedef struct {
```

```
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                  rectangle;
    typedef struct {
        Pixel           fg;                 /* foreground color */
        Pixel           fg_insensitive;     /* insensitive color */
        Boolean         no_draw;            /* should it not draw */
        XrmQuark *      inherit_resources;  /* resources to inherit */
    } XoObjDrawPart,                obj_draw;
    typedef struct {
        XoPointing      direction;          /* which way arrow points */
        GC              gc;                 /* GC for drawing the border */
    } XoObjArrowPart,               obj_arrow;
} XoObjArrowRec,                    * XoObjArrowWidget;
```

5.5 ObjPixmap Class

Displays a pixmap. Currently, this is unimplemented. It should, however, eventualy use XPM3 to read bitmaps and color pixmaps. These will be scalabel to fit the current size.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/ObjPixmap.h>
extern WidgetClass xoObjPixmapWidgetClass;
```

The following resources are provided by the *ObjPixmap* widget:

ObjPixmap			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
continued			

continued			
Name	**Class**	**Type**	**Default**
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
ObjDraw (see page 54)			
foreground	Foreground	Pixel	XtDefaultForeground
insensitiveColor	Foreground	Pixel	"gray90"
inheritResources	QuarkSortedList	XrmQuark *	NULL
noDraw	DoDraw	Boolean	"False"
ObjPixmap (see page 58)			
pixmap	Pixmap	Pixmap	NULL
scale	Scale	Boolean	False
tile	Tile	Boolean	False

XtNpixmap A picture to display instead of *label*. If this is set, then *label* is not used. The *gravity* resource is used to place this pixmap. In addition, if the *tile* resource is set, then this pixmap is tiled accros the entire widget. However, if *scale* is True, then that takes precedence over tiling and the pixmap is scaled to fit the size of the widget.

We should use XPM2 format for the converters into this data. [*pixmap*]

XtNscale If True, the displayed object (either text or pixmap) should be scaled to fit the size of the widget. For text, the *fontPattern* resource is used to obtain a list of fonts. From that list, one is selected based on avg_x and avg_y to fit the text within the widget. The text is then changed to use this new font.

For Pixmap's, some unknown algorithm is used to scale the pixmap to fit the entire widget. [*scale*]

XtNtile If True and a pixmap is being displayed, then the pixmap is displayed across the entire widget. Tiling starts in the upper left corner regardless of the values of *gravity* and *justify*. [*tile*]

```
typedef struct _XoObjPixmapRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
        Widget          parent;         /* parent widget */
```

```
        XrmName         xrm_name;          /* widget resource name quarkified */
        Boolean         being_destroyed;   /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
        XtPointer       constraints;       /* constraint record */
    } ObjectPart,                    object;
    typedef struct {
        Position        x;                 /* window position */
        Position        y;                 /* window position */
        Dimension       width;             /* window dimensions */
        Dimension       height;            /* window dimensions */
        Dimension       border_width;      /* window border width */
        Boolean         managed;           /* is widget geometry managed? */
        Boolean         sensitive;         /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                   rectangle;
    typedef struct {
        Pixel           fg;                /* foreground color */
        Pixel           fg_insensitive;    /* insensitive color */
        Boolean         no_draw;           /* should it not draw */
        XrmQuark *      inherit_resources; /* resources to inherit */
    } XoObjDrawPart,                 obj_draw;
    typedef struct {
        Boolean         tile;              /* tile pixmap */
        Boolean         scale;             /* scale object to widget size */
        Pixmap          pixmap;            /* picture to draw */
    } XoObjPixmapPart,               obj_pixmap;
} XoObjPixmapRec,                    * XoObjPixmapWidget;
```

5.6 ObjText Class

This provides an object for drawing text. Currently only 8-bit text is supported. Multiple lines may be used by including either an ASCII newline or embedding the string \n in the actual text. The lines of text are either left, right or center justified.

A default width and height can be set with defaultLabel. The maximum width and height of this and the displayed label are returned for any geometry requests. This is useful if a label is going to have many different values but one does not want it constantly resizing.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/ObjText.h>
extern WidgetClass xoObjTextWidgetClass;
```

The following resources are provided by the *ObjText* widget:

ObjText			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
ObjDraw (see page 54)			
foreground	Foreground	Pixel	XtDefaultForeground
insensitiveColor	Foreground	Pixel	"gray90"
inheritResources	QuarkSortedList	XrmQuark *	NULL
noDraw	DoDraw	Boolean	"False"
ObjText (see page 60)			
inheritResources	QuarkSortedList	XrmQuark *	"label, sensitive, ancestorSensitive, defaultLabel,fontFamily,fontStyle,fontSize,rotate"*
defaultLabel	DefaultLabel	String	NULL
fontFamily	FontFamily	String	"times"
fontPattern	FontPattern	String	XtDefaultFont
fontSize	FontSize	int	(XtPointer) AtFontNORMAL
fontStyle	FontStyle	int	(XtPointer) AtFontBOLD
justify	Justify	XoJustify	"Left"
label	Label	String	NULL
rotate	Rotate	Boolean	False

XtNdefaultLabel

Provides the default size of this text box. When the widget is asked to determine it's size, the maximum width and height of this *default_label* and the normal *label* are returned. This is useful if this is going to be set to a number of different values but you only want it to remain a constant size. To do so, just set *default_label* to the largest string. [*default_label*]

XtNfontFamily

The family of fonts this text should use. Based on font size and font style a particular font is displayed. [*font_family*]

XtNfontPattern
Pattern to use to search for an appropriate font. Value should be specific enough to select the correct family (such as courier) and weight of the font (such as bold) but general enough that all possible point sizes are available (don't specify point size or pixel height and width).

A library function is needed to allow some caching and a centralized mechanism to determine the correct font size. [*font_pattern*]

XtNfontSize The size of the font to use for this label, in points. [*font_size*]

XtNfontStyle
If this text is bold, italic, normal etc. by default [*font_style*]

XtNjustify Controls how multiple lines of text are layed out with respect to each other. See *gravity* to place the block of lines within the widget. Possible values are Left, Center, and Right. Left justify causes each line of text to start at the same x coordinate. Right justify causes each line of text to finish at the same x coordinate. Center justify causes each line of text to be centered with respect to the widest line of of text. [*justify*]

XtNlabel The text to display using *foreground* as the color and *font* as the typeface for the string. If the string contains newlines (ascii character 10) or the two character sequence \n, the text occupies multiple lines. Each text line is then justified according to *justify* relative to the other text lines. [*label*]

XtNrotate Indicates if text should be rotated 90 degrees counter clockwise [*rotate*]

ObjDraw.inherit_resources
No documentation.

default_height The default calculated height.

default_lines An array of lines of text. This is used to determine the default size of this text.

default_width The default calculated width.

family The actual family to use when doing any drawing. The *font_family* is converted to this type.

gc This gc is created to display the text. The font and foreground colors are set in GetGC().

lines An array of lines of text. Each entry has a copy of the corresponding string, width, height, x offset (used for justification) and y offset (used to establish the baseline).

```
typedef struct _XoObjTextRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
    } ObjectPart,                   object;
    typedef struct {
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                  rectangle;
    typedef struct {
        Pixel           fg;                 /* foreground color */
        Pixel           fg_insensitive;     /* insensitive color */
        Boolean         no_draw;            /* should it not draw */
        XrmQuark *      inherit_resources;  /* resources to inherit */
    } XoObjDrawPart,                obj_draw;
    typedef struct {
        String          font_family;        /* family of fonts */
        AtFontFamily *  family;             /* the family */
        int             font_size;          /* size of the label */
        int             font_style;         /* bold, italic, etc */
        Boolean         rotate;             /* if rotate 90 degrees */
        XoJustify       justify;            /* justifcation for muli-lines */
        String          label;              /* the label string */
        String          font_pattern;       /* pattern for scaled fonts */
        GC              gc;                 /* for drawing the text */
        _XoTextInfo *   lines;              /* lines of text */
        String          default_label;      /* default label */
        _XoTextInfo *   default_lines;      /* default text for size */
        Dimension       default_width;      /* normal width */
        Dimension       default_height;     /* normal height */
    } XoObjTextPart,                obj_text;
} XoObjTextRec,                 * XoObjTextWidget;
```

5.7 ObjBorder Class

There are several possible border types such as simple rectangles, beveled edges, oval shapes and drop shadows. This widget class provides the basic features that all the above inherit from. This class should not be instantiated since it has no real implementation.

All border objects support clearInverted and inverted resources. Inverted should be True to indicate this is currently selected. clearInverted is set to indicate when inverted is False a very mild, preferably no, border is displayed. ClearInverted is mostly usueful for menus.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/ObjBorder.h>
extern WidgetClass xoObjBorderWidgetClass;
```

The following resources are provided by the *ObjBorder* widget:

ObjBorder			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
ObjDraw (see page 54)			
foreground	Foreground	Pixel	XtDefaultForeground
insensitiveColor	Foreground	Pixel	"gray90"
inheritResources	QuarkSortedList	XrmQuark *	NULL
noDraw	DoDraw	Boolean	"False"
ObjBorder (see page 64)			
clearInverted	ClearInverted	Boolean	False
inverted	Inverted	Boolean	False

XtNclearInverted

Flag that indicates the inverted state should be shown by not drawing a border. This is useful for menus and menubars that have a better visual appearance when the highlighted state is very distinct from the unhighlighted state.

When the inverted value changes from True to False, this causes an XClearArea() to be executed with no expose events generated. [*clear_inverted*]

XtNinverted This flag should be set when the border should be drawn with the shadow and highlight edges reversed. This makes the object look like it has been pressed. [*inverted*]

```
typedef struct _XoObjBorderRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
    } ObjectPart,                   object;
    typedef struct {
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                  rectangle;
    typedef struct {
        Pixel           fg;                 /* foreground color */
        Pixel           fg_insensitive;     /* insensitive color */
        Boolean         no_draw;            /* should it not draw */
        XrmQuark *      inherit_resources;  /* resources to inherit */
    } XoObjDrawPart,                obj_draw;
    typedef struct {
        Boolean         clear_inverted;     /* clear when inverted */
        Boolean         inverted;           /* draw edges depressed */
    } XoObjBorderPart,              obj_border;
} XoObjBorderRec,           * XoObjBorderWidget;
```

5.8 Obj2dBorder Class

Provides a simple square border. This is particularly suitable for monochrome displays since a 3 dimension border does not look nearly as good.

When the inverted resource is set to true, the border width is increased by partWidth (for partial width)

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Obj2dB.h>
extern WidgetClass xoObj2dBorderWidgetClass;
```

The following resources are provided by the *Obj2dBorder* widget:

Obj2dBorder			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
ObjDraw (see page 54)			
foreground	Foreground	Pixel	XtDefaultForeground
insensitiveColor	Foreground	Pixel	"gray90"
inheritResources	QuarkSortedList	XrmQuark *	NULL
noDraw	DoDraw	Boolean	"False"
ObjBorder (see page 64)			
clearInverted	ClearInverted	Boolean	False
inverted	Inverted	Boolean	False
Obj2dBorder (see page 65)			
inheritResources	QuarkSortedList	XrmQuark *	"foreground, background, sensitive, ancestorSensitive, clearInverted, insensitiveColor"*
borderWidth	BorderWidth	Dimension	(XtPointer) 2*
partWidth	PartWidth	int	(caddr_t) 2

XtNpartWidth

To get the normal drawing border width, divide the border_width by this value. If the result is zero, do not draw a border [*part_width*]

ObjDraw.inherit_resources
No documentation.

RectObj.border_width
No documentation.

gc This GC has the foreground color set to *foreground* and the line width set to either border_width or border_width/part_width. It is used for the XDrawLine() of the border.

