ORDERING INFORMATION

UNIX® SYSTEM V RELEASE 4 DOCUMENTATION

To order single copies of UNIX® SYSTEM V Release 4 documentation, please call (201) 767-5937.

ATTENTION DOCUMENTATION MANAGERS AND TRAINING DIRECTORS:
For bulk purchases in excess of 30 copies, please write to:

Corporate Sales
Prentice Hall
Englewood Cliffs, N.J. 07632

Or call: (201) 461-8441.

ATTENTION GOVERNMENT CUSTOMERS:
For GSA and other pricing information, please call (201) 767-5994.
Contents

1 Introduction
  Preface 1-1

2 The OPEN LOOK Toolkit
  The X Window System and the Xt Intrinsics 2-1
  The OPEN LOOK Widgets 2-2
  Widget Functions and Applications 2-18
  Text Selection Operations 2-50

3 Programming Using the OPEN LOOK Toolkit
  Introduction 3-1
  How to Write OPEN LOOK Programs 3-2
  Navigating Through the System 3-4
  Annotated Sample Programs 3-6
  Using Flattened Widgets 3-29
  Programming Caveats 3-36

4 X Window System, Version 11, Conventions for OPEN LOOK
  Introduction 4-1
  General Considerations 4-2
  Window Properties 4-6
  Relationship to Inter-Client Conventions 4-14
  Workspace and File Manager Conventions 4-21
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miscellaneous Implementation Issues</td>
<td>4-27</td>
</tr>
<tr>
<td><strong>5</strong> Mouseless Operations</td>
<td></td>
</tr>
<tr>
<td>Overview</td>
<td>5-1</td>
</tr>
<tr>
<td>Keyboard Traversal</td>
<td>5-2</td>
</tr>
<tr>
<td>Keyboard Accelerators</td>
<td>5-5</td>
</tr>
<tr>
<td>Mnemonics</td>
<td>5-7</td>
</tr>
<tr>
<td><strong>6</strong> Internationalization</td>
<td></td>
</tr>
<tr>
<td>Introduction to Internationalization</td>
<td>6-1</td>
</tr>
<tr>
<td>The Internationalization of the OPEN LOOK Toolkit</td>
<td>6-5</td>
</tr>
<tr>
<td><strong>7</strong> Extensive Widget Sampler Program</td>
<td></td>
</tr>
<tr>
<td>Design Objectives</td>
<td>7-1</td>
</tr>
<tr>
<td>Program Description</td>
<td>7-3</td>
</tr>
<tr>
<td><strong>A</strong> Manual Pages: Introduction</td>
<td></td>
</tr>
<tr>
<td>Introduction to the Manual Pages</td>
<td>A-1</td>
</tr>
<tr>
<td>Introduction to General Resources</td>
<td>A-4</td>
</tr>
<tr>
<td><strong>B</strong> Manual Pages: Convenience Routines</td>
<td></td>
</tr>
<tr>
<td>Introduction to the Convenience Routines</td>
<td>B-1</td>
</tr>
<tr>
<td>Convenience Routines</td>
<td>B-6</td>
</tr>
</tbody>
</table>
# Table of Contents

## Manual Pages: Widgets
- Introduction to the Widgets
- Widgets

## Manual Pages: Obsolete Routines
- Introduction to the Obsolete Routines
- Obsolete Routines

## Glossary
- Glossary

## Index
- Index
Figures and Tables

Figure 5-1: Sub-Object Traversal within a Container Object 5-3
Figure 5-2: Oblong Button with and without an Accelerator Visual 5-6
Figure 5-3: Oblong Button with and without a Mnemonic Visual 5-8
Figure 6-1: EUC Code Set Representations 6-2
Figure 6-2: EUC and Corresponding 16-bit Wide-character Representation 6-3
Figure 6-3: EUC and Corresponding 32-bit Wide-character Representation 6-4
Figure 6-4: Four kinds of Input Methods 6-21
Figure 7-1: The Open Look Widget Sampler 7-2
Figure C-1: Sample Manual Page Format C-3
1 Introduction

<table>
<thead>
<tr>
<th>Preface</th>
<th>1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>1-1</td>
</tr>
<tr>
<td>Overview</td>
<td>1-1</td>
</tr>
<tr>
<td>Document Conventions</td>
<td>1-2</td>
</tr>
</tbody>
</table>
Preface

Purpose

The purpose of this manual is to provide a guide for programmers using the OPEN LOOK® Graphical User Interface (GUI). The book is intended for skilled C language programmers, intending to develop applications using a “windowed” front-end. While the OPEN LOOK GUI toolkit is built on top of the X Window System™, you do not need to be familiar with that system to be able to make effective use of the OPEN LOOK toolkit.

The intention of this manual is to guide programmers in the use of the toolkit. Reference material and manual pages are also found in this manual.

Overview

Chapter 2 of this manual provides an overall description of the OPEN LOOK toolkit. The description includes background information on the X Window System toolkit, called the Xt Intrinsics. The Intrinsics provide an opaque interface to the X Window System and some of the Intrinsic functions are necessary for using the OPEN LOOK toolkit. Of particular importance is the notion of what a “widget” is.

This provides a basis for describing the specific OPEN LOOK abstractions and a brief description of each of the OPEN LOOK widgets and important widget attributes (called “resources”).

Chapter 2 also describes the OPEN LOOK functions and how they are generally used.

Chapter 3 provides a brief description of how to program using the OPEN LOOK toolkit; how to lay out a screen; how to create and manage widgets and “callbacks.” Each of the sample programs that are distributed with the OPEN LOOK toolkit are presented and described. The applications are presented in increasing complexity.

Chapter 4 presents conventions for using the OPEN LOOK toolkit so that applications can co-exist and interoperate.

Chapter 5 discusses mouseless operations.
Chapter 6 presents an extensive example program that demonstrates the use of most of the OPEN LOOK widgets.

**Document Conventions**

The OPEN LOOK® Graphical User Interface Programmer’s Guide uses certain typographical conventions, such as boldface and italics, to identify different types of information. The following conventions apply:

- Commands and pathnames that must be typed on the computer exactly as shown appear in constant width.

- Variables to those commands and pathnames appear in italic. For example, in the command

  ```
  xwd -out file
  ```

  *file* will be any name that you select as the file name to be entered.

- Routines, widget names, and procedure names appear in constant width.

- Structure member names appear in italic.

- Computer output such as prompts and messages appear in computer style type. Text references to parts of a displayed program or routine also appear in this type style. Resource names, resource classes, and widget classes also appear in this constant width style.
## The OPEN LOOK Toolkit

### The X Window System and the Xt Intrinsics

2-1

### The OPEN LOOK Widgets

2-2

- Flat Widgets and Gadgets
  - Flat Widgets
  - Gadgets
- Naming Conventions for Widgets
- Resources for Widgets
- Getting and Setting Widget Resources
- Basic Widget Resources
- Specifying Resources for Flattened Widgets
  - Sub-Object Resource Lists
  - Inheriting Resources
  - Ordering Resources in the List
  - Resources for Specifying Sub-Objects in a Flat Widget
- Screen Resolution and Color
  - Low and High End Color
  - Device Resolutions
- Supported Fonts
  - Standard Font
  - Automatic Choice of Font for Resolution

2-15

### Widget Functions and Applications

2-18

- OPEN LOOK Widget Descriptions
  - Action Widgets
  - Text Control Widgets
  - Container Widgets
  - Popup Choices

2-19

Table of Contents
# Table of Contents

**OPEN LOOK Routines**  
- Necessary Xt Intrinsic Routines 2-44  
- OPEN LOOK Initialization 2-45  
- Registering Help 2-46  
- Packed Widgets 2-46  
- Error Handling Routines 2-46  
- Controlling Input Focus 2-47  
- Flat Widget Routines 2-48  
- Convenience Routines 2-49  
**Applications with Multiple Base Windows** 2-49  

**Text Selection Operations** 2-50  
- Setting Insert Point 2-50  
- Wipe-through Selection 2-50  
- Adjusted Selection 2-50  
- Multi-click Selection 2-51  
- Copying Text 2-51  
- Cutting Text 2-51  
- Pasting Text 2-51  
- Selecting Text with the Keyboard 2-51
The X Window System and the Xt Intrinsics

The X Window System is a comprehensive mechanism for creating and managing a window environment. Access to the X Window system is usually through an Application Programmer Interface (API) toolkit, such as the OPEN LOOK Toolkit. The base level of the X Window System, Xlib, is a collection of C routines that perform fundamental screen and window management operations. The application programmer would use Xlib directly, for example, for drawing lines, arcs, rectangles, and so on.

The API toolkit can be separated into two distinct layers:

- **Xt Intrinsics**
- **Widgets**

The fundamental layer of the API toolkit is the Xt Intrinsics (Xt is shorthand for X Toolkit). The Xt Intrinsics are a set of C routines that monitor events related to end-user interactions and dispatch the correct code to handle those events.

A major function of the Xt Intrinsics is to provide for the creation and management of "Widgets". A widget is a set of code and data that provides a certain "look and feel" to an end-user. A widget defines a rectangular area of a screen that complies with an application interface policy, such as the OPEN LOOK GUI. A widget is a user-interface component combining an X window with the necessary semantics to form an object. The object provides an intuitive user-interface abstraction, such as a **button** or a **scrollbar**.

The Xt Intrinsics contain facilities to create, customize, organize, and destroy widgets. They also translate event sequences from the window server into procedure calls, which the application can then interpret. The intrinsics keep track of the state of a particular widget and negotiate screen real estate when a widget changes size or position.
The OPEN LOOK Widgets

The OPEN LOOK Toolkit provides the application programmer with a defined set of widgets and other user-interface abstractions. The application programmer is not primarily concerned with defining widgets. You are concerned, rather, with defining the layout of the screen for a particular application and designing the code to manage specific end-user interactions with the widgets. The OPEN LOOK API also gives you the ability to customize applications by defining sub-classes of the OPEN LOOK widgets. This is achieved by giving you direct access to the underlying C structures that define the widgets.

Widgets have certain attributes, called resources. At the programming interface level a resource is a named data item, a named component of a structure definition. For example, some of the resources associated with widgets are the background color, the parent widget (all widgets have a parent widget, where the "topmost" parent is the application's base window), the height and width of the defined area, and so on.

Some widgets exist only to define an area in which other widgets can be defined; that is, they are Composite Widgets and exist only as parents to child widgets. For example, the Bulletin Board widget simply provides a space to attach other widgets to. A widget with no children is a Primitive Widget. Primitive widgets are directly associated with an action: they perform a function, enter data, or output data. They do not contain other widgets.

Each time you specify a widget, you can also register the name of the routine(s) you have written to process that widget; that is, you pass the routine name(s) to the Xt Intrinsics. These application registered routines are termed "callbacks." Callbacks manage the semantics of an end-user interaction. The Xt Intrinsics also monitor application registered, non-graphical events and dispatch application routines to handle them. These features allow application programmers to use this implementation of an OPEN LOOK Toolkit in database management, network management, process control, and other applications requiring response to external events.

NOTE

The widgets as outlined in this guide, when implemented with the X Toolkit Intrinsics in XWIN Release 4i, meet Level 1 compliance with the OPEN LOOK specification and some Level 2 features, such as the use of color and menus with more than one type of control.
Flat Widgets and Gadgets

The OPEN LOOK Toolkit provides two other abstract interface components:

- Flat Widgets
- Gadgets

These abstractions are very much like widgets. They are designed to enhance the performance of OPEN LOOK applications by saving time and space. They minimize the space used by redundant window definitions.

For the most part, the term widget will be used to refer to any of widgets, flat widgets or gadgets.

Flat Widgets

A Flat Widget is a single widget that maintains a collection of similar user-interface components (sub-objects). The flat widget gives the appearance and behavior of many widgets. The sub-objects could be defined as widgets, but, because they share the same basic characteristics, defining them collectively results in improved performance. Flattened widgets, for instance, consume only a fraction of the memory that an equivalent hierarchy of widgets requires.

In this version of OPEN LOOK there are three flat widgets:

1. Flat Exclusives, containing the equivalent of RectButton widgets.
2. Flat NonExclusives, also containing the equivalent of RectButton widgets.
3. Flat Checkboxes, equivalent to a nonexclusives widget populated with Checkbox widgets.

In general, flat widgets (or flattened widgets) have the following attributes:

- They are container objects, responsible for managing the look and feel of one or more sub-objects.
- After the container is populated, minimal or no manipulation is done on the sub-objects.
- Each container is simply a region that contains zero or more sub-objects of a certain type.
The sub-objects within the container do not have an associated window or widget structure.

From the end-user's perspective, there is no discernible difference between a flattened widget interface and the same interface defined as a composite widget with child widgets.

From the application programmer's perspective, flattened widgets have a different interface than traditional widgets or gadgets. Flattened widgets are more efficient and easier to deal with, particularly when the sub-objects are regularly spaced and have similar attributes. A single toolkit request can specify an arbitrary number of sub-objects to the flattened widget, thus achieving a substantial reduction in the lines of code required to produce a complex graphical interface component. Additionally, there is a single callback routine for all sub-objects of the flattened widget.

Chapter 3 includes a detailed description of programming using flattened widgets.

### Gadgets

A Gadget is a windowless object; that is, it is a widget that uses its parent's window. For the most part, there is no difference in the application programmer's handling of a gadget and of the same object defined as a widget. The only difference is in the widget class (for example, an OblongButton Widget belongs to the **OblongButtonWidgetClass** while the OblongButton Gadget belongs to the **OblongButtonGadgetClass**).

It is important to note that a gadget is a subclass of the RectObj class while a widget is a subclass of Core, with a gadget having only a subset of the resources found associated with the Core class. Thus care should be taken to avoid referencing fields which exist for widgets but not gadgets, both in writing gadget code or in converting widget code to gadget code in existing applications.
MenuButton gadgets cannot be parents (that is, cannot be used as the parent parameter when creating a widget or other gadget).

Gadgets share some core fields. But since they are not subclasses of Core, they do not have all Core fields. In particular, they don’t have a name field or a translation field (so translations cannot be specified/overridden).

Naming Conventions for Widgets

The OPEN LOOK Toolkit programmer may end up dealing with hundreds of programming particles: widgets, widget resources, Xt intrinsic routines, OPEN LOOK routines, and so on. A well-defined set of naming conventions has been adopted to simplify the process of reading and writing applications. The naming conventions are important, and are essential to effective toolkit programming and debugging.

The names of structure members are lower-case; underscores are used to join compound words. Examples of these names are tag, display, and id_type.

Type and procedure names begin with upper-case letters. Capitalization is used to separate the components of compound words. Examples of this style are:

- XtGetValues
- XtSetArg
- XtSetValues
- ArgList
- OlInitialize

Because of the object-oriented nature of the toolkit, all data structures and most data types are type-defined (“typedef-ed”). Further stylization is often accomplished using typedefs for derivatives of these private data types.

Additional conventions stipulate that:

- A resource name will have the prefix XtN. Using this convention, the resource name XtNbackgroundPixmap will associate with a structure member background_pixmap.
A resource class will have the prefix Xtc. For example, the resource class Background will be defined as XtcBackground.

Basic widget naming conventions can, therefore, be thought of according to the following list:

- Intrinsics procedure names begin with “Xt”
- OPEN LOOK routines start with “Ol”
- Resource names begin with “XtN”
- Resource class names begin with “Xtc”

These conventions are like those used for X Window’s Xlib

**Resources for Widgets**

A resource is a named data item. This section describes how default values for resources are established. The next section describes how these values are set and obtained after the widget is created. There is a fundamental distinction between the nature of a widget, or its “Widget Class” (for example, an Oblong-Button), and a specific widget, or an instance of a widget (for example, This Blue OblongButton Over Here). Widgets may be defined as sub-classes of other defined widgets, which are then considered to be their “super-class” widget.