```
typedef struct _XoObj2dBorderRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
    } ObjectPart,                       object;
    typedef struct {
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                      rectangle;
    typedef struct {
        Pixel           fg;                 /* foreground color */
        Pixel           fg_insensitive;     /* insensitive color */
        Boolean         no_draw;            /* should it not draw */
        XrmQuark *      inherit_resources;  /* resources to inherit */
    } XoObjDrawPart,                    obj_draw;
    typedef struct {
        Boolean         clear_inverted;     /* clear when inverted */
        Boolean         inverted;           /* draw edges depressed */
    } XoObjBorderPart,                  obj_border;
    typedef struct {
        int             part_width;         /* divisor of border_width */
        GC              gc;                 /* GC for drawing the border */
    } XoObj2dBorderPart,                obj_2d_border;
} XoObj2dBorderRec,         * XoObj2dBorderWidget;
```

5.9 Obj3dBorder Class

Provides a 3 dimensional appearance. This is looks like the Motif 3d borders. The edge are called beveled, much as if a chisel had cut the area out. The colors for the highlight bevel and shadow bevels can be set.

When the border is inverted, the colors of the highlight and beveled edges are exchanged. This causes a widget to appear to have been pushed down.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Obj3dB.h>
extern WidgetClass xoObj3dBorderWidgetClass;
```

The following resources are provided by the *Obj3dBorder* widget:

Obj3dBorder			
Name	**Class**	**Type**	**Default**
Object (see page 52)			
destroyCallback	Callback	XtCallbackList	NULL
RectObj (see page 53)			
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
borderWidth	BorderWidth	Dimension	0
height	Height	Dimension	0
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
ObjDraw (see page 54)			
foreground	Foreground	Pixel	XtDefaultForeground
insensitiveColor	Foreground	Pixel	"gray90"
inheritResources	QuarkSortedList	XrmQuark *	NULL
noDraw	DoDraw	Boolean	"False"
ObjBorder (see page 64)			
clearInverted	ClearInverted	Boolean	False
inverted	Inverted	Boolean	False
Obj3dBorder (see page 68)			
inheritResources	QuarkSortedList	XrmQuark *	"foreground, bevelShadow, bevelHighlight, sensitive, ancestorSensitive, clearInverted, insensitiveColor, background"*

continued

continued			
Name	**Class**	**Type**	**Default**
borderWidth	BorderWidth	Dimension	(XtPointer) 2*
bevelHighlight	BevelHighlight	Pixel	"gray85"
bevelShadow	BevelShadow	Pixel	"gray65"
defBackground	Background	Pixel	"grey80"

XtNbevelHighlight
Color for drawing the lighter (upper and left) edges of the border. Should be the same hue as the background but a higher intensity. Note that all four colors, *foreground*, *background*, *bevelHighlight*, and *bevelShadow* must be chosen in concert to produce an appealing appearance. The *borderWidth* determines the width of the bevels. [*bevel_highlight*]

XtNbevelShadow
Color for drawing the darker (lower and right) edges of the border. Should be the same hue as the background but a lower intensity. Note that all four colors, *foreground*, *background*, *bevelHighlight*, and *bevelShadow* must be chosen in concert to produce an appealing appearance.[1] [*bevel_shadow*]

XtNdefBackground
The default color for drawing the background. The background of the parent widget is initially set to this value when the 3d border object is created (irrespective of what the parent's background color is supposed to be). Thereafter, this follows the value of the parents background. [*def_background*]

ObjDraw.inherit_resources
No documentation.

RectObj.border_width
No documentation.

highlight_gc This GC has the foreground color set to *bevelHighlight*. It is used for the XFillPolygon() of the upper and left edges.

shadow_gc This GC has the foreground color set to *bevelShadow*. It is used for the XFillPolygon() of the lower and right edges.

```
typedef struct _XoObj3dBorderRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
        Widget          parent;         /* parent widget */
```

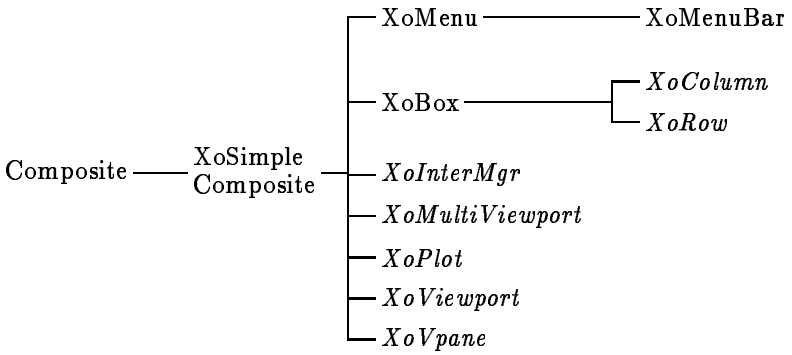
[1] We should provide an utility to help select such values. Does an hsv model work well for makeing such choices?

```
        XrmName         xrm_name;          /* widget resource name quarkified */
        Boolean         being_destroyed;   /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
        XtPointer       constraints;       /* constraint record */
    } ObjectPart,                    object;
    typedef struct {
        Position        x;                 /* window position */
        Position        y;                 /* window position */
        Dimension       width;             /* window dimensions */
        Dimension       height;            /* window dimensions */
        Dimension       border_width;      /* window border width */
        Boolean         managed;           /* is widget geometry managed? */
        Boolean         sensitive;         /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    } RectObjPart,                   rectangle;
    typedef struct {
        Pixel           fg;                /* foreground color */
        Pixel           fg_insensitive;    /* insensitive color */
        Boolean         no_draw;           /* should it not draw */
        XrmQuark *      inherit_resources; /* resources to inherit */
    } XoObjDrawPart,                 obj_draw;
    typedef struct {
        Boolean         clear_inverted;    /* clear when inverted */
        Boolean         inverted;          /* draw edges depressed */
    } XoObjBorderPart,               obj_border;
    typedef struct {
        Pixel           def_background;    /* background color */
        Pixel           bevel_highlight;   /* Color for highlighted edge */
        Pixel           bevel_shadow;      /* Color for shadowed edge */
        GC              highlight_gc;      /* GC for upper edge */
        GC              shadow_gc;         /* GC for lower edge */
    } XoObj3dBorderPart,             obj_3d_border;
} XoObj3dBorderRec,          * XoObj3dBorderWidget;
```

Chapter 6

The Manager Widgets

The following diagram illustrates the set of manager widgets intended:



6.1 Composite Widget Class

This class allows other widgets to be it's children.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Composite.h>
extern WidgetClass compositeWidgetClass;
```

The following resources are provided by the *Composite* widget:

Composite			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL

XtNchildren A read only list of children of the widget. [*children*]

XtNinsertPosition

Points to a function that should return the insertion order for a widget. The declaration is

```
Cardinal OrderProc (Widget gw);
```

it should return a number indicating how many children should go before the widget. Return zero indicates that the widget should go before all other children, and returning *num_children* indicates that it should go after all other children. The default *insertPosition* returns *num_children*. [*insert_position*]

XtNnumChildren

The number of children in *children*. [*num_children*]

num_slots The amount of space allocated for *children*.

```
typedef struct _CompositeRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList      children;           /* array of ALL widget children */
        Cardinal        num_children;       /* total number of widget children */
        Cardinal        num_slots;          /* number of slots in children array */
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                    composite;
} CompositeRec,             * CompositeWidget;
```

6.2 CompBase Widget Class

This class should be used to add features such as printing, etc to all composite widgets in this set.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/CompBase.h>
extern WidgetClass xoCompBaseWidgetClass;
```

The following resources are provided by the *CompBase* widget:

CompBase			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
CompBase (see page 75)			
borderWidth	BorderWidth	Dimension	0*

continued

continued			
Name	**Class**	**Type**	**Default**
doLayout	DoLayout	Boolean	(caddr_t) True
traverse	Traverse	XtCallbackList	NULL

XtNdoLayout

If this flag is set, then the widget should readjust the children whenever changes occur. If the flag is False, the widget should ignore changes in the set of manage children and resize events until the flag is set to True. When the flag changes from False (do not reaarange the children) to True, the complete layout is recalculated.

This flag makes it conveniant to create or manage/unmanage a group of widgets one at a time without incurring the overhead of constantly rearranging the widgets. [*do_layout*]

XtNtraverse List of callbacks executed to accomplish keyboard traversals. There should only be one callback on this list. Behaviour is undefined if there is more than one callback. If the application wishes to change the callback executed, it should delete the callback list and then install it's own private handler.

At initialization time, the Initialize method executes XtAddCallback() for this callback list with the value of the call method *traverse* as the callback to execute. To replace the default traversal method, subclasses should override this value.

The arguments passed in *call_data* is a structure of type XoTraversalInfo defined as follows:

```
typedef enum
{
        XoTRAVERSE_PREVIOUS, XoTRAVERSE_NEXT, XoTRAVERSE_HOME,
        XoTRAVERSE_END,
} XoDirection;

/*
 * This information is passed to the traverse callback.
 */
typedef struct _xo_traverse_info
{
        Widget          t_start;        /* widget moving relative too */
        XoDirection     t_direction;    /* which way to move */
} XoTraverseInfo;
```

The functions *XoTraverse()* or *xoTraverseFromString()* are typically used to cause traversals to occur. They are conveniance routines that search the

ancestors for ones that know about the traverse callback and execute the callbacks. [*traverse*]

Core.border_width
No documentation.

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

```
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                    composite;
    typedef struct {
        Boolean         do_layout;          /* place widgets flag */
        XtCallbackList  traverse;           /* for keyboard traversals */
    } XoCompBasePart,                   comp_base;
} XoCompBaseRec,        * XoCompBaseWidget;
```

6.3 Box Widget Class

The *Box* widget maintains the number of children on each row based on the number of managed children. Each child is forced to the width of the widest managed child and to the height of the highest managed child. This insures that all widgets have a consistent appearance.

The prefered size of the *Box* widget is calculated as:

$$width = num\_cols \times (max\_width + hoffset) + hoffset$$

$$height = \left\lceil \frac{num\_managed}{num\_cols} \right\rceil \times (max\_height + voffset) + voffset$$

The *max_width* and *max_height* are determined by quering the geometry of each of the managed children to determine the child's preferred width and height. The $hoffset$ and $voffset$ correspond to the *Box*'s resources. The expression for *num_cols* is given below.

$$num\_cols = \begin{cases} \lceil \sqrt{num\_managed} \rceil & \text{if } num\_managed < 9 \\ \lfloor \sqrt{num\_managed} \rfloor & \text{if } num\_managed \geq 9 \end{cases}$$

There are various options to control the resize policy. The resize policy takes affect when the calculated width and heights do not match the *Box* widget's actual width and height. The child widgets can either resized to just fill the *Box* or some can be truncated if the *Box* is too small or extra space inserted if the *Box* is too large.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Box.h>
extern WidgetClass xoBoxWidgetClass;
```

The following resources are provided by the *Box* widget:

Box			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
CompBase (see page 75)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
traverse	Traverse	XtCallbackList	NULL
Box (see page 78)			
hoffset	Offset	Dimension	(caddr_t) 5
insertHspace	Spacers	Boolean	(caddr_t) True
insertVspace	Spacers	Boolean	(caddr_t) True
shrink	Shrink	Boolean	(caddr_t) False
stretch	Stretch	Boolean	(caddr_t) True
voffset	Offset	Dimension	(caddr_t) 5

XtNhoffset The horizontal distance between widgets in the same row. This value is overriden when *stretch* is False and *insertHspace* is True.

If the Box is too small, then at least this much space is placed between each widget. In addition, this indicates the amount of space that is always maintainted between any widgets on the left or right border. [*hoffset*]

XtNinsertHspace
: If set, then extra space is inserted between the children in a row to expand the row to the full width of the box. If this is False, then the children in each row are always seperated by *hoffset* space. [*insert_hspace*]

XtNinsertVspace
: If set, then extra space is inserted between each row to expand the rows to the full height of the box. If this is False, then each row is seperated by *voffset* space. [*insert_vspace*]

XtNshrink
: If this is True and there is not enough room to display all the children, they are all resized proportionally so that they fit. A different scaling factor is calculated for width and height. In other words, if the Box is wide and short, then the child widgets are all proportionally wide and short.

  If this is False, then if there is not enough room to display all the widgets, then any widgets that display past the right or bottom edge are truncated.

  If the Box is very small so that the widgets would have a zero width or height, then the corresponding dimension is made 4 (arbitrary value). [*shrink*]

XtNstretch
: If this is True and there is more room than the widgets fit into, then each widget should be stretched to fill up the extra space. Setting this takes precedence over setting *insertHspace* and *insertVspace*. [*stretch*]

XtNvoffset
: The vertical distance between rows of widgets. This value is overriden when *stretch* is False and *insertVspace* is True.

  If the Box is too small, then at least this much space is placed between each row of widgets. In addition, the amount of space is always maintained between the top edge and the top row of widgets and the bottom edge and bottom row of widgets. [*voffset*]

max_height
: The talles child of the managed children. When sameHeight is True, all children are forced to this height.

max_width
: The widest child of the managed children. When sameWidth is True, all children are forced to this same width.

num_cols
: The number of columns. This is calculated as:

$$num\_cols = \begin{cases} \lceil \sqrt{num\_managed} \rceil & \text{if } num\_managed < 9 \\ \lfloor \sqrt{num\_managed} \rfloor & \text{if } num\_managed \geq 9 \end{cases}$$

num_managed
: This value is calculated in *CalcLayout* and indicates the number of managed children. It is used to calculate the number of rows.