The resource value may be set by a program, specified by a user, or specified as a default. For example, in the case of the OblongButton widget, some of the resource items that are specific to a given instant of the widget include foreground color, text font, the label string, and label justification. Some of these resource elements may come from the widget’s superclass, such as background color, border color, and border width.

During initialization of the OPEN LOOK Toolkit (through a call to the OlInitialize routine), the resources are merged from several sources in the order shown below. During the merge, unrelated resources are simply added to the complete set, while overlapping resources are replaced with the latest values. For instance, if the resource XtNforeground is specified in several sources, the value of XtNforeground from the last source overrides earlier values.
Resource values may be supplied to an application from the following sources, in the order given:

1. **Internal defaults** - Each object in the toolkit has object-specific defaults for the resources it uses. These defaults are in effect unless overridden from another source.

2. **Application defaults** - The application-specific resource file name is constructed from the class name of the application and points to a site-specific resource file that usually is installed by the site manager when the application is installed. On UNIX-based systems, the application resource file is `/usr/X/lib/app-defaults/class`, where *class* is the application class name.

3. **Server resource or .Xdefaults file** - The next source is the X server’s `RESOURCE_MANAGER` property, as returned by the `XOpenDisplay` routine. If no such property exists, the `.Xdefaults` file in the user’s home directory, if it exists, is loaded in place of the server property. However, when the OPEN LOOK Workspace Manager (`olwsm`) is running, the `RESOURCE_MANAGER` property always exists, and obtains its values from the `.Xdefaults` file. This essentially makes moot the differences between these alternate sources.

   **NOTE** A user can make changes to the `.Xdefaults` file as long as the `olwsm` program is *not* running. Changes made while the Workspace Manager is running will be lost.

   An application can use the `olwsm` program to store changes to resource values. The changes are stored immediately in both the `.Xdefaults` file and the `RESOURCE_MANAGER` property.

4. **XENVIRONMENT or .Xdefaults-host file** - The user’s environment variable `XENVIRONMENT` provides the name of the file for the next source. If this environment variable is not set, the file `.Xdefaults-host` in the user’s home directory, if present, is used, where *host* is the name of the user’s host machine.

5. **Command line options** - The next source is the command line, where the user can give several standard and application-specific options. The `OlInitialize` routine has a table of the standard command line options for adding resources to the resource database, and it takes as a parameter additional application-specific resource abbreviations. See "Parsing the
Command Line” in Chapter 4 of the *X Toolkit Intrinsics* for the format of this table.

6. Application overriding values - Resource values set within the application program are the last source. These assignments will override any, and all, previous assignments. See the next section, “Getting and Setting Widget Resources”, for details on how the application can set resource values.

**Getting and Setting Widget Resources**

The intrinsics provide procedures for obtaining a widget’s current resource values and assigning values to a widget’s resources. These functions are used after a widget is created.

The values can be set at initialization time, can be set with a call to - *XtSetValues*, can be read with a call to *XtGetValues*, or can be set in other ways (see the reference sections of this guide for specific information).

The following table describes the arguments for the *XtSetValues* and *XtGetValues*.

<table>
<thead>
<tr>
<th></th>
<th><strong>XtSetValues</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Widget</strong></td>
<td>Specifies the widget.</td>
</tr>
<tr>
<td><strong>args</strong></td>
<td>Specifies a variable length argument list of the name/value parts to be modified.</td>
</tr>
<tr>
<td><strong>num_args</strong></td>
<td>Specifies the number of entries in the argument list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th><strong>XtGetValues</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Widget</strong></td>
<td>Specifies the widget.</td>
</tr>
<tr>
<td><strong>args</strong></td>
<td>Specifies a variable length argument list of name/address pairs. The address part is the address of an object of the type given as name.</td>
</tr>
<tr>
<td><strong>num_args</strong></td>
<td>Specifies the number of entries in the argument list.</td>
</tr>
</tbody>
</table>

Both functions require the number of arguments (num_args) to be passed as a parameter. The function *XtNumber(array_name)* can be used to return the number of entries in a fixed length array.
Basic Widget Resources

This section describes those widget resources that belong to the Core Widget. The Core Widget contains the definitions of resources that are common to all widgets. All widgets are subclasses of Core.

A description of these resources are briefly presented here and in more detail later in the reference sections of this Guide. These resources are common to all widgets and you should consult the reference manual before actually using them.

<table>
<thead>
<tr>
<th>Resource Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
</tr>
<tr>
<td>XtNbackground</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
</tr>
<tr>
<td>XtNborderColor</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
</tr>
<tr>
<td>XtNborderWidth</td>
</tr>
<tr>
<td>XtNdepth</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
</tr>
<tr>
<td>XtNheight</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
</tr>
<tr>
<td>XtNsensitive</td>
</tr>
<tr>
<td>XtNtranslations</td>
</tr>
<tr>
<td>XtNwidth</td>
</tr>
<tr>
<td>XtNx</td>
</tr>
<tr>
<td>XtNy</td>
</tr>
</tbody>
</table>
The Access column is interpreted as follows:

S  Value can be set by XtNsetValues
G  Value can be read by XtNgetValues
I  Value can be set at initialization
*  Value set in other ways

**XtNancestorSensitive**

Range of Values:

- TRUE
- FALSE

This argument specifies whether the immediate parent of the widget will receive input events.

**XtNbackground**

Range of Values:

- Any pixel value valid for the current display

This resource specifies the background color of the widget.

**XtNbackgroundPixmap**

The application can specify a pixmap to be used for tiling the background. This takes precedence over XtNbackground.

**XtNborderColor**

Range of Values:

- Any pixel value valid for the current display

This resource specifies the color of the border.

**XtNborderPixmap**

The application can specify a pixmap to be used for tiling the border. This takes precedence over XtNborderColor.
The OPEN LOOK Widgets

**XtNborderWidth**

Range of Values:

\[ 0 \leq \text{XtNborderWidth} \leq \min (\text{XtNwidth}, \text{XtNheight}) / 2 \]

This resource sets the width of the border for a widget. A width of zero means no border will show.

**XtNdepth**

Range of Values:

\[ \text{0 or (any value supported by the current display)} \]

Determines how many bits should be used for each pixel in the widget’s window. The value of this resource is used by the Xt Intrinsics to set the depth of the widget’s window when it is created.

**XtNdestroyCallback**

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

**XtNheight**

Range of Values:

\[ 0 \leq \text{XtNheight} \]

This resource contains the height of the widget’s window (in pixels), not counting the border area.

**XtNmappedWhenManaged**

Range of Values:

TRUE

FALSE

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

Range of Values:
- TRUE
- FALSE

This resource determines whether a widget will receive input events. Note that in the table this resource is marked *. In order to set this resource, you should use the function **XtSetSensitive**. This is because this resource affects the status of any child widgets and the **XtSetSensitive** function will propagate the new value to all children.

**XtNwidth**

Range of Values:
- \( 0 \leq \text{XtNwidth} \)

This resource contains the width of the widget’s window in pixels, not including the border area.

**XtNtranslations**

This resource should not be set by an application.

**XtNx, XtNy**

These resources contain the x-coordinate and y-coordinate of the widget’s upper left hand corner, excluding the border, relative to its parent widget.

## Specifying Resources for Flattened Widgets

A flattened widget has three kinds of resources:

- Resources that affect the container object; for example, **XtNcontainerType** or **XtNgravity**, specifying how the sub-objects in this container are positioned within another container.
- Resources that must be specified for each sub-object; for example, 
  *XtNlabel*, specifying the label that goes inside the button or checkbox.

- Resources that may be applied to every sub-object or may be applied indi-
  vidually to each sub-object; for example, height, width, font, background 
  color, and so on.

This means that resource specification must allow for variation in the number of 
specified resources (that is, where the number of specifications is a function of 
the number of sub-objects) and also allow for variation in specifying which 
resources will be defined for the container and which will be defined for the 
sub-objects. This variation is determined by using resource lists for the 
specification.