```
typedef struct _XoBoxRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList      children;           /* array of ALL widget children */
        Cardinal        num_children;       /* total number of widget children */
        Cardinal        num_slots;          /* number of slots in children array */
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                composite;
    typedef struct {
        Boolean         do_layout;          /* place widgets flag */
        XtCallbackList  traverse;           /* for keyboard traversals */
    } XoCompBasePart,                   comp_base;
```

```
typedef struct {
    Dimension       hoffset;          /* horizontal separation */
    Dimension       voffset;          /* vertical separation */
    Boolean         shrink;           /* shrink widgets if too many */
    Boolean         stretch;          /* stretch widgets if more space */
    Boolean         insert_hspace;    /* add extra space */
    Boolean         insert_vspace;    /* add extra space */
    int             num_managed;      /* number of managed children */
    Dimension       max_width;        /* widest child */
    Dimension       max_height;       /* highest child */
    int             num_cols;         /* number of columns */
} XoBoxPart,                box;
} XoBoxRec,                 * XoBoxWidget;
```

6.4 Row Widget Class

Layout widgets all with the same height and in a single line.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Row.h>
extern WidgetClass xoRowWidgetClass;
```

The following resources are provided by the *Row* widget:

Row			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
			continued

continued			
Name	**Class**	**Type**	**Default**
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
CompBase (see page 75)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
traverse	Traverse	XtCallbackList	NULL
Row (see page 82)			

filler keep compiler happy.

```
typedef struct _XoRowRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
        Widget          parent;         /* parent widget */
        XrmName         xrm_name;       /* widget resource name quarkified */
        Boolean         being_destroyed; /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
        XtPointer       constraints;    /* constraint record */
        Position        x;              /* window position */
        Position        y;              /* window position */
        Dimension       width;          /* window dimensions */
        Dimension       height;         /* window dimensions */
        Dimension       border_width;   /* window border width */
        Boolean         managed;        /* is widget geometry managed? */
        Boolean         sensitive;      /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;    /* private to event dispatcher */
        XtTMRec         tm;             /* translation management */
        XtTranslations  accelerators;   /* accelerator translations */
```

```
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList      children;           /* array of ALL widget children */
        Cardinal        num_children;       /* total number of widget children */
        Cardinal        num_slots;          /* number of slots in children array */
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                composite;
    typedef struct {
        Boolean         do_layout;          /* place widgets flag */
        XtCallbackList  traverse;           /* for keyboard traversals */
    } XoCompBasePart,                   comp_base;
    typedef struct {
        int             filler;             /* keep compiler happy */
    } XoRowPart,                    row;
} XoRowRec,                 * XoRowWidget;
```

6.5 Column Widget Class

The *Column* widget class implements a simple geometry policy. Each child is forced to the width of the widest child and stacked vertically. No attempt is made to provide multiple columns.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Column.h>
extern WidgetClass xoColumnWidgetClass;
```

The following resources are provided by the *Column* widget:

Column			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Constraint (see page 108)			
ConsBase (see page 110)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
foreground	Foreground	Pixel	XtDefaultForeground
traverse	Traverse	XtCallbackList	NULL
Column (see page 84)			
vspace	Space	Dimension	(XtPointer) 5

XtNvspace The vertical spacing to insert between each widget. [*vspace*]

The following resources are associated with child of this widget.

The following resources are provided by the *Column* widget:

Column			
Name	**Class**	**Type**	**Default**
ConsBase (see page 110)			
sortOrder	SortOrder	int	0
Column (see page 84)			
shrink	Shrink	int	(XtPointer) 1
stretch	Stretch	int	(XtPointer) 1

XtNshrink The amount this widget can shrink. Note that this is not a pixel value and has nothing to do with the number of pixels used. When the normal height of each of the managed child widgets is summed and this value is greater then the current height, then this value is totalled from each of the children. The amount of extra space is then divided by this and each widget has this value times that factor subtracted from it's height. [*shrink*]

XtNstretch The amount this widget can stretch. Note that this is not a pixel value and has nothing to do with the number of pixels used. When the normal height of each of the managed child widgets is summed and this value is less then the current height, then this value is totalled from each of the children. The amount of extra space is then divided by this and each widget has this value times that factor added to it's height. [*stretch*]

skip Flag that subclasses can set to cause the Column geometry handler and query geometry methods to ignore this child.

user_height Whenever quering the child for a height, if this is non-zero use this instead of the returned query height. This allows the Vpane (and other subclasses) to allow the user to change the size of a widget

```
typedef struct _XoColumnRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
        Widget          parent;         /* parent widget */
        XrmName         xrm_name;       /* widget resource name quarkified */
        Boolean         being_destroyed; /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
        XtPointer       constraints;    /* constraint record */
        Position        x;              /* window position */
        Position        y;              /* window position */
        Dimension       width;          /* window dimensions */
        Dimension       height;         /* window dimensions */
        Dimension       border_width;   /* window border width */
        Boolean         managed;        /* is widget geometry managed? */
```

```
        Boolean          sensitive;          /* is widget sensitive to user events */
        Boolean          ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable     event_table;        /* private to event dispatcher */
        XtTMRec          tm;                 /* translation management */
        XtTranslations   accelerators;       /* accelerator translations */
        Pixel            border_pixel;       /* window border pixel */
        Pixmap           border_pixmap;      /* window border pixmap or NULL */
        WidgetList       popup_list;         /* list of popups */
        Cardinal         num_popups;         /* how many popups */
        String           name;               /* widget resource name */
        Screen *         screen;             /* window's screen */
        Colormap         colormap;           /* colormap */
        Window           window;             /* window ID */
        Cardinal         depth;              /* number of planes in window */
        Pixel            background_pixel;   /* window background pixel */
        Pixmap           background_pixmap;  /* window background pixmap or NULL */
        Boolean          visible;            /* is window mapped and not occluded? */
        Boolean          mapped_when_managed; /* map window if it's managed? */
    } CorePart,                  core;
    typedef struct {
        WidgetList       children;           /* array of ALL widget children */
        Cardinal         num_children;       /* total number of widget children */
        Cardinal         num_slots;          /* number of slots in children array */
        XtOrderProc      insert_position;    /* compute position of new child */
    } CompositePart,             composite;
    typedef struct {
        XtPointer        mumble;             /* No new fields, keep C compiler happy */
    } ConstraintPart,            constraint;
    typedef struct {
        Pixel            fg;                 /* foreground color */
        Boolean          reorder;            /* children out of order */
        Boolean          do_layout;          /* place widgets flag */
        XtCallbackList   traverse;           /* for keyboard traversals */
    } XoConsBasePart,            cons_base;
    typedef struct {
        Dimension        vspace;             /* distance between widgets */
    } XoColumnPart,              column;
} XoColumnRec,                 * XoColumnWidget;
```

6.6 Menu Widget Class

The *Menu* widget class implements a simple geometry policy and is used to provide popup menus. Hierarchical (walking) menus are supported. Each child is forced to the width of the widest child and stacked vertically. No attempt is made to provide multiple columns.

The *Menu* widget follows a simple protocol to support popup, pulldown and walking menus for each of those. The method is independent of the widgets being used, however those widgets need special code to cooperate with the menuing system. In particular, the *Menu* widget does not add or change any translations of the child widgets.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Menu.h>
extern WidgetClass xoMenuWidgetClass;
```

The following resources are provided by the *Menu* widget:

Menu			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0

continued

continued			
Name	**Class**	**Type**	**Default**
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Constraint (see page 108)			
ConsBase (see page 110)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
foreground	Foreground	Pixel	XtDefaultForeground
traverse	Traverse	XtCallbackList	NULL
Column (see page 84)			
vspace	Space	Dimension	(XtPointer) 5
Menu (see page 88)			
callbackMenu	CallbackMenu	XtCallbackList	NULL
popup	Popup	Widget	NULL

XtNcallbackMenu
This list of callbacks is executed to popup/down the menu. [*callback_menu*]

XtNpopup Menu used as the popup shell. This is popped up or down by the callbackMenu callback and replaced with the call_data from that callback. [*popup*]

The following resources are associated with child of this widget.

The following resources are provided by the *Menu* widget:

Menu			
Name	**Class**	**Type**	**Default**
ConsBase (see page 110)			
sortOrder	SortOrder	int	0
Column (see page 84)			
shrink	Shrink	int	(XtPointer) 1
stretch	Stretch	int	(XtPointer) 1
Menu (see page 88)			

dummy keep compiler happy.

```
typedef struct _XoMenuRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
```

```
    Widget          parent;             /* parent widget */
    XrmName         xrm_name;           /* widget resource name quarkified */
    Boolean         being_destroyed;    /* marked for destroy */
    XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
    XtPointer       constraints;        /* constraint record */
    Position        x;                  /* window position */
    Position        y;                  /* window position */
    Dimension       width;              /* window dimensions */
    Dimension       height;             /* window dimensions */
    Dimension       border_width;       /* window border width */
    Boolean         managed;            /* is widget geometry managed? */
    Boolean         sensitive;          /* is widget sensitive to user events */
    Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    XtEventTable    event_table;        /* private to event dispatcher */
    XtTMRec         tm;                 /* translation management */
    XtTranslations  accelerators;       /* accelerator translations */
    Pixel           border_pixel;       /* window border pixel */
    Pixmap          border_pixmap;      /* window border pixmap or NULL */
    WidgetList      popup_list;         /* list of popups */
    Cardinal        num_popups;         /* how many popups */
    String          name;               /* widget resource name */
    Screen *        screen;             /* window's screen */
    Colormap        colormap;           /* colormap */
    Window          window;             /* window ID */
    Cardinal        depth;              /* number of planes in window */
    Pixel           background_pixel;   /* window background pixel */
    Pixmap          background_pixmap;  /* window background pixmap or NULL */
    Boolean         visible;            /* is window mapped and not occluded? */
    Boolean         mapped_when_managed; /* map window if it's managed? */
} CorePart,                 core;
typedef struct {
    WidgetList      children;           /* array of ALL widget children */
    Cardinal        num_children;       /* total number of widget children */
    Cardinal        num_slots;          /* number of slots in children array */
    XtOrderProc     insert_position;    /* compute position of new child */
} CompositePart,            composite;
typedef struct {
    XtPointer       mumble;             /* No new fields, keep C compiler happy */
} ConstraintPart,           constraint;
typedef struct {
    Pixel           fg;                 /* foreground color */
    Boolean         reorder;            /* children out of order */
    Boolean         do_layout;          /* place widgets flag */
    XtCallbackList  traverse;           /* for keyboard traversals */
```

```
    } XoConsBasePart,                cons_base;
    typedef struct {
        Dimension       vspace;          /* distance between widgets */
    } XoColumnPart,                  column;
    typedef struct {
        XtCallbackList  callback_menu;   /* execute for new popup */
        Widget          popup;           /* menushell to hold popups */
    } XoMenuPart,                    menu;
} XoMenuRec,                         * XoMenuWidget;
```

6.7 MenuBar Widget Class

Layout widgets all with the same height and in a single line.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/MenuBar.h>
extern WidgetClass xoMenuBarWidgetClass;
```

The following resources are provided by the *MenuBar* widget:

MenuBar			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
			continued

continued			
Name	**Class**	**Type**	**Default**
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
CompBase (see page 75)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
traverse	Traverse	XtCallbackList	NULL
Row (see page 82)			
MenuBar (see page 91)			
callbackMenu	CallbackMenu	XtCallbackList	NULL
popup	Popup	Widget	NULL

XtNcallbackMenu
This list of callbacks is executed to popup/down the menu. [*callback_menu*]

XtNpopup Menu used as the popup shell. This is popped up or down by the callback-Menu callback and replaced with the call_data from that callback. [*popup*]

```
typedef struct _XoMenuBarRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
```

```
        XtTMRec          tm;                /* translation management */
        XtTranslations   accelerators;      /* accelerator translations */
        Pixel            border_pixel;      /* window border pixel */
        Pixmap           border_pixmap;     /* window border pixmap or NULL */
        WidgetList       popup_list;        /* list of popups */
        Cardinal         num_popups;        /* how many popups */
        String           name;              /* widget resource name */
        Screen *         screen;            /* window's screen */
        Colormap         colormap;          /* colormap */
        Window           window;            /* window ID */
        Cardinal         depth;             /* number of planes in window */
        Pixel            background_pixel;  /* window background pixel */
        Pixmap           background_pixmap; /* window background pixmap or NULL */
        Boolean          visible;           /* is window mapped and not occluded? */
        Boolean          mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList       children;          /* array of ALL widget children */
        Cardinal         num_children;      /* total number of widget children */
        Cardinal         num_slots;         /* number of slots in children array */
        XtOrderProc      insert_position;   /* compute position of new child */
    } CompositePart,            composite;
    typedef struct {
        Boolean          do_layout;         /* place widgets flag */
        XtCallbackList   traverse;          /* for keyboard traversals */
    } XoCompBasePart,           comp_base;
    typedef struct {
        int              filler;            /* keep compiler happy */
    } XoRowPart,                row;
    typedef struct {
        XtCallbackList   callback_menu;     /* execute for new popup */
        Widget           popup;             /* menushell to hold popups */
    } XoMenuBarPart,            menu_bar;
} XoMenuBarRec,                 * XoMenuBarWidget;
```

6.8 MenuBox Widget Class

This class implements a Menubar that organizes in a box, just as the Box widget does. This is done as a test to insure that it is easy to subclass an existing composite widget to implement a different layout policy.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/MenuBox.h>
extern WidgetClass xoMenuBoxWidgetClass;
```

The following resources are provided by the *MenuBox* widget:

MenuBox			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
CompBase (see page 75)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
traverse	Traverse	XtCallbackList	NULL
Box (see page 78)			
hoffset	Offset	Dimension	(caddr_t) 5
insertHspace	Spacers	Boolean	(caddr_t) True
insertVspace	Spacers	Boolean	(caddr_t) True
shrink	Shrink	Boolean	(caddr_t) False

continued

continued			
Name	**Class**	**Type**	**Default**
stretch	Stretch	Boolean	(caddr_t) True
voffset	Offset	Dimension	(caddr_t) 5
MenuBox (see page 93)			
callbackMenu	CallbackMenu	XtCallbackList	NULL
popup	Popup	Widget	NULL

XtNcallbackMenu
This list of callbacks is executed to popup/down the menu. [*callback_menu*]

XtNpopup Menu used as the popup shell. This is popped up or down by the callback-Menu callback and replaced with the call_data from that callback. [*popup*]

```
typedef struct _XoMenuBoxRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
```

```
        Pixel           background_pixel;  /* window background pixel */
        Pixmap          background_pixmap; /* window background pixmap or NULL */
        Boolean         visible;           /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList      children;          /* array of ALL widget children */
        Cardinal        num_children;      /* total number of widget children */
        Cardinal        num_slots;         /* number of slots in children array */
        XtOrderProc     insert_position;   /* compute position of new child */
    } CompositePart,                    composite;
    typedef struct {
        Boolean         do_layout;         /* place widgets flag */
        XtCallbackList  traverse;          /* for keyboard traversals */
    } XoCompBasePart,                   comp_base;
    typedef struct {
        Dimension       hoffset;           /* horizontal separation */
        Dimension       voffset;           /* vertical separation */
        Boolean         shrink;            /* shrink widgets if too many */
        Boolean         stretch;           /* stretch widgets if more space */
        Boolean         insert_hspace;     /* add extra space */
        Boolean         insert_vspace;     /* add extra space */
        int             num_managed;       /* number of managed children */
        Dimension       max_width;         /* widest child */
        Dimension       max_height;        /* highest child */
        int             num_cols;          /* number of columns */
    } XoBoxPart,                    box;
    typedef struct {
        XtCallbackList  callback_menu;     /* execute for new popup */
        Widget          popup;             /* menushell to hold popups */
    } XoMenuBoxPart,                    menu_box;
} XoMenuBoxRec,             * XoMenuBoxWidget;
```

6.9 Shell Widget Class

Shell widgets hold an applications's top-level widgets to allow them to communicate with the window manager. Shells have been designed to be as nearly invisible as possible. Clients have to create them, but they should never have to worry about sizes.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Shell.h>
extern WidgetClass shellWidgetClass;
```

The following resources are provided by the *Shell* widget:

Shell			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Shell (see page 96)			
allowShellResize	AllowShellResize	Boolean	False
createPopupChildProc	CreatePopupChildProc	XtCreateChildPopupChildProc	NULL
geometry	Geometry	String	NULL
overrideRedirect	OverrideRedirect	Boolean	False
popdownCallback	Callback	XtCallbackList	NULL
popupCallback	Callback	XtCallbackList	NULL
saveUnder	SaveUnder	Boolean	False
visual	Visual	Visual *	(XtPointer) CopyFromParent

XtNallowShellResize

Controls wheter the widget contained by the child is allowed to try to resize

itself. If allow_shell_resize is False, any geometry requests made by the child will always return XtGeometryNo without interacting with the window manager. [*allow_shell_resize*]

XtNcreatePopupChildProc
Function called to make the popup widget. This is called by XtPopup(). NULL means to not execute the method. The only argument passed is this widget – notably, there is no client_data passed so you cannot have any additional information. [*create_popup_child_proc*]

XtNgeometry
Specifies the x, y, width and height of this widget with a specific format. [*geometry*]

XtNoverrideRedirect
Determines whether the window manager can intercede when the shell window is mapped. [*override_redirect*]

XtNpopdownCallback
Called during XtPopdown(). [*popdown_callback*]

XtNpopupCallback
Called during XtPopup(). [*popup_callback*]

XtNsaveUnder
Instructs the server to attempt to save the contents of windows obscured by the shell when it is mapped (popped up) and to restore those contents when the shell is unmapped. It is useful for pop-up menus. [*save_under*]

XtNvisual The visual to use for this widget and all of it's children [*visual*]

client_specified
Internal state. Probably indicates or of _XtShellPositionValid (1), _XtShellNotReparented (2), _XtShellPPositionOK (4), _XtShellGeometryParsed (8).

grab_kind Internal value that is initialized by the call to XtPopup().

popped_up Internal flag set to indicate this shell is already popped up. Set by XtPopup() and cleared by XtPopdown(). There is a Xo public function, XoIsPoppedUp() that returns the value of this field.

spring_loaded Internal flag set by XtPopupSpringLoaded() to indicate this is popped up by a button down event.

```
typedef struct _ShellRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
```

```
    Widget          parent;             /* parent widget */
    XrmName         xrm_name;           /* widget resource name quarkified */
    Boolean         being_destroyed;    /* marked for destroy */
    XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
    XtPointer       constraints;        /* constraint record */
    Position        x;                  /* window position */
    Position        y;                  /* window position */
    Dimension       width;              /* window dimensions */
    Dimension       height;             /* window dimensions */
    Dimension       border_width;       /* window border width */
    Boolean         managed;            /* is widget geometry managed? */
    Boolean         sensitive;          /* is widget sensitive to user events */
    Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
    XtEventTable    event_table;        /* private to event dispatcher */
    XtTMRec         tm;                 /* translation management */
    XtTranslations  accelerators;       /* accelerator translations */
    Pixel           border_pixel;       /* window border pixel */
    Pixmap          border_pixmap;      /* window border pixmap or NULL */
    WidgetList      popup_list;         /* list of popups */
    Cardinal        num_popups;         /* how many popups */
    String          name;               /* widget resource name */
    Screen *        screen;             /* window's screen */
    Colormap        colormap;           /* colormap */
    Window          window;             /* window ID */
    Cardinal        depth;              /* number of planes in window */
    Pixel           background_pixel;   /* window background pixel */
    Pixmap          background_pixmap;  /* window background pixmap or NULL */
    Boolean         visible;            /* is window mapped and not occluded? */
    Boolean         mapped_when_managed; /* map window if it's managed? */
} CorePart,                 core;
typedef struct {
    WidgetList      children;           /* array of ALL widget children */
    Cardinal        num_children;       /* total number of widget children */
    Cardinal        num_slots;          /* number of slots in children array */
    XtOrderProc     insert_position;    /* compute position of new child */
} CompositePart,                    composite;
typedef struct {
    String          geometry;           /* geometry specification */
    XtCreateChildPopupChildProc create_popup_child_proc; /* method to make the child */
    XtGrabGind      grab_kind;          /* grab to execute */
    Boolean         spring_loaded;      /* is this spring loaded */
    Boolean         popped_up;          /* popped up */
    Boolean         allow_shell_resize; /* request new sizes from child */
    Boolean         client_specified;   /* internal state */
```

```
        Boolean          save_under;        /* tell server to try save under */
        Boolean          override_redirect; /* if window manager can effect */
        XtCallbackList   popup_callback;    /* executed on popup */
        XtCallbackList   popdown_callback;  /* executed on popdown */
        Visual *         visual;            /* visual to use */
    } ShellPart,                 shell;
} ShellRec,                      * ShellWidget;
```

6.10 OverrideShell Widget Class

Used for shell windows that completely bypass the window manager (for example, pop-up menu shells).

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/OverrideS.h>
extern WidgetClass overrideShellWidgetClass;
```

The following resources are provided by the *OverrideShell* widget:

OverrideShell			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen

continued

continued

Name	Class	Type	Default
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Shell (see page 96)			
allowShellResize	AllowShellResize	Boolean	False
createPopupChildProc	CreatePopupChildProc	XtCreateChildPopupChildProc	NULL
geometry	Geometry	String	NULL
overrideRedirect	OverrideRedirect	Boolean	False
popdownCallback	Callback	XtCallbackList	NULL
popupCallback	Callback	XtCallbackList	NULL
saveUnder	SaveUnder	Boolean	False
visual	Visual	Visual *	(XtPointer) CopyFromParent
OverrideShell (see page 100)			

emptry keep compiler happy.

```
typedef struct _OverrideShellRec {
    typedef struct {
        Widget          self;           /* pointer to widget itself */
        WidgetClass     widget_class;   /* pointer to Widget's ClassRec */
        Widget          parent;         /* parent widget */
        XrmName         xrm_name;       /* widget resource name quarkified */
        Boolean         being_destroyed; /* marked for destroy */
        XtCallbackList  destroy_callbacks; /* who to call when widget destroyed */
        XtPointer       constraints;    /* constraint record */
        Position        x;              /* window position */
        Position        y;              /* window position */
        Dimension       width;          /* window dimensions */
        Dimension       height;         /* window dimensions */
        Dimension       border_width;   /* window border width */
        Boolean         managed;        /* is widget geometry managed? */
        Boolean         sensitive;      /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;    /* private to event dispatcher */
        XtTMRec         tm;             /* translation management */
```

```
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList      children;           /* array of ALL widget children */
        Cardinal        num_children;       /* total number of widget children */
        Cardinal        num_slots;          /* number of slots in children array */
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                composite;
    typedef struct {
        String          geometry;           /* geometry specification */
        XtCreateChildPopupChildProc create_popup_child_proc; /* method to make the child */
        XtGrabGind      grab_kind;          /* grab to execute */
        Boolean         spring_loaded;      /* is this spring loaded */
        Boolean         popped_up;          /* popped up */
        Boolean         allow_shell_resize; /* request new sizes from child */
        Boolean         client_specified;   /* internal state */
        Boolean         save_under;         /* tell server to try save under */
        Boolean         override_redirect;  /* if window manager can effect */
        XtCallbackList  popup_callback;     /* executed on popup */
        XtCallbackList  popdown_callback;   /* executed on popdown */
        Visual *        visual;             /* visual to use */
    } ShellPart,                shell;
    typedef struct {
        int             emptry;             /* keep compiler happy */
    } OverrideShellPart,                override_shell;
} OverrideShellRec,         * OverrideShellWidget;
```