**Sub-Object Resource Lists**

The resources for the sub-objects of a flat container are specified in list for-
mat. This allows the programmer to determine which container resources will 
be inherited by the sub-objects and which will be specified individually.

Each list is an array of application-defined records (typically, in a “C” structure 
format or as an array), where each record describes a particular sub-object. The 
items to be entered on the list are determined by the application programmer. 
In the same application, different flat containers can have different lists. For a 
given flat container all items on the list are specified in the same format. That 
is, for efficiency reasons, each record in the array must have the same form as 
the other records in the array; each structure in the list has identical fields.

Every different flat container widget may have a different set of fields for each 
record in its sub-object resource list. For example, if an application wanted to 
specify an “unselect” callback procedure for one group of exclusives but not for 
another, the application would specify an *XtNunselectProc* field as an element 
field for the first list but not for the second list. For data alignment and parsing 
reasons, the fields of each record must use the *XtArgVal* type (see the code 
example in Chapter 3).
Inheriting Resources

Since all sub-object resources are part of a container (parent), sub-objects inherit any non-specified resource from the parent container. For example, if the application requires a particular font color for all sub-objects, you do not have to specify the XtNfontColor resource for each sub-object; you simply set the font color resource on the parent container and all sub-objects will use that font color.

Though sub-object resources are part of its container’s resource set, none of the sub-object resources have any direct effect on the container.

Ordering Resources in the List

Since the “form” of the sub-object record is defined within the application’s domain, the container must be given a hint about the record’s form so that it can parse the supplied list. A resource name list is the key to unlocking the application’s sub-object list. While the ordering of fields in each record is not important, the application must give the resource names in the same sequence that their associated values appear in the record.

For example, if the records specifying sub-objects of a flat exclusives container had a “XtNlabel” field followed by the “XtNselectProc” callback field, the application must supply the container with the XtNlabel resource name followed by the XtNselectProc resource name. Inconsistent ordering of the fields will result in undefined behavior when the sub-objects are instantiated.

Resources for Specifying Sub-Objects in a Flat Widget

Five common resources are used by each container class to describe the necessary sub-object information:
The OPEN LOOK Widgets

Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNitems</td>
<td>XtCItems</td>
<td>XtPointer</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNnumItems</td>
<td>XtCNumItems</td>
<td>Cardinal</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemFields</td>
<td>XtCItemFields</td>
<td>String *</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNnumItemImageFields</td>
<td>XtCNumItemImageFields</td>
<td>Cardinal</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemsTouched</td>
<td>XtCItemsTouched</td>
<td>Boolean</td>
<td>SG</td>
</tr>
</tbody>
</table>

XtNitems
This is the list of sub-object items.

XtNnumItems
Specifies the number of sub-object items in the list.

XtNitemFields
Contains the list of resource names used to parse the records in the XtNitems list.

XtNnumItemImageFields
Specifies the number of resource names in XtNitemFields.

XtNitemsTouched
Values are TRUE or FALSE (the default). Whenever you modify an item list directly, you must set this as TRUE to the flat widget container so that it can update the visual.

Screen Resolution and Color

Low and High End Color

This implementation of an OPEN LOOK toolkit supports the low and high end color representation, that is, the primary colors, as well as the fine shadings.

If you wish to fine-tune your color choices, you can edit the color resources in your .Xdefaults file. Note that you should only do so when OPEN LOOK is not currently running.
The OPEN LOOK Widgets

<table>
<thead>
<tr>
<th>RESOURCE</th>
<th>AFFECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Background</td>
<td>windows and widget background</td>
</tr>
<tr>
<td>*TextBackground</td>
<td>TextEdit, xterm</td>
</tr>
<tr>
<td>*TextFontColor</td>
<td>TextEdit, TextField, ScrollingList, xterm</td>
</tr>
<tr>
<td>*inputFocusColor</td>
<td>cursor, traversal, highlighting</td>
</tr>
<tr>
<td>*inputWindowHeader</td>
<td>header of window with input focus</td>
</tr>
<tr>
<td>olwsm.workspace</td>
<td>workspace color</td>
</tr>
<tr>
<td>*foreground</td>
<td>window title, window menu label, buttons, scrollbars sliders</td>
</tr>
<tr>
<td>*borderColor</td>
<td>border of widgets and windows</td>
</tr>
<tr>
<td>*FontColor</td>
<td>TextEdit, TextField and ScrollingList widgets and xterm</td>
</tr>
</tbody>
</table>

Refer to the resource sets of widgets in the “Manual Pages: Widgets” appendix of this guide.

Device Resolutions

The table below presents the screen formats that are supported with this toolkit. Assuming 11 inch diagonal and 13 inch diagonal (nominal) monitors are used, the table also shows the corresponding pixel densities. The numbers are given in pixels per inch horizontally by pixels per inch vertically.

<table>
<thead>
<tr>
<th>Adapter Type</th>
<th>Format</th>
<th>11&quot; Monitor</th>
<th>13&quot; Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGA</td>
<td>640 x 350</td>
<td>76 x 55</td>
<td>66 x 48</td>
</tr>
<tr>
<td>EGA (AT&amp;T Extended)</td>
<td>640 x 400</td>
<td>76 x 63</td>
<td>66 x 55</td>
</tr>
<tr>
<td>VGA</td>
<td>640 x 480</td>
<td>76 x 76</td>
<td>66 x 66</td>
</tr>
<tr>
<td>Enhanced VGA</td>
<td>800 x 600</td>
<td>95 x 95</td>
<td>83 x 83</td>
</tr>
</tbody>
</table>

On screens that meet these resolutions, the visuals presented by the toolkit adhere to the 12-point size visuals required by the OPEN LOOK specification. However, the toolkit will work with any resolution. On low resolution screens, such as the CGA format, the visuals will be larger than 12 points; on higher resolutions, the visuals will be smaller than 12 points.
Supported Fonts

Standard Font
The defined OPEN LOOK font "Lucida" is used in labels for all labeled controls and as the default font in text widgets.

Automatic Choice of Font for Resolution
The toolkit automatically selects the correct default font to match the resolution of the device. The application can override the font selection, but then the toolkit does not automatically adjust the font to accommodate a change in screen resolution.

Note that fonts must be cached on a per screen basis.
Widget Functions and Applications

This section describes the OPEN LOOK widgets and processing routines. The widgets defined for the OPEN LOOK GUI are listed below:

Action Widgets
- OblongButton
- RectButton
- CheckBox
- MenuButton (was: ButtonStack)
- AbbrevMenuButton (was: AbbrevStack)
- Slider
- Scrollbar
- Stub

Text Control Widgets
- StaticText
- Text
- TextField

Container Widgets
- BulletinBoard
- ControlArea
- Form
- Caption
- FooterPanel
- Exclusives
- Nonexclusives
- FlatCheckbox
- FlatExclusives
- FlatNonExclusives
- ScrolledWindow
- ScrollingList

Popup Choices
- Notice
- PopupWindow
- Menu
OPEN LOOK Widget Descriptions

**NOTE** A reference to a widget class is formed by using the lower case widget name followed by "WidgetClass," such as `menuButtonWidgetClass` and `menuWidgetClass`.

**Action Widgets**

**OblongButton Widget and Gadget**

The **OblongButton** widget is a primitive widget consisting of a label surrounded by a rounded oblong border (Figure 1).

![Figure 1. Oblong Button Widgets](image)

In a series of Buttons, the default choice is indicated by a double border. The **OblongButton** provides a single-action control for the end-user. The user initiates an action each time the Button is selected, unless the Button is marked as "busy" or "inactive".
The **OblongButton** is typically incorporated into a **Menu, ControlArea**, or other composite widget as part of a set of controls. Because of this, we recommend you use an **OblongButton Gadget** instead. This saves both time and space.

**RectButton**

The **RectButton** is a primitive widget consisting of a label surrounded by a rectangular border. Figure 2 illustrates Rectangular Buttons under several possible conditions, and using different borders to indicate the condition. While a **RectButton** may be used alone, it is generally used as a component child of an **Exclusives or Nonexclusives** composite widget. The **RectButton** is used to indicate a choice (for example, to set or reset a switch value) in the application.

![Figure 2. Rectangular Button Widgets](image)
**CheckBox**

The `CheckBox` widget consists of a label followed by a check box. The widget acts as a toggle switch. The first time it is selected a check mark is drawn in the box. Selecting it again removes the check mark. If it is initially set, the first time it is selected it will be un-set.

`CheckBox` widgets may be used alone or as part of another composite, but they are almost always used as children of the `Nonexclusive` composite widget. The `Nonexclusives` widget manages the appearance of the `CheckBox`. These widgets provide the end-user with a way to make one or more selections from a list of choices in a form that looks like items being checked off a list.

Figure 3 illustrates a `CheckBox` and specifies each of its components.

![Image of CheckBox widget](image.png)
**MenuButton Widget and Gadget**

The **MenuButton** (ButtonStack in Release 1.0) is a composite widget that provides the look of a regular **OblongButton**, or "stack" of OblongButtons. The stack of buttons acts as a visual representation for a menu. It has all the features that the **Menu** has and also provides quick access to the menu defaults, both for selection and previewing.

Figure 4 illustrates a **MenuButton**. The inverted triangle (also known as a Menu mark) visually distinguishes a **MenuButton** from a primitive Button.

The **MenuButton** is normally defined as a Gadget for more efficient performance.

---

**Figure 4. MenuButton Widget**
AbbrevMenuButton

The AbbrevMenuButton (AbbrevStack in Release 1.0) composite widget is similar to the MenuButton control. It appears on the screen without a label, taking up less screen space. The AbbrevMenuButton widget can be used with other widgets to allow the end user to add new items to the menu.

Figure 5 illustrates an AbbrevMenuButton widget next to a Current Selection Widget. The Current Selection widget is independently created by the application. Typically, you would place both widgets side by side in a composite widget. The AbbrevMenuButton widget only automatically previews the default selection in the selection widget; the application is responsible for showing any other selections.

Abbreviated Menu Button

AbbrevMenuButton Widget

Current Selection Widget

Figure 5. AbbrevMenuButton Widget
Widget Functions and Applications

Slider
The **Slider** widget provides the graphical equivalent of an analog control, allowing the end-user to move a slider element to a position that represents a value along a continuum.

The application specifies the minimum and maximum values as well as the granularity. Figures 6A and 6B illustrate both a horizontal and vertical **Slider** widget and labels each of its components.

![Slider Widget Diagram](image)

**Figure 6A. Slider Widget (Horizontal)**
Figure 6B. Slider Widget (Vertical)
Scrollbar

The **Scrollbar** widget has no intrinsic function; it is always associated with an adjoining window, that contains the "Content" to be scrolled.

The Content is composed of units of data (for example, lines of text), that typically exceed the size of its window (called a window pane). The end-user scrolls through the Content by selecting and moving the **Scrollbar**. In appearance, the **Scrollbar** looks like an elevator or a cable car that moves back and forth (or up and down) on a cable. The Content moves through the pane proportionately to the movement of the **Scrollbar**. Figures 7A and 7B illustrate different aspects of **Scrollbars**, showing their components and illustrating fully extended **Scrollbars**, both horizontal and vertical. **Scrollbars** can also be abbreviated (when they are attached to a small window, for example). In this case the cable component is eliminated.

![Scrollbar Widget](image)

**Figure 7A. Scrollbar Widget (Horizontal)**
Figure 7B. Scrollbar Widget: Elevator and Proportion Indicator at Left/Top Limits

The Scrollbar does not provide semantics for scrolling through any particular content, but provides an interface for an application or another widget, such as the ScrolledWindow and widgets, to implement a scrollable window pane.

Stub

The Stub widget is designed to give you increased flexibility in controlling the screen during execution. It allows you to specify procedures at creation and/or XtSetValues time that are normally restricted to a widget’s class part. Most of the class part procedures have been attached to the instance part.

This allows you to use Stub widgets to build local widgets (that is, to design application-specific widgets). The Stub is particularly useful for drawing graphics.
Text Control Widgets

StaticText

The StaticText widget implements the OPEN LOOK Message control to provide an uneditable display. The application has some flexibility in laying out the text within the StaticText widget.

Figure 8 illustrates a typical StaticText widget, showing the resources used to control the display coloration.

```
XtNfontColor

The quick brown fox jumped over the lazy widget.

XtNborderColor
(XtNborderPixmap)

XtNbackground
(XtNbackgroundPixmap)
```

Figure 8. StaticText Widget

TextEdit

The TextEdit widget provides an interface for the end user to enter and edit text. It provides a basic set of editing controls, including selection control for text, copy, move, and cut and paste.

TextEdit widgets are often used in conjunction with ScrolledWindows and Scrollbars to manage text that cannot fit within the provided window frame.
**TextField**

The **TextField** widget provides a one-line text field, with an application-set width, for letting the end user enter and edit text. Controls are included for scrolling long text left and right within the field. Figure 9 illustrates a **TextField** widget with various controls and those resources used to control the coloration. The left and right arrows are used to indicate the direction of text overflow; the vertical triangle is the editing cursor.

![TextField Diagram](image)

**Figure 9. TextField Widget**
Container Widgets

BulletinBoard

The BulletinBoard is a composite widget; that is, it contains and manages other widgets (which may themselves be composites). The BulletinBoard widget provides minimal management of its widget children, simply providing a space into which they can be placed. It allows the application to configure the overall size of the "bounding box" around the child widgets, but does not provide facilities for ordering the widgets. When defining a BulletinBoard you can specify that it have a fixed screen size, that it have a minimal size (just large enough to hold its children), or that it grow to meet its children's needs.

Figure 10 illustrates a BulletinBoard and shows the coloration resources.

![Diagram of BulletinBoard Widget](image)

**Figure 10. BulletinBoard Widget (Coloration)**
**ControlArea**

The **ControlArea** is a composite widget that manages the layout of its children on the screen. It allows the controls to be laid out in one of four patterns and provides column or row alignment. The four patterns are:

- Fixed number of rows in the control area
- Fixed number of columns
- Fixed overall width of the control area
- Fixed overall height

The child widgets are automatically laid out left to right and then top to bottom in the order that they are added to the **ControlArea** composite widget. Typical layouts, coloration and representative resources are illustrated in Figures 11A and 11B.

**Figure 11A. ControlArea Widget (Coloration)**
Figure 11B. ControlArea Widget (Height and Width Control)
Form

The **Form** composite widget provides more sophisticated management of other widgets than the **BulletinBoard** composite. It allows the application to specify the layout of the widgets, so that their position relative to each other remains consistent in the face of window resizing done by the end user. A **Form** widget is illustrated in Figure 12.

![Figure 12. Form Widget (Coloration)](image)
Caption

The Caption widget provides a convenient way to label other controls. It allows the application to choose the placement of a text label next to one widget. The Caption widget is illustrated in Figure 13. Some OPEN LOOK composite widgets recognize the Caption widgets and will align them specially.

![Caption Widget Diagram](image)

Figure 13. Caption Widget

FooterPanel

The FooterPanel composite widget provides a convenient way of getting a footer at the bottom of an OPEN LOOK window. It allows the application to use any widget best suited for the content of the footer and handles the resize management of the footer and the pane above it.
**Exclusives**

The *Exclusives* is a composite widget that allows the end-user to select one, and only one, of a series of choices. The *Exclusives* composite is only used together with *RectButton* primitive widgets as children (Figure 14).

The *Exclusives* widget provides the layout and selection control of the *RectButton* widgets, ensuring that the controls align in columns and/or rows, that only one (or none) of the buttons is set at one time, and that dimmed buttons are reset if a choice is made.

*Exclusives* can also be defined as Flattened Widgets. When defined in this way its children will not be treated as individual widgets. For the most part, you should use the Flattened Widget definition to improve overall performance.