6.11 MenuShell Widget Class

Implements a popup shell that can manage multiple children and is useful for walking menus.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/MenuShell.h>
extern WidgetClass xoMenuShellWidgetClass;
```

The following resources are provided by the *MenuShell* widget:

MenuShell			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Shell (see page 96)			
allowShellResize	AllowShellResize	Boolean	False
createPopupChildProc	CreatePopupChildProc	XtCreateChildPopupChildProc	NULL

continued

continued			
Name	**Class**	**Type**	**Default**
geometry	Geometry	String	NULL
overrideRedirect	OverrideRedirect	Boolean	False
popdownCallback	Callback	XtCallbackList	NULL
popupCallback	Callback	XtCallbackList	NULL
saveUnder	SaveUnder	Boolean	False
visual	Visual	Visual *	(XtPointer) CopyFromParent
OverrideShell (see page 100)			
MenuShell (see page 103)			
borderWidth	BorderWidth	Dimension	0*
borderClass	Class	WidgetClass	"DefaultBorder"
borderWidget	BorderWidget	Widget	NULL
callbackMenu	CallbackMenu	XtCallbackList	NULL
current	Current	Widget	NULL
popupStarter	PopupStarter	Widget	NULL
start	Start	Boolean	False
traverse	Traverse	XtCallbackList	NULL

XtNborderClass
Object class that implements drawing the border. [*border_class*]

XtNborderWidget
This is the instance of the widget that draws the border You have two choices, either set the *WidgetClass* at initialization time or create an object with this widget as it's parent and then use *XtSetValues()* to set this resource. The old widget is destroyed. [*border_widget*]

XtNcallbackMenu
This list of callbacks is executed to popup/down the menu. [*callback_menu*]

XtNcurrent Set this to the current child that should be displayed. Must be a child of this widget. This override shell is resized to the width and height of the child. The new submenu is managed if it was not already managed. [*current*]

XtNpopupStarter
Set this to the widget that caused this menu to be displayed. This is used for traversal so that we know our real parent knows where to go to next. [*popup_starter*]

XtNstart Indicates that this override shell is the beginning of a possible cascade of menus. It should be set to True for MenuShell's that are a child of a MenuBar or for the first in a popup menu. [*start*]

XtNtraverse List of callbacks executed to accomplish keyboard traversals. There should only be one callback on this list. Behaviour is undefined if there is more than one callback. If the application wishes to change the callback executed, it should delete the callback list and then install it's own private handler.

At initialization time, the Initialize method executes XtAddCallback() for this callback list with the value of the call method *traverse* as the callback to execute. To replace the default traversal method, subclasses should override this value.

The arguments passed in *call_data* is a structure of type XoTraversalInfo defined as follows:

```
typedef enum
{
        XoTRAVERSE_PREVIOUS, XoTRAVERSE_NEXT, XoTRAVERSE_HOME,
        XoTRAVERSE_END,
} XoDirection;

/*
 * This information is passed to the traverse callback.
 */
typedef struct _xo_traverse_info
{
        Widget          t_start;        /* widget moving relative too */
        XoDirection     t_direction;    /* which way to move */
} XoTraverseInfo;
```

The functions *XoTraverse()* or *xoTraverseFromString()* are typically used to cause traversals to occur. They are conveniance routines that search the ancestors for ones that know about the traverse callback and execute the callbacks. [*traverse*]

Core.border_width
No documentation.

The following actions are defined by this widget.

XoMenuPopdown
Remove all currently popped up menus.

The following translations are used by this widgets.

<Btn1Up> XoMenuPopdown()

Remove all the popped down menus whenever a button is released.

```
typedef struct _XoMenuShellRec {
    typedef struct {
        Widget              self;                /* pointer to widget itself */
        WidgetClass         widget_class;        /* pointer to Widget's ClassRec */
        Widget              parent;              /* parent widget */
        XrmName             xrm_name;            /* widget resource name quarkified */
        Boolean             being_destroyed;     /* marked for destroy */
        XtCallbackList      destroy_callbacks;   /* who to call when widget destroyed */
        XtPointer           constraints;         /* constraint record */
        Position            x;                   /* window position */
        Position            y;                   /* window position */
        Dimension           width;               /* window dimensions */
        Dimension           height;              /* window dimensions */
        Dimension           border_width;        /* window border width */
        Boolean             managed;             /* is widget geometry managed? */
        Boolean             sensitive;           /* is widget sensitive to user events */
        Boolean             ancestor_sensitive;  /* are all ancestors sensitive? */
        XtEventTable        event_table;         /* private to event dispatcher */
        XtTMRec             tm;                  /* translation management */
        XtTranslations      accelerators;        /* accelerator translations */
        Pixel               border_pixel;        /* window border pixel */
        Pixmap              border_pixmap;       /* window border pixmap or NULL */
        WidgetList          popup_list;          /* list of popups */
        Cardinal            num_popups;          /* how many popups */
        String              name;                /* widget resource name */
        Screen *            screen;              /* window's screen */
        Colormap            colormap;            /* colormap */
        Window              window;              /* window ID */
        Cardinal            depth;               /* number of planes in window */
        Pixel               background_pixel;    /* window background pixel */
        Pixmap              background_pixmap;   /* window background pixmap or NULL */
        Boolean             visible;             /* is window mapped and not occluded? */
        Boolean             mapped_when_managed; /* map window if it's managed? */
    } CorePart,                    core;
    typedef struct {
        WidgetList          children;            /* array of ALL widget children */
        Cardinal            num_children;        /* total number of widget children */
        Cardinal            num_slots;           /* number of slots in children array */
        XtOrderProc         insert_position;     /* compute position of new child */
    } CompositePart,                        composite;
    typedef struct {
        String              geometry;            /* geometry specification */
        XtCreateChildPopupChildProc create_popup_child_proc; /* method to make the child */
        XtGrabGind          grab_kind;           /* grab to execute */
```

```
        Boolean         spring_loaded;      /* is this spring loaded */
        Boolean         popped_up;          /* popped up */
        Boolean         allow_shell_resize; /* request new sizes from child */
        Boolean         client_specified;   /* internal state */
        Boolean         save_under;         /* tell server to try save under */
        Boolean         override_redirect;  /* if window manager can effect */
        XtCallbackList  popup_callback;     /* executed on popup */
        XtCallbackList  popdown_callback;   /* executed on popdown */
        Visual *        visual;             /* visual to use */
    } ShellPart,                    shell;
    typedef struct {
        int             emptry;             /* keep compiler happy */
    } OverrideShellPart,            override_shell;
    typedef struct {
        XtCallbackList  callback_menu;      /* execute for new popup */
        Widget          current;            /* current submenu */
        Widget          popup_starter;      /* widget that popped me up */
        Boolean         start;              /* if first in hierarchy */
        WidgetClass     border_class;       /* Widget Class of border object */
        Widget          border_widget;      /* draws the border */
        XtCallbackList  traverse;           /* for keyboard traversals */
    } XoMenuShellPart,              menu_shell;
} XoMenuShellRec,           * XoMenuShellWidget;
```

Chapter 7

The Manager + Data Widgets

7.1 Constraint Widget Class

This class allows other widgets to be it's children.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Constraint.h>
extern WidgetClass constraintWidgetClass;
```

The following resources are provided by the *Constraint* widget:

Constraint			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0

continued

continued			
Name	**Class**	**Type**	**Default**
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Constraint (see page 108)			

mumble No new fields, keep C compiler happy.

```
typedef struct _ConstraintRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
```

```
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList      children;           /* array of ALL widget children */
        Cardinal        num_children;       /* total number of widget children */
        Cardinal        num_slots;          /* number of slots in children array */
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                composite;
    typedef struct {
        XtPointer       mumble;             /* No new fields, keep C compiler happy */
    } ConstraintPart,               constraint;
} ConstraintRec,            * ConstraintWidget;
```

7.2 ConsBase Widget Class

This class provides the framework for all constraint based widgets in this widget set. A constraint resource, sortOrder, is added to each child. The application can change this to cause an arbitrary ordering of the children independent of the creation order.

To support sortOrder a new class method, sort, is used to do the sorting. The default one performs a simple insertion sort which has a worst case run time of $O(n^2)$ however offers the advantage of being only $O(n)$ if only one element is out of order.

The instance variable doLayout determines when new children and sorting order is to be taken into account. Setting this to False causes any changes to the set of managed children and sorting order to be ignored. When it is changed to be True, any such changes are accounted for. This allows the application to add, remove or reorder a number of widgets without causing expensive resize calculations to be performed.

A traversal callback is added to support arbitrary keyboard traversals. This is identical to the CompBase's traversals. See the traversal resource for more details.

The geometry policy of this class is quite simple. All requests for changes in position and size are granted. After any such changes, a new size is calculated as being the right most widget and the bottom most widget. The parent is then asked for this new size.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/ConsBase.h>
extern WidgetClass xoConsBaseWidgetClass;
```

The following resources are provided by the *ConsBase* widget:

ConsBase			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Constraint (see page 108)			
ConsBase (see page 110)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
foreground	Foreground	Pixel	XtDefaultForeground
traverse	Traverse	XtCallbackList	NULL

XtNdoLayout

If this flag is set, then the widget should readjust the children whenever

changes occur. If the flag is False, the widget should ignore changes in the set of manage children and resize events until the flag is set to True. When the flag changes from False (do not reaarange the children) to True, the complete layout is recalculated.

This flag makes it conveniant to create or manage/unmanage a group of widgets one at a time without incurring the overhead of constantly rearranging the widgets. [*do_layout*]

XtNforeground

The color for drawing primary objects such as text and lines. Changing this during execution causes the object to change color. [*fg*]

XtNtraverse List of callbacks executed to accomplish keyboard traversals. There should only be one callback on this list. Behaviour is undefined if there is more than one callback. If the application wishes to change the callback executed, it should delete the callback list and then install it's own private handler.

At initialization time, the Initialize method executes XtAddCallback() for this callback list with the value of the call method *traverse* as the callback to execute. To replace the default traversal method, subclasses should override this value.

The arguments passed in *call_data* is a structure of type XoTraversalInfo defined as follows:

```
typedef enum
{
        XoTRAVERSE_PREVIOUS, XoTRAVERSE_NEXT, XoTRAVERSE_HOME,
        XoTRAVERSE_END,
} XoDirection;

/*
 * This information is passed to the traverse callback.
 */
typedef struct _xo_traverse_info
{
        Widget          t_start; /* widget moving relative too */
        XoDirection     t_direction;    /* which way to move */
} XoTraverseInfo;
```

The functions *XoTraverse()* or *xoTraverseFromString()* are typically used to cause traversals to occur. They are conveniance routines that search the ancestors for ones that know about the traverse callback and execute the callbacks. [*traverse*]

Core.border_width

No documentation.

reorder If this flag is set, it indicates that cons_base_class.sort needs to called to sort the children. It is usually false. It is set when a child's sort_order constraint changes and do_layout is False. When do_layout is changed to True, SetValues classes the class sort method and clears reorder.

The following resources are associated with child of this widget.

The following resources are provided by the *ConsBase* widget:

ConsBase			
Name	**Class**	**Type**	**Default**
ConsBase (see page 110)			
sortOrder	SortOrder	int	0

XtNsortOrder

This defines the order that the children are displayed in. Never changing this keeps the children in the order they are inserted in. If you are reording all the children, it is useful to set doLayout to False, change each of the children widgets sortOrder with XtSetValues(), and then set doLayout to True. Otherwise, each time sortOrder changes, all the children are layed out, again. [*sort_order*]

```
typedef struct _XoConsBaseRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
```

```
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
        Cardinal        depth;              /* number of planes in window */
        Pixel           background_pixel;   /* window background pixel */
        Pixmap          background_pixmap;  /* window background pixmap or NULL */
        Boolean         visible;            /* is window mapped and not occluded? */
        Boolean         mapped_when_managed; /* map window if it's managed? */
    } CorePart,                 core;
    typedef struct {
        WidgetList      children;           /* array of ALL widget children */
        Cardinal        num_children;       /* total number of widget children */
        Cardinal        num_slots;          /* number of slots in children array */
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                composite;
    typedef struct {
        XtPointer       mumble;             /* No new fields, keep C compiler happy */
    } ConstraintPart,               constraint;
    typedef struct {
        Pixel           fg;                 /* foreground color */
        Boolean         reorder;            /* children out of order */
        Boolean         do_layout;          /* place widgets flag */
        XtCallbackList  traverse;           /* for keyboard traversals */
    } XoConsBasePart,                cons_base;
} XoConsBaseRec,            * XoConsBaseWidget;
```

7.3 List Widget Class

Formats a list of RectObj's into rows and columns.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/List.h>
extern WidgetClass xoListWidgetClass;
```

The following resources are provided by the *List* widget:

List			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Constraint (see page 108)			
ConsBase (see page 110)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
foreground	Foreground	Pixel	XtDefaultForeground
traverse	Traverse	XtCallbackList	NULL
Column (see page 84)			
vspace	Space	Dimension	(XtPointer) 5
List (see page 114)			
callback	Callback	XtCallbackList	NULL
current	Current	Widget	NULL

XtNcallback List of functions executed by the Notify() action or whenever the currently selected object is changed. [*callbacks*]

XtNcurrent The currently selected object. This can be changed either by the user clicking on one of the objects and causing the Notify() action or by using XtSetValues() to change this value. In either case, the XtNselected resource of the

object is set to True and the previously selected object's XtNselected resource is set to False.