![Figure 14. Exclusives Widget Example](image)
Nonexclusives

The Nonexclusives is a composite widget that allows the end-user to select one or more of a series of choices. The Nonexclusive widget can manage either RectButton or CheckBox primitive widgets.

The Nonexclusives widget provides the layout and selection control of the RectButton or CheckBox widgets, ensuring that the controls align in columns and/or rows.

When the container will be populated with RectButtons, the Nonexclusives can also be defined as Flattened Widgets. When defined in this way its children will not be treated as individual widgets. Use the Flattened Widget definition to improve overall performance.

Flat CheckBox

The FlatCheckBox widget is equivalent to a NonExclusives widget that is populated entirely by Check Boxes. The advantages of using this flattened widget definition is that only a single widget is defined and that resource values of the container can be inherited by each of the sub-objects.
ScrolledWindow

The ScrolledWindow widget can be used as the basis for implementing a scrollable pane, for example, for a TextEdit widget. However, it has no innate text or graphics semantics; it must be combined with other widgets for this.

To use the ScrolledWindow you create a widget capable of displaying the entire Content as a child of the ScrolledWindow widget. The ScrolledWindow widget positions the child "within" the view of the Content and creates scroll bars for the horizontal and vertical overflow as needed.

Figure 15 illustrates a ScrolledWindow widget, showing the content (which does not appear on the screen) and the View of the Content (which does). The vertical and horizontal ScrollBar widgets are separately defined and are attached to the ScrolledWindow widget.

Figure 15. Scrolled Window Widget
ScrollingList

The ScrollingList widget is another kind of window pane, where the end-user manipulates the list with a Scollbar. The list consists of identifiable items; that is, there is a "current item" that the end-user selects that is highlighted in the pane. The list is completely under the control of the application. You can turn selected items into Exclusives or NonExclusives. You can define selected items to be editable text. You can add, change or delete items from the list. You use the callback routines to determine what action, if any, to take with the current item. Figure 16A illustrates common components of a "typical" ScrollingList widget and Figure 16B shows the same widget configured to add a new component to the List.

Figure 16A. Common ScrollingList Widget Components
Figure 16B. Adding an Element to a ScrollingList

Attaching a Menu to a ScrollingList

A menu can be added to a ScrollingList Widget but it requires some work on the part of the application. The code required by the application to create a menu is shown below:

```c
/* Create Menu */
cnt = 0;
XtSetArg(args[cnt], XtNmenuAugment, False); cnt++;
menu = XtCreatePopupShell("menu", menuShellWidgetClass, list_widget, args, cnt);

/* Add callback to catch MENU button */
XtAddCallback(list_widget, XtNconsumeEvent, PopupMenuCB, menu);

/* Get Menu Pane */
XtSetArg(args[0], XtNmenuItemPane, &menuPane);
XtGetValues(menu, args, 1);

/* Add buttons to menu pane */
cnt = 0;
XtSetArg(args[cnt], XtNaccelerator, "Ctrl<c>"); cnt++;
XtSetArg(args[cnt], XtNmemonic, 'c'); cnt++;
```
change = XtCreateManagedWidget("change", oblongButtonGadgetClass,
        menuPane, args, cnt);
XtAddCallback(change, XtNselect, EditCB, NULL);

cnt = 0;
XtSetArg(args[cnt], XtNaccelerator, "Ctrl<d>"); cnt++;
XtSetArg(args[cnt], XtNmemonic, 'd'); cnt++;
delete = XtCreateManagedWidget("delete", oblongButtonGadgetClass,
        menuPane, args, cnt);
XtAddCallback(delete, XtNselect, DeleteCB, NULL);
}

static void
PopupMenuCB(w, closure, call_data)
    Widget w,
    XtPointer  closure, call_data;
{
    OIVirtualEvent  ve = (OIVirtualEvent) call_data;
    Position        x, y;

    /* Use OIMenuPopup(w, state, setpos, x, y, pos_proc) */
    /* to pop-up menu.  See manual page for more details. */

    switch(ve->virtual_name)
    {
    case OL_MENU :
        ve->consumed = True;
        /* Let Menu determine (x,y) */
        OIMenuPopup((Widget)closure, OL_PRESS_DRAG_MENU, False, 0, 0, NULL);
        break;
    case OL_MENUKEY :
        ve->consumed = True;
        /* calculate (x,y) here */
        ...
        OIMenuPopup((Widget)closure, OL_PRESS_DRAG_MENU, True, x, y, NULL);
        break;
    }
}
Popup Choices

Notice

The **Notice** widget implements a high-priority pop-up window. Once a **Notice** is popped up the end-user must respond to it before they can continue on in that application (though they can exercise other applications that may be sharing the screen). **Notices** are usually used for messages such as “Are you sure you want to do that?”, coupled to Yes/No buttons (see Figure 17).

The **Notice** widget automatically handles the window-level creation and management. It also provides a text widget interface for registering the text to present to the end user and a control area widget for attaching the buttons the user needs to dispatch the **Notice**.

![Figure 17. Notice Widget](image-url)
PopupWindow

The PopupWindow widget is used to allow you to create Command and Property windows. A Command Window is a popup window that allows you to solicit parameters for a command. A Property Window is a popup window that is typically used to allow the end-user to specify overall properties of some aspect of an application. This widget handles the window-level creation and management, leaving the application to populate the interior of the window with controls. Figure 18 shows the various components of a PopupWindow widget.

![Diagram of PopupWindow widget components]

Figure 18. PopupWindow Widget
Menu
The Menu composite widget manages a set of widgets that comprise the items in the menu. It arranges for the proper response of the menu items to user-generated events like press or click SELECT as opposed to press or click MENU, and oversees the setting of new defaults by the end user. This widget also arranges to automatically "pop up" on the screen when the user presses or clicks MENU on the parent widget, by augmenting the way events are handled for its parent widget.

A Menu consists of a set of items presented to the end-user for selection. One of the items is always considered to be the Default item. Menus have a Title, a Separator, separating the title from the items, a Border and a Pushpin. The Pushpin is used by the end-user to control whether the menu stays up after it is used (if the Pushpin is set) or if it pops back down again. Figure 19 illustrates a typical Menu widget.

![Figure 19. Menu Widget](image-url)
OPEN LOOK Routines

In addition to a widget set, the OPEN LOOK toolkit provides a number of routines to support you in using the widgets and managing your application. The OPEN LOOK initialization routine is mandatory; the remainder provide additional control, flexibility and convenience in developing applications.

This section presents those routines; it simply presents the function name and describes its functionality. The reference manual describes its parameters and usage.

The first part of this section reviews those Xt Intrinsics routines that are necessary within the OPEN LOOK toolkit.

Necessary Xt Intrinsic Routines

This section presents a minimal set of Xt Intrinsics routines that are necessary to develop OPEN LOOK applications. There are a significant number of other routines that are essential for developing applications that create graphic images or sophisticated text, varying text fonts and sizes. These are not described here, though a few of them are used in the program "s_sampler" presented in Appendix A.

XtSetArg

Many intrinsic routines need to be passed pairs of resource names and values. These are passed as an argument list. This function specifies which pair in the list to set and what the name and value are.

XtNumber

This function is used to get the number of elements in a fixed array (such as the size of a resource table).

XtCreateWidget
XtCreateManagedWidget
XtCreatePopupShell

These are the functions used to create specific entities on the screen. The call gives the class of the widget (that is, the kind of widget), its label, and so on.
XtDisplay
XtPopup

These functions are used to control the display of widgets on the screen.

XtDestroyWidget

Used to destroy a temporarily created widget.

XtAddCallback

This function is used to register a callback routine with the intrinsics and to associate that callback with a specific widget.

XtRealizeWidget
XtMainLoop

These are generally the last two statements of your main program. All of the XtCreate... and XtSet... functions merely get the widgets ready. Nothing actually appears on the screen until the widgets are made real via the XtRealize widget function.

The last statement of your program is XtMainLoop. This turns control over to the X Window System to manage the end-user's interactions with the screen. The MainLoop manages interactions and dispatches your callback routines as requested by the end-user.

OPEN LOOK Initialization

The following routine must be used in an application in order for the application programmer's interface to function properly.

OlInitialize

This routine sets initial values needed by other routines and the widgets, and registers any resource converters used by the application programmer's interface.
Widget Functions and Applications

Registering Help

OlRegisterHelp

This routine associates help information with either a specific widget, or a widget class.

You can supply specific Help text with the call; you can name a file containing the Help text; or you can specify an application defined routine that can implement a more elaborate help procedure.

Packed Widgets

OlCreatePackedWidget

This routine lets the application programmer create a related tree of widgets in a single call. Child-Parent relationships are handled automatically.

Error Handling Routines

OlError
OlWarning
OlVaDisplayErrorMsg
OlVaDisplayWarningMsg
OlSetErrorHandler
OlSetWarningHandler
OlSetVaDisplayErrorMsgHandler
OlSetVaDisplayWarningMsgHandler

See the error(3W) manual page in the “Manual Pages: Convenience Routines” appendix of this guide.

Because of the non-procedural aspects of “widget programming”, there are no convenient error returns that an application can check. Currently, the default routines used are OlError and OlWarning. These print messages to the UNIX standard error channel to convey error conditions.

You can override the default messages by registering your own custom error messages. The OlSet... routines are used to do that.
The widgets attempt to continue running whenever they can. This allows you to discover multiple errors in a single run. They cannot recover from bad pointers, but when they encounter illegally set resource values, they use the default value and report on their actions. When unrecoverable errors occur, the widgets generate an error message and terminate the process.

Controlling Input Focus

These routines allow the application programmer to determine which widget is receiving input focus and which widget is next to receive input focus. The routines are:

- OlCallAcceptFocus
- OlGetCurrentFocusWidget
- OlMoveFocus
- OlCanAcceptFocus
- OlSetInputFocus
- OlHasFocus

The OlHasFocus routine returns true or false depending upon whether the designated widget currently has the input focus. The OlCanAcceptFocus determines whether a particular widget is able to be designated to get the input focus.

The OlCallAcceptFocus routine determines whether the widget can accept input focus and, if it can, moves the focus to it.

The OlMoveFocus routine shifts the focus to the next widget in a specified direction (current, next or previous).

Please see the Input_Focus manual page in the “Manual Pages: Convenience Routines” chapter of this guide.
Flat Widget Routines

These two routines are the equivalent of XtGetValues and XtSetValues for flattened widgets. The only distinction between the Xt routines and the Ol routines is that the index of the sub-object must be specified. These routines are illustrated in Chapter 3. The routines are:

- OlFlatGetValues
- OlFlatSetValues
- OlVaFlatGetValues
- OlVaFlatSetValues
- OlFlatCallAcceptFocus
- OlFlatGetFocusItem
- OlFlatGetItemIndex
- OlFlatGetItemGeometry

See the Flattened Widget Utilities(3W) manual page for more information.
Convenience Routines

The following routines are not a necessary functional part of the application programmer's interface, but they make it easier to get certain features:

- OlMMToPixel
- Ol_MMToPixel
- OlPointToPixel
- Ol_PointToPixel
- OlScreenMMToPixel
- Ol_ScreenMMToPixel
- OlScreenPointToPixel
- Ol_ScreenPointToPixel
- OlPixelToMM
- Ol_PixelToMM
- OlPixelToPoint
- Ol_PixelToPoint
- OlScreenPixelToMM
- Ol_ScreenPixelToMM
- OlScreenPixelToPoint
- Ol_ScreenPixelToPoint

The first set provides screen independence for sizing and layout of controls, giving conversions between millimeters and pixels, and points and pixels. The latter converts from virtual OPEN LOOK translations to standard X Toolkit Intrinsics translations.

Applications with Multiple Base Windows

Every application has a single base window, automatically created by the OlInitialize call that initializes the toolkit. If your application needs additional base windows, they are created with a call to XtCreateApplicationShell. The widget class pointers that can be used here are transientShellWidgetClass or applicationShellWidgetClass.
Text Selection Operations

The `StaticText`, `TextEdit`, and `TextField` widgets use the following operations to copy and move text.

**Setting Insert Point**

Clicking SELECT sets the insert point at the boundary between two characters or spaces nearest the pointer. This makes an inactive caret active and highlights the header of the main window (base window or pop-up window) containing the specific text widget, to show which window has the input focus. Any active selection on the screen is deselected.

**Wipe-through Selection**

Pressing and dragging SELECT marks the bounds of a new selection and highlights it, and deselects any other active selection on the screen. While SELECT is pressed, the active or inactive caret that marks the insert point is invisible, but when SELECT is released, the insert point is left at the position of the release. This does not make the insert point (caret) active if it isn’t already active.

The selection starts with the character where SELECT is pressed and extends to the character where SELECT is released. If the pointer moves outside the widget and the widget can scroll in that direction (that is, there is a scrollbar for that direction), the widget scrolls additional text into the widget and adds it to the selection. The rate at which text scrolls into the widget is the same rate at which pressing SELECT on the arrows of the Scrollbar scrolls the widget.

Deletion of the new selection is pending: new text entered from the keyboard or pasted from the clipboard replaces the selection.

**Adjusted Selection**

Clicking SELECT, moving the pointer, and clicking ADJUST marks the bounds of a selection and highlights it. A subsequent click of ADJUST changes the end bound of the selection. The ADJUST may also follow a wipe-through selection. The selection starts with the character where SELECT was clicked and extends to the character where ADJUST is clicked. The insert point is moved to the position of the ADJUST. As above, deletion of the new selection is pending.
**Multi-click Selection**

Double-clicking SELECT selects the word nearest the pointer. In case of a tie, the word to the left is selected. Triple-clicking SELECT selects the entire line, and quadruple-clicking selects the entire content. The selection is highlighted and the insert point is left at the position of the multi-click.

Multi-Click Selection does not work with mouseless operations. Please see Chapter 7 in the OPEN LOOK User's Guide for more information about CUT, COPY, and PASTE using the keyboard.

**Copying Text**

Using COPY copies any selected text to the clipboard and deselects it.

**Cutting Text**

Using CUT moves any selected text to the clipboard and deletes it from the Input Field.

**Pasting Text**

After setting the insert point, using PASTE copies text from the clipboard as though it were typed in, leaving the insert point at the end of the pasted text. This will replace any text currently selected in the widget. Note that the data on the clipboard may have come from outside the Input Field, but it must be text. If the clipboard is empty, the system beeps.

**Selecting Text with the Keyboard**

The keyboard can be used to select and adjust text. The SELCHARFWD key adjusts the selection one character to the right of the insert point and moves the insert point one character to the right. The SELWORDFWD key adjusts the selection one word to the right of the insert point and moves the insert point to the end of the word. The SELLINEFWD key adjusts the selection from the insert point to the end of the line. The insert point moves to after the last character in the line but before the newline.
Similar keys adjust the selection backwards. The SELCHARBACK key adjusts the selection one character to the left of the insert point and moves the insert point one character to the left. The SELWORDBACK key adjusts the selection one word to the left of the insert point and moves the insert point to the beginning of the word. The SELLINEBACK key adjusts the selection from the insert point to the beginning of the line. The insert point moves to before the first character in the line.

The key SELLINE adjusts the selection to include the entire line in which the insert pointer is set. The insert pointer is positioned to the right of the last character in the line.