If you want to have widgets, such as Button, be in this list, then the application code should add a callback to the Button (or whichever) widget and use XtSetValues() to update the XtNcurrent in the callback. However, if a widget is used, the List widget does not modify the translations of that widget in any way – so it is up to the child widget to implement notifing the application and the application notifying the list of the change in selected items.

Changing this value executes the XtNcallback. [*current*]

which Keeps track of which widget is currently selected. This is calculated by the CurrentLocate() action. Setting this to -1 indicates nothing is selected. The CurrentSet() action uses this index to set the currently selected widget.

The following resources are associated with child of this widget.

The following resources are provided by the *List* widget:

List			
Name	**Class**	**Type**	**Default**
ConsBase (see page 110)			
sortOrder	SortOrder	int	0
Column (see page 84)			
shrink	Shrink	int	(XtPointer) 1
stretch	Stretch	int	(XtPointer) 1
List (see page 114)			

dummy keep compiler happy.

The following actions are defined by this widget.

Notify Executes the XtNcallback list. Used to notify the application that a new item on the list has been selected.

CurrentClear

Tells the currently selected widget that it is no longer selected. Sets the internal value of the currently selected widget to NULL.

CurrentLocate

Uses the x and y coordinates of the event to locate an object. If none is found, then the internal value of *which* is set to -1 to indicate nothing was found.

CurrentSet Updates the currently selected widget.

The following translations are used by this widgets.

<Btn1Down> CurrentClear()

Remove any selected item from the list.

<Btn1Up> CurrentLocate() CurrentSet() Notify()

Choose the new item and notify the application a new selection is made.

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

```
        WidgetList      children;           /* array of ALL widget children */
        Cardinal        num_children;       /* total number of widget children */
        Cardinal        num_slots;          /* number of slots in children array */
        XtOrderProc     insert_position;    /* compute position of new child */
    } CompositePart,                    composite;
    typedef struct {
        XtPointer       mumble;             /* No new fields, keep C compiler happy */
    } ConstraintPart,                   constraint;
    typedef struct {
        Pixel           fg;                 /* foreground color */
        Boolean         reorder;            /* children out of order */
        Boolean         do_layout;          /* place widgets flag */
        XtCallbackList  traverse;           /* for keyboard traversals */
    } XoConsBasePart,                     cons_base;
    typedef struct {
        Dimension       vspace;             /* distance between widgets */
    } XoColumnPart,                     column;
    typedef struct {
        XtCallbackList  callbacks;          /* executed when selected */
        Widget          current;            /* currently selected */
        int             which;              /* index of selected widget */
    } XoListPart,                   list;
} XoListRec,                        * XoListWidget;
```

7.4 Vpane Widget Class

Arranges children in a vertical fashion. All children are forced to the same width. Children can have something much like TeX's glue attached to indicate how the widgets should be resized in relation to each other. There is a stretchability and shrinkability factor with 0 indicate none. The total glue is added and any extra space or stretching is used to adjust the heights.

The following include file is needed. The class variable is declared in this file and should be used in the call to *XtCreateWidget()* or *XtCreateManagedWidget()*.

```
#include <X11/Xo/Vpane.h>
extern WidgetClass xoVpaneWidgetClass;
```

The following resources are provided by the *Vpane* widget:

Vpane			
Name	**Class**	**Type**	**Default**
Core (see page 12)			
accelerators	Accelerators	XtTranslations	NULL
ancestorSensitive	Sensitive	Boolean	XtCopyAncestorSensitive
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	Pixmap	Pixmap	XtUnspecifiedpixmap
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	Pixmap	Pixmap	XtUnspecifiedPixmap
borderWidth	BorderWidth	Dimension	0
colormap	Colormap	Colormap	XtCopyFromParent
depth	Depth	Cardinal	XtCopyFromParent
destroyCallback	Callback	XtCallbackList	NULL
height	Height	Dimension	0
mappedWhenManaged	MappedWhenManaged	Boolean	True
screen	Screen	Screen *	XtCopyScreen
sensitive	Sensitive	Boolean	True
translations	Translations	XtTMRec	XtInheritTranslations
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
Composite (see page 71)			
children	ReadOnly	WidgetList	NULL
insertPosition	InsertPosition	XtOrderProc	NULL
numChildren	ReadOnly	Cardinal	NULL
Constraint (see page 108)			
ConsBase (see page 110)			
borderWidth	BorderWidth	Dimension	0*
doLayout	DoLayout	Boolean	(caddr_t) True
foreground	Foreground	Pixel	XtDefaultForeground
traverse	Traverse	XtCallbackList	NULL
Column (see page 84)			
vspace	Space	Dimension	(XtPointer) 5
Vpane (see page 118)			
vspace	Space	Dimension	(XtPointer) 10*
gripClass	Class	WidgetClass	"Grip"

XtNgripClass

Specifies the widget class for creating grips. For $n-1$ children (all but the last) a grip is created to allow the user to interactively resize the child. This class is passed to create the appropiate widget. It is expected to have an XtNmoveCallback that passes an XoMoveCBdata structure for the call data – typically on each motion notify event.

This could be generalized to support other methods of notification but this seems adequate. [*grip_class*]

Column.vspace
No documentation.

gc Used to draw the lines seperating the children.

grip_count How many grip widgets are in *grips*.

grips List of grips, each associated with a particular child through the constraint resources.

layout_grips Set to True if it is safe to layout the grips. This is set to False when grips are being added and destroyed.

move_gc Used to draw the lines seperating the children while the grips are moving.

moving_grip The grip widget that is currently being moved. NULL if one is not being moved. This is used as a flag to indicate when *start_y* should be initialized.

start_y This gives the initial y coordinate of the grip currently being moved. This is used to determine how much to resize the child corresponding to this grip.

total_shrink This internal value is used to keep track of the total shrinkability from all the currently managed widgets.

total_stretch This internal value is used to keep track of the total stretchability from all the currently managed widgets.

The following resources are associated with child of this widget.

The following resources are provided by the *Vpane* widget:

Vpane			
Name	**Class**	**Type**	**Default**
ConsBase (see page 110)			
sortOrder	SortOrder	int	0
Column (see page 84)			
shrink	Shrink	int	(XtPointer) 1
stretch	Stretch	int	(XtPointer) 1
Vpane (see page 118)			
			continued

continued			
Name	**Class**	**Type**	**Default**
childType	ChildType	XoChildType	XoCHILD

XtNchildType

This is set to XoGRIP for widgets the vpane creates and XoCHILD for all others. This allows us to distuinguish between the widgets. Applications should not be setting this value since it is used internally by the Vpane. [*type*]

after Which child this grip is placed above.

before Which child this grip is placed below.

orig_y The position this grip would be in without the user moving things around.

```
typedef struct _XoVpaneRec {
    typedef struct {
        Widget          self;               /* pointer to widget itself */
        WidgetClass     widget_class;       /* pointer to Widget's ClassRec */
        Widget          parent;             /* parent widget */
        XrmName         xrm_name;           /* widget resource name quarkified */
        Boolean         being_destroyed;    /* marked for destroy */
        XtCallbackList  destroy_callbacks;  /* who to call when widget destroyed */
        XtPointer       constraints;        /* constraint record */
        Position        x;                  /* window position */
        Position        y;                  /* window position */
        Dimension       width;              /* window dimensions */
        Dimension       height;             /* window dimensions */
        Dimension       border_width;       /* window border width */
        Boolean         managed;            /* is widget geometry managed? */
        Boolean         sensitive;          /* is widget sensitive to user events */
        Boolean         ancestor_sensitive; /* are all ancestors sensitive? */
        XtEventTable    event_table;        /* private to event dispatcher */
        XtTMRec         tm;                 /* translation management */
        XtTranslations  accelerators;       /* accelerator translations */
        Pixel           border_pixel;       /* window border pixel */
        Pixmap          border_pixmap;      /* window border pixmap or NULL */
        WidgetList      popup_list;         /* list of popups */
        Cardinal        num_popups;         /* how many popups */
        String          name;               /* widget resource name */
        Screen *        screen;             /* window's screen */
        Colormap        colormap;           /* colormap */
        Window          window;             /* window ID */
```

```
    Cardinal        depth;              /* number of planes in window */
    Pixel           background_pixel;   /* window background pixel */
    Pixmap          background_pixmap;  /* window background pixmap or NULL */
    Boolean         visible;            /* is window mapped and not occluded? */
    Boolean         mapped_when_managed; /* map window if it's managed? */
} CorePart,                 core;
typedef struct {
    WidgetList      children;           /* array of ALL widget children */
    Cardinal        num_children;       /* total number of widget children */
    Cardinal        num_slots;          /* number of slots in children array */
    XtOrderProc     insert_position;    /* compute position of new child */
} CompositePart,                composite;
typedef struct {
    XtPointer       mumble;             /* No new fields, keep C compiler happy */
} ConstraintPart,               constraint;
typedef struct {
    Pixel           fg;                 /* foreground color */
    Boolean         reorder;            /* children out of order */
    Boolean         do_layout;          /* place widgets flag */
    XtCallbackList  traverse;           /* for keyboard traversals */
} XoConsBasePart,                 cons_base;
typedef struct {
    Dimension       vspace;             /* distance between widgets */
} XoColumnPart,                 column;
typedef struct {
    GC              gc;                 /* for drawing lines */
    GC              move_gc;            /* for drawing lines */
    int             total_stretch;      /* summary of stretch */
    int             total_shrink;       /* summary of shrink */
    Position        start_y;            /* starting position */
    Widget          moving_grip;        /* grip currentl moving */
    WidgetClass     grip_class;         /* what widget to create */
    int             grip_count;         /* number of grips */
    Widget *        grips;              /* list of grips */
    int             layout_grips;       /* safe to place grips */
} XoVpanePart,                  vpane;
} XoVpaneRec,               * XoVpaneWidget;
```

Chapter 8

Utility functions

8.1 Inheriting Resources

A novel approach is using gadgets to accomplish most if not all of the display portion of the widget set. This gives the user and application developer great flexibility in choosing how a widget actually displays. Unfortunately, it makes it difficult or unintuitive to set resources for the objects that actually do the drawing for each widget. The obvious solution is to allow a set of resources to be passed from the widget to each of the gadgets that implement the output. The complication is that we cannot hardcode this set of resources because we do not know a priori the complete set of gadgets that might be used by each widget.

There are two functions that solve this problem for us. The first, *_xoInheritedSet* is called by the parent widget on each of it's gadgets at the time the gadget is created (typically in the *Initialize()* method) to pass any arguments the application may have used in creating the widget. It is also used in the *SetValues* method to pass any arguments to each of the gadgets this widget uses. It is not sufficient to just pass these arguments along because there may be things set, such as *width* or *height* that would not be appropriate to pass along.

The second is *_xoInheritedGet* and is used by each of the gadgets (actually it is only needed by ObjDraw) to fetch any resources in the resource database by pretending to have the same name as the parent widget.

The final key to the above is determining the set of resources to inherit. This varies from gadget to gadget but may also vary depending on how the gadget is being used. To circumvent this problem, the resource *inheritResources* is used from the gadget. It is set to sorted list of strings in a *NULL* terminated array of strings. The application writer can change this in the resource database if a gadget is not being used in the anticipated way.

8.2 Converters

There are several converters registered by this widget set. Every class at initialization time calls the *_xoRegisterConverters* function. This registers converters for:

String	ObjectClass
String	Gravity
String	Justify
String	Sorted String List

These converters need some work. In particular, they should ignore case when doing comparisons. It would also be nice to do the opposite conversion back to a string. It should also be easier for an application to convert the values. The string to object class converter also needs methods to add additional classes to the list.

8.3 Creating Menus

An interface is also included for building hierarchical menus from an external specification. The external specification mimics that used by OSF's **mwm** window manager. A short example illustrating some of the possibilities follows:

```
Menu Main {
        File    _F                      f.bar File
        Edit    _E                      f.bar Edit
        View    _V                      f.bar View
        Options _O                      f.bar Options
        Help    _H                      f.bar Help
}

Menu File {
        Get     _G                      f.menu Get
        Put     _P                      f.menu Put
        Print   _P      Alt<Key>Print   f.exec print
        Restart                         f.exec restart
        Source                          f.exec source
        Exit    _E                      f.exec exit
}
...
```

The basic syntax is

```
Menu menu_name {
  <label1> [mnemonic] [accelerator] function [arguments]
  <label2> [mnemonic] [accelerator] function [arguments]
  ...
}
```

Currently there are several functions defined: f.bar, f.menu, f.seperator, and f.exec. Some small amount of additional code needs to be added to allow the application to define additional functions.

The *mnemonic* and *accelerator* can be entered but are ignored.

The specification is read from a file using *xoMenuReadFile* which returns an opaque pointer and takes a single argument, the name of the file to read. A single application can have multiple such lists.

The menu hierarchy is created by calling *xoMenuCreate* with three arguments: the data returned from the earlier call to *xoMenuReadFile*, the name of the Menu to create and the widget that is the parent of the menu system.

8.4 Miscellaneous

To get the name of a widget use *XoName*. This returns the name of a widget in the form `Class.instance` where `Class` is the class name of the widget and `instance` is the instance name of the widget. A static buffer is used so only a single call at a time is supported.

```
char *
XoName (gw)
Widget gw;
```

In order to change the stacking mode of a child *XoStackChild* directly calls *XConfigureWindow* to change the current stacking mode.

```
void
XoStackChild (gw, stack_mode)
Widget gw;
int stack_mode;
```

To change wheter a widget is mapped or unmapped, *XoMap* provides a simple interface to *XtMap/UnmapWidget*. If *flag* is *True* then the widget is mapped otherwise it is unmapped.

```
void
XoMap (gw, flag)
Widget gw;
int flag;
```

To call a widgets *expose* method directly, use *_XoObjRedraw*.

```
void
_XoObjRedraw (gw, event, region)
```

```
Widget gw; /* should be subclassof ObjDraw */
XEvent *event; /* the exposure event */
Region region; /* the uncovered region */
```

To cause a widget to redisplay itself, use *_XoRedisplaySelf* which clears the widget's window without causing an expose event and then calls the *expose* method. This is particular useful for actions that need to force a redraw.

```
void
_XoRedisplaySelf (gw)
Widget gw;
```

A utility to determine if a *Shell* widget is currently popped up is *_XoIsPoppedUp*. This returns true if the private instance field *popped_up* of the shell is True. If the widget is not a subclass of *Shell* it returns False.

```
Boolean
_XoIsPoppedUp (gw)
Widget gw;
```

To return the parent shell of a widget, use *XoShellGet*. This returns the first shell that the passed in widget is an ancestor of. This is particularly useful for displaying messages correctly placed in the center of the application.

```
Widget
XoShellGet (gw)
Widget gw;
```

To find the parent of all widgets in the application, use *XoTop*. This just uses the *parent* relation to find the topmost widget. You may want to use *XoToplevelGet* which finds the highest popped up shell of the passed in widget. This may be what is intuitively expected if an application uses a single, unrealized top level shell to initialize the application and the creates several other shells for the actual windows of the application.

```
Widget
XoTop (gw)
Widget gw;
Widget
XoToplevelGet (gw)
Widget gw;
```

Within the widget set, warning messages are more easily reported with *_xoWarn*. This takes four arguments, a widget used to determine the application context and possibly some more information, the general type of the error such as badArgument; detailed name of the error and finally the actual error message. *XtAppWarningMsg* is called with the resource class of the error set to `"XoError"`. The error message is supplemented with the widget class and name of the widget reporting the error by prefixing the error message with "and passing in the result from *XoName* to get the instance and class name of the widget.

```
void
_XoWarn (gw, err_class, err_detail, err_message)
```

```
Widget gw; /* widget for context and info */
char *err_class; /* general type of error */
char *err_detail; /* particular error */
char *err_message; /* description */
```

Need to provide a facility for defining application specific key grabs. Should also provide some help mechanism for that.

Chapter 9

The build Program

When writing widgets, one of the key features is that much of it is really boilerplate.

One needs to create an include file that has a bunch of `#define`'s for the resource strings and classes as well as a declaration of the class variable. One is also supposed to include a description of the resources used and the default values. Usually this is only partially correct.

A second include file that defines the actual class and instance parts and relates those to the classes the widget is inheriting from needs to be written.

Finally, the actual code is placed in a *.c* file that contains 60+ lines of initialization plus potentially several dozen more lines that deal with action and translation tables.

Typically one copies all the above boilerplate from some other widget and uses a *sed* script or *emacs* to change all the names to the new class. This is followed by some hand editing to add some new class initialization, resources descriptions, fix up errors introduced by the sed script, etc. The result is usually something that does not properly document things in the include files, has no external documentation and is a mish mash of coding styles depending on the ancestory of the template used. Worse, it's impossible to get it all right in one try.

The frustrating part is it really is just boilerplate – not at all interesting to the job at hand. A program was written to handle generating all of this boilerplate. It's called *build* because *buildwidgets* is too long and the original name, *we* which stands for widget editor, is too ostentatious for what the program actually does.

9.1 Output Files

The following section describes the actual input to *build* but the output is introduced early to help justify it's use. For now just assume the input is a description of several widget

classes with each new widget class only specifying what is new (added class variables and instance variables) or what has changed (overriding superclass default values).

The result of running *build* is four output files per class. Three are C code to be included by the widget writer and one is documentation to be used by the widget reader. There are plans to write another documentation file that can be used as a checklist to examine the source code for a widget to insure all instance variables are initialized and any allocated resources are destroyed but this is not implemented. An additional output file that provides a simple indented list of the class hierarchy is also provided.

The three include files are the public header file used by applications; the private header file used by widget writers; and a header file used by the widget writer to initialize the resources and class variables.

The file is only replaced if it changes from one version to the next. The backup file follows the GNU emacs naming conventions of appending a "`.~n~`" where *n* is incremented starting from 1 until a unique file name is generated.

9.1.1 Public Include File

The public include file is used by the application primarily for the declaration of the global widget class variable passed to *XtCreateWidget()* and the defined constants for resource names. This public include file also provides the recommended way for documenting the resources used by the widgets. Rarely is the documentation complete for most widgets.

The first thing to note about all the generated include file's is they are carefully wrapped in *ifdef*'s to prevent the file from being multiply included during compilation. The symbol used to avoid the file being multiply included is generated as:

<*widget-set-name*><*filename*>_

with *filename* having all '.' (periods) replaced with '_' (underscores) and an '_' prepended and appended to the name. All characters in the filename are converted to upper case. For example, the file *Label.h* in the Open widget set with 'Xo' used as the abbreviation, generates the symbol `_XO_LABEL_H_` to avoid including the file several times.

The next thing that is common for all the include files is that RCS id and log variables are defined for revision control purposes. Unfortunately, this implies that the include file's may drift from the original specification which would be a serious mistake. Another approach that should probably be taken is to incorporate the log information from the original specification. In any case, it's there and for disciplined developers using RCS it may provide some additional means of tracking changes to the widget set.

Following that the superclass public include file is then included. Careful attention is paid to insure the path to the include file is correct since it is dependent on what widget set defines the superclass. Unfortunately, it is always assumed that the widget set either has the include

files directly under */usr/include/X11* or in a subdirectory named after the widget set. This does not seem to offer a problem, although time will tell. This is also conditionally included in an attempt to make compilation slightly more efficient but this is only a feature and not a necessary step.

Following this is an alphabetical listing of all resources defined by this class and all of it's superclasses. The list includes the resource name, class, type and default value. Most importantly, the list is accurate and complete since the same information is used to generate the resource declarations.

Next comes the declaration of the widget class variable.

Finally, the new resources and classes declared by this widget are defined using the approved syntax of `#define␣XtN<name>␣"<name>"` for resource names and `#define␣XtC<Name>␣"<Name>"` for class names. Note that the use of a single space, '␣,' is significant. Each of these definitions is itself wrapped in an `ifdef` to insure the value is not defined differently elsewhere.

Following is an example from the *Label.h*:

```
#ifndef _XO_LABEL_H_
#define _XO_LABEL_H_
/*
 * $Id:$
 *
 * $Log:$
 */
#ifndef _XO_SIMPLE_H_
#include <X11/Xo/Simple.h>
#endif

/*
 * Name          Class                 RepType        Default Value
 * ——           ——-                  ——-            ————-
 *                          *** Core ***
 * accelerators  Accelerators          XtTranslations NULL
 * ancestorSensitive Sensitive         Boolean        XtCopyAncestorSensitive
 * background    Background            Pixel          XtDefaultBackground
 * backgroundPixmap Pixmap             Pixmap         XtUnspecifiedpixmap
 * borderColor   BorderColor           Pixel          XtDefaultForeground
 * borderPixmap  Pixmap                Pixmap         XtUnspecifiedPixmap
 * borderWidth   BorderWidth           Dimension      0
 * colormap      Colormap              Colormap       XtCopyFromParent
 * depth         Depth                 Cardinal       XtCopyFromParent
 * destroyCallback Callback            XtCallbackList NULL
 * height        Height                Dimension      0
 * mappedWhenManaged MappedWhenManaged Boolean        True
 * screen        Screen                Screen *       XtCopyScreen
 * sensitive     Sensitive             Boolean        True
 * translations  Translations          XtTMRec        XtInheritTranslations
 * width         Width                 Dimension      0
 * x             Position              Position       0
 * y             Position              Position       0
 *                          *** Simple ***
```

```
 * backingStore     BackingStore              int             "NotUseful"
 * borderClass      Class                     WidgetClass     "None"
 * borderWidget     BorderWidget              Widget          NULL
 * cursor           Cursor                    Cursor          "cross"
 * foreground       Foreground                Pixel           XtDefaultForeground
 * inheritSet       InheritSet                XtCallbackList  NULL                    40
 * insensitiveStipple InsensitiveStipple Pixmap               NULL
 * traverseAccept TraverseAccept             Boolean         (XtPointer) True
 *                                  *** Label ***
 * drawClass        Class                     WidgetClass     "ObjText"
 * drawWidget       DrawWidget                Widget          NULL
 * gravity          Gravity                   XoGravity       "Center"
 */
extern WidgetClass                 xoLabelWidgetClass;

typedef struct _XoLabelClassRec  *XoLabelWidgetClass;                              50
typedef struct _XoLabelRec       *XoLabelWidget;

#ifndef XtNdrawClass
#       define XtNdrawClass "drawClass"
#endif
#ifndef XtNdrawWidget
#       define XtNdrawWidget "drawWidget"
#endif
#ifndef XtNgravity
#       define XtNgravity "gravity"                                                60
#endif
#ifndef XtCDrawWidget
#       define XtCDrawWidget "DrawWidget"
#endif
#ifndef XtCGravity
#       define XtCGravity "Gravity"
#endif
#ifndef XtRGravity
#       define XtRGravity "Gravity"
#endif                                                                             70

#endif /* _XO_LABEL_H_ */
```

There are some problems that need to be addressed. In particular, the addition of public methods and structures need to be addressed. Also, the possible use of a widget set wide include file to define many common features should also be accomodated.

9.1.2 Private Include File

The private include file is used by widget writers to define the internal structures used by the widget. This file should be included only by widget writers for the class or any subclasses and never by applications. (The `P` stands for Private and not for public).

The generated file follows the same convention as used in the public include file. The file is wrapped in *ifdef*'s, RCS id and log variables are defined, the private include file of the

parent class is included and then the public include file of this class is included. Note that this scheme causes the entire widget hierarchy to be included.

The class and instance records are defined next. Each of these has two components: the partial record and the concatenation of partial records from all the superclasses. We first define the class and then the instance pieces.