The SELFLIPENDS key moves the insert pointer from one end of the selection to the other without adjusting the selection.
3 Programming Using the OPEN LOOK Toolkit

Introduction 3-1

How to Write OPEN LOOK Programs 3-2
Object Oriented Programming 3-2

Navigating Through the System 3-4
Include File Directories 3-4
Libraries 3-4
Compilation Command 3-4
Public and Private Include Files 3-5

Annotated Sample Programs 3-6
Creating a Button Widget 3-7
  ▪ Program Description 3-8
Creating a Composite Widget 3-10
  ▪ Program Description 3-10
Creating a Menu 3-14
  ▪ Program Description 3-14
Excerpted Programming Examples from s_sampler 3-20
  ▪ Entering Data 3-20
  ▪ Using a Slider Widget 3-22
  ▪ Defining and Using a Stub Widget 3-24
  ▪ Using a Form Widget 3-26
# Table of Contents

## Using Flattened Widgets
- Specifying the Container Setting 3-29
- Callbacks and Flat Widgets 3-30
- Setting the State of a Sub-Object 3-31
- Getting the State of a Sub-Object 3-34
- Obtaining Help on a Sub-Object 3-34

## Programming Caveats
- Naming Conventions 3-36
- Macro Alerts 3-36
- Callback Restrictions 3-36
- Global Name Space Restrictions 3-37
- Debugging Hints 3-37
Introduction

This chapter provides a practical introduction to how to develop programs using the OPEN LOOK Toolkit. It consists of five sections:

- How to Write OPEN LOOK programs
- Navigating Through the System
- Annotated Sample Programs
- Using Flattened Widgets
- Programming Caveats

The fundamental assumption of this chapter is that you are a competent C Programmer, though not necessarily familiar with programming for the X Window System or Object Oriented programming.

The “How to Write OPEN LOOK Programs” section offers a brief walk-through of the steps necessary to get started.

The section on “Navigating Through the System” simply lays out the necessary #include file structure and the library structure for using the OPEN LOOK Toolkit.

The “Annotated Sample Programs” and “Using Flattened Widgets” sections present several source programs of increasing complexity that create widgets and associated callback routines. The intent of this section is to provide coding models that you can use that will prove to be the base for your own use of the OPEN LOOK toolkit.

The final section “Programming Caveats” presents tips and warnings.
How to Write OPEN LOOK Programs

Object Oriented Programming

In order to read and write Object Oriented code effectively, you need to be aware both of the style requirements and the design requirements.

You will notice the programming style differences immediately upon looking at a listing of OPEN LOOK code: you will be hard pressed to find ints, or longs, or chars. In Object Oriented Programming things are distinct; a widget is a Widget; a display is a Display. It is probably a good idea (in general) to look through the include files and familiarize yourself with the style used to define and develop the components you work with. After a while the specific typing of objects will become familiar and you will get over the strangeness and appreciate the increased clarity.

The design requirements are critical. You will need to change how you think about your application. Object Oriented Programs are not sequential. They are not well-ordered. The program flow is not a function of the input data stream; rather, it is entirely a function of the end-user’s choosing.

Object Oriented programs are usually "event driven." Control is given to a routine as a function of an event, usually associated with an action taken by the end-user. In the case of an OPEN LOOK application, the event is a combination of a particular mouse (or key) action on a particular screen widget. Usually, it is the selection of a particular option, though the end-user may have requested help on that widget, may have requested a menu, and so on. Routines are initiated as a combination of a specific action on a specific object.

You need to know that it might not be a sensible action; not necessarily the correct next-thing-to-do. And you need to program for that. You must also be aware that because the actions are essentially asynchronous, it may be very hard to determine when errors have occurred. You will have to think about debugging your application taking this form of flow and usage into consideration.

In another way, programming is actually much easier. The notion of object orientation gives rise to small, relatively stand-alone routines. They can be written and tested outside of the full application system.

In the simplest sense applications are divided into two distinct parts: laying out the screen and programming the proper response to specific events. The first part consists of creating widgets; the second part consists of programming callbacks. Laying out the screen is completely separate from developing the callback routines. A useful way to proceed is to write your program using dummy
callbacks that do nothing but return. Once you are satisfied with the screen lay­
out and have tested and debugged the screen management you can add the call­
backs one at a time.

In the examples, the order of the code is:

- Include Files
- Necessary Global Data Definitions
- Callback Routines
- The Main Routine that Creates the Widgets
- Realizing the Widgets and Exiting to the Xt Main Loop

It was done this way for clarity of presentation. You should probably put the
screen layout section in a separate routine and thereby keep the code that much cleaner. It also means that the exposition is a bit backwards; the event driven
actions are described before the objects of the event are created.
Navigating Through the System

Include File Directories

The XWIN implementation of the OPEN LOOK Toolkit places all \#include files and libraries in the following directories:

- `/usr/X/include/Xol` holds all include files for the OPEN LOOK widgets
- `/usr/X/include/Xll` holds all include files for the X Toolkit Intrinsics
- `/usr/X/libl` holds all object libraries.

The names of the actual public and private include files for the OPEN LOOK widgets are listed at the end of this section.

Libraries

The names of the object libraries are:

**Object libraries:**
- `/usr/X/lib/Xll.a`
- `/usr/X/lib/libXt.a`
- `/usr/X/lib/libXol.a`

**Shared libraries:**
- `/usr/X/lib/Xll.so`
- `/usr/X/lib/libXt.so`
- `/usr/X/lib/libXol.so`

Compilation Command

The prototypical command for compiling an application built with the XWIN implementation of the OPEN LOOK GUI is the following for UNIX System V Release 4:

```
cc -I/usr/X/include -I/usr/X/include/Xol -I/usr/X/include/Xll -c ...
cc -o ... -L/usr/X/lib -lXol -lXt -lXmu -lXll -lnsl -ldl -lw
```
There is a dependency upon \texttt{libw -lw} from the MNLS package for Release 4. Compilation will not work for pre-Release 4 releases because there is no MNLS.

For UNIX System V Release 4, it is necessary to set the following flags as part of the compilation line:

\begin{verbatim}
-DUSG -Datt -DSYSV
\end{verbatim}

for Release 4 an additional flag must be added:

\begin{verbatim}
-DSVR4_0 -D__TIMEVAL__
\end{verbatim}

Note that the order of linking the libraries is important.

**Public and Private Include Files**

Include file names are limited to ten characters. For private include files the character before the “.h” is always “P”. The private files give you access to the internal widget class definitions used in the Toolkit. By having access to the underlying C structure definitions, you can extend the widget definitions and define customized, application-specific widgets.

In order to develop OPEN LOOK applications you need to include the appropriate widget class definition files and the basic Xt Intrinsics and OPEN LOOK files. The three fundamental files are:

\begin{verbatim}
/usr/X/include/X11/Intrinsic.h
/usr/X/include/X11/StringDefs.h
/usr/X/include/Xol/OpenLook.h
\end{verbatim}

For the names of the OPEN LOOK GUI public and private include files, see the widget manual page in the “Manual Pages: Widgets” appendix of this guide. Include files are listed at the top of the page.
Annotated Sample Programs

This section presents and analyzes source code that makes use of the OPEN LOOK Toolkit. It includes three stand-alone programs, and four excerpts from a comprehensive example, `s_sanpler`. In each of these examples, we specify where and how you can choose between defining an object as a widget and defining the same object as a gadget.

The three stand-alone programs describe the creation and use of:

- a button widget (named `s_button.c`)
- a composite widget (`s_composite.c`)
- and a menu widget (`s_menu.c`)

Chapter 6 contains the code for `s_sanpler`, that illustrates all of the objects in the OPEN LOOK toolkit. This chapter includes four excerpts from that example that describe the creation and use of:

- entering data using a textfield widget
- a slider widget
- using a stub widget for drawing
- and defining and placing objects on a form widget

Each program is preceded by a brief explanation. The source code then follows on the right-hand page with explanations of the source code appearing on the corresponding left-hand page.

The four stand-alone programs are also available on-line with OPEN LOOK. The programs are stored in the directory `/usr/X/lib/tutorial/Xol`. To compile one of these programs, such as `s_button.c`, enter that directory and execute the following commands for Release 4:

```
cc -I/usr/X/include -I/usr/X/include/Xol -I/usr/X/include/X11 -c s_button.c
cc -o s_button s_button.o /usr/X/lib/libXol.so /usr/X/lib/libXt.so
   /usr/X/lib/libX11.so -lnsl -lc
```
Creating a Button Widget

The first program introduced here is the easiest