Following is the private include file for the *Label* widget.

```
#ifndef _XO_LABELP_H_
#define _XO_LABELP_H_
/*
 * $Id:$
 *
 * $Log:$
 */
#ifndef _XO_SIMPLEP_H_
#include <X11/Xo/SimpleP.h>
#endif                                                          10
#ifndef _XO_LABEL_H_
#include <X11/Xo/Label.h>
#endif

/*
 * Label's class record.
 */
typedef struct
}
        XtProc          set2;           /* Called from bogus() action */    20
} XoLabelClassPart;
/*
 * Full class record for Label widget.
 */
typedef struct _XoLabelClassRec
{
        CoreClassPart     core_class;       /* the Core class variables */
        XoSimpleClassPart simple_class;     /* the Simple class variables */
        XoLabelClassPart  label_class;      /* the Label class variables */
} XoLabelClassRec;                                              30

extern XoLabelClassRec  xoLabelClassRec;

/*
 * Label's instance record.
 */
typedef struct
{
        XoGravity       gravity;        /* how to place the label */
        WidgetClass     draw_class;     /* Widget class that draws */       40
        Widget          draw_widget;    /* Widget for drawing text. */
} XoLabelPart;

/*
 * Label's complete instance record.
```

```
*/
typedef struct _XoLabelRec
{
        CorePart        core;           /* the Core instance variables */
        XoSimplePart    simple;         /* the Simple instance variables */                50
        XoLabelPart     label;          /* the Label instance variables */
} XoLabelRec;

#endif /* _XO_LABELP_H_ */
```

Remember that all the above is mechanically generated.

9.1.3 Initialization Include File

The final code generated is used to define any actions, prototypes for any class methods; the translations and initialize the class variables for this widget class.

By declaring this in a seperate file, one hides a considerable amount of code and makes examining a widget implementation far less formidable. In the case of the *Button* widget, it hides 163 lines of initialization code. The other reason for declaring it in a seperate file is that the *build* program need not perform any special editing on a file to add this code fragment.

Like all the other include files, it is wrapped in *ifdef*'s, and has the RCS keywords. A terrible kludge is then purpotrated to allow the list of actions to always have a constant value in the definition of the class variables (something similar should be done with translations but it is so terrible I couldn't repeat it).

Two defines, *ACTIONS* and *ACTIONNUM* are added. If the class defines any actions, they are set to "*actions*" and "*XtNumber(actions)*", respectively. If no actions are defined, they are set to *NULL* and *0*, respectively. The difficulty arises because one cannot declare zero sized arrays whichi s what would happen if there are no actions defined. The default value defined for *actions* and *num_actions* in the class record for *Core* could have set these to NULL and 0, but it seemed like most widget sets would find real values more conveniant.

The prototypes for the action routines are then declared, properly encased in the *XoProto()* macro. The array that maps the name to the function pointer is then defined.

Next the translations for this widget are defined.

Next, the resources defined by this widget class are added.

Finally, the actual widget class is initialized. Care is taken so that the default value for each field is used unless it is overriden by the defining class. All the class methods are properly prototyped.

```
#ifndef _XO_LABELREC_H_
#define _XO_LABELREC_H_
/*
```

```
 * $Id:$
 *
 * $Log:$
 */
#define ACTIONS NULL
#define ACTIONNUM 0
#define offset(field) XtOffset(XoLabelWidget, label.field)
static XtResource resources[] = {
        {XtNgravity, XtCGravity, XtRGravity, sizeof(XoGravity),
         offset(gravity), XtRString, "Center"},
        {XtNdrawClass, XtCClass, XtRObjectClass, sizeof( WidgetClass),
         offset(draw_class), XtRString, "ObjText"},
        {XtNdrawWidget, XtCDrawWidget, XtRPointer, sizeof( Widget),
         offset(draw_widget), XtRImmediate, NULL},
};
#undef offset
XoProto (static void,     ClassInit, (void));
XoProto (static void,     ClassPartInitialize, ( WidgetClass class));
XoProto (static void,     Redisplay, ( Widget gw, XEvent * event, Region region));
XoProto (static void,     Initialize, ( Widget request, Widget new,
                        ArgList arglist, Cardinal* num_args));
XoProto (static XtGeometryResult,
                        GeometryQuery, ( Widget widget,
                                XtWidgetGeometry* request,
                                XtWidgetGeometry* reply));
XoProto (static void,     Resize, ( Widget gw));
XoProto (static Boolean, SetValues, ( Widget old, Widget request, Widget new,
                        ArgList arglist, Cardinal* num_args));
XoLabelClassRec xoLabelClassRec = {
        { /* Core */
                ( WidgetClass) &xoSimpleClassRec, /* superclass */
                "Label",                          /* class_name */
                sizeof(XoLabelRec),               /* widget_size */
                ClassInit,                        /* Core.class_initialize */
                ClassPartInitialize,              /* Core.class_part_initialize */
                False,                            /* class_inited */
                Initialize,                       /* Core.initialize */
                NULL,                             /* initialize_hook */
                XtInheritRealize,                 /* realize */
                ACTIONS,                          /* actions */
                ACTIONNUM,                        /* num_actions */
                resources,                        /* resources */
                XtNumber(resources),              /* num_resources */
                NULLQUARK,                        /* xrm_class */
                True,                             /* compress_motion */
                True,                             /* compress_exposure */
                True,                             /* compress_enterleave */
                False,                            /* visible_interest */
                NULL,                             /* destroy */
                Resize,                           /* Core.resize */
                Redisplay,                        /* Core.expose */
                SetValues,                        /* Core.set_values */
                NULL,                             /* set_values_hook */
                XtInheritSetValuesAlmost,         /* set_values_almost */
                NULL,                             /* get_values_hook */
```

```
                NULL,                           /* accept_focus */
                XtVersion,                      /* version */                   60
                NULL,                           /* callback_private */
                NULL,                           /* tm_table */
                GeometryQuery,                  /* Core.query_geometry */
                NULL,                           /* display_accelerator */
                NULL,                           /* extension */
        },
        { /* Simple */
                XtInheritSet,                   /* set */
        },
        { /* Label */                                                           70
                XtInheritSet,                   /* set2 */
        },
};
WidgetClass xoLabelWidgetClass = (WidgetClass) &xoLabelClassRec;

/*
 * The following make it easier to access class methods.
 */
#define MySuperClass xoSimpleClassRec
#define MyClass xoLabelClassRec                                               80
#define ThisSuperClass(gw) (*(XoLabelClassRec *)XtSuperClass(gw))
#define ThisClass(gw) (*(XoLabelClassRec *)XtClass(gw))

#endif /* _XO_LABELREC_H_ */
```

9.1.4 Documentation

The documentation is then written. The first piece of data is a table of all the resources defined by this class. This list is alphabetized by the instance name of each of the variables.

Following that is a lengthier description of each of the resources and instance variables. Each resource has the name of the corresponding instance variable added to the description.

Instance variables that do not have any corresponding resource (private instance variables) are also described but have `\ifinternal` conditions to insure they are not added to the user documentation.

Next all the actions defined by this widget are described.

Then finally the translations table and the intended meaning of each entry is described. Combined with the action table, this should make customization of the translation mechanism much easier.

9.2 Input Files

The description of the widget classes are placed in one or more files. Currently, *intrinsics.spec* defines the widget classes as defined by the Xt Intrinsics. Most classes are defined with the

exception of the some of the Shell subclasses.

The OpenWidgets are defined in four files organized by which intrinsic superclass behaviour is inherited from. These are *core.spec* for simple widgets, *rectobj.spec* for gadgets, *composite.spec* for manager widgets (as well as the MenuShell), and *constraint.spec* for manager widgets with data. There is no signficance to what file or what order the files are read.

A class is defined as a set of variables, a class_record, instance variables, actions, translations, and functions. The method for defining all these is ad hoc mostly because each represents an experiment to determine the most conveniant form. Time has not permitted them to be unified.

The grammar for describing a class is:

```
class      : class <class_name> = { class_info }
class_info : assignment
           : class_record
           : instance_record
           : constraints_record
           : actions
           : translations
```

9.2.1 Variables

Variables are set with:

```
name = value ;
```

Each variable is only defined within the class. In other words, setting a variable does not change its value in any other class. Several variables, such as *nocode* and *nosuperclass* are flags that just by assigning anything to them sets the flags. They are typically set to True for stylistic reasons but any value such as "false" or the empty string "" has the same effect.

superclass The class to inherit behaviour and resources from. Some classes, such as Core and Object, set this to the empty string. There must be a corresponding class as set with <class_name>.

purpose A description of what this widget does. This is included as the introductory documentation for the widget.

class_name Establishes the actuall name of the class.

nocode If set, indicates that none of the .h files should be generated. This is used for all the Intrinsic classes.

class_prefix The prefix to use before all variables, structures and typedefs. A 'X' is prepended to this value. Setting this also implies that all include files are found in:

`/usr/include/X11/Xclass_prefix/filename.`

public_pre_defines
This is included at the beginning of the public .h file. Ideally, most things needed are calculated by the *build* program. However, many things such as structures, typedefs or other include files used in implementing a widget need to be defined. This and the following several variables can be used to effect the contents of the public and private include files.

public_post_defines
This is included at the end of the public .h file.

private_pre_defines
This is included at the beginning of the private P.h file.

private_pre_defines
This is included at the beginning of the private P.h file.

private_include
Several of the intrinsic widgets are declared in include files that are not derivable from the class name. An example is the OverrideShell which is defined in *Shell.h* and *ShellP.h*. Setting this gives the base name, the ".h" and "P.h" are appended automatically.

public_include
Similar to private_include except for the public include file.

guard Each `#include` statement is carefully wrapped in `#ifdef`'s to avoid multiple inclusion and also increase efficiency. Widgets generated by *build* have very regular forms for determining if a file has already been included. Unfortunately, the Intrinsics are irregular about the name used to guard against multiple inclusion. This allows one to override *build*'s default rules for specifying the guard.

guardP same as guard except for the private include file.

nosuperclass indicates this class does not inherit any of the class variables from the superclass – but does inherit the instance variables. This is needed specifically for RectObj since it inherits fields from Object but none of the class variables. Note that Core gets around this because it does not truly inherit from Object or RectObj but conveniantly defines the same fields in the same location.

superclass_rec
Set internally by *build* to allow initializations to refer to the class record.

instance_rec_type
: The type of the instance record. Set internally by *build* so the default initializations work.

constraint_rec_type
: The type of the classes constraint record. Set internally by *build* so the default initializations work.

9.2.2 Class Record

9.2.3 Instance Record

9.2.4 Constraint Record

9.2.5 Actions

9.2.6 Translations

9.2.7 Functions

Chapter 10

Testing

Currently only a very simple test program exists. It is interactive and intended only to test the geometry management of the *Box* widget, popup and pulldown menus. A standard set of tests need to be developed for simple and composite widgets. These tests should be non-interactive and written with the intention of breaking the widgets. Following is a brief sketch of what such tests may incorporate.

10.1 Simple Widgets

The following tests are executed by a work proc and can be applied to any simple, non-composite widgets.

Destruct Continuosly create and destroy a single widget. The number of destructions should be a parameter. Keep track of the amount of memory after the first widget is created and insure it does not change after the test is completed. This is to insure there are no memory leaks. Report the time elapsed and the time per creation and destruction.

SetValues Allow a series of resources to be specified with alternate values. Run through the list of values alternating between values. After each pair of values has been tried, no additional memory should be leaking. Try changing each value a number of times as controlled by some parameter (different then the number of destruct times). Perhaps incorporate a call to XtGetResourceList() to verify most of the resources are being tested. Incorporate tests from parent classes. Allow some standard resource types to be used, such as color and font that can run through some normal set of values.

Geometry Make sure the *query_geometry* function is working properly by using XtSetValues() to change the width and height and using *query_geometry* to make sure it returns a different value.

Each individual widgets may also have some tests that can be run non-interactively. Perhaps a table of function pointers, names and private data for the test can be used. Allow the control of things such as repetition, timing and status to be controled by a driver function.

10.2 Composite Widgets

We might be able to execute identical tests for composite widgets except we may need to add several children for the composite to actually work. Also, some widgets such as *Menu* are probably difficult to test with this method since it requires interactive work to actually provide the events (too much work to simulate the events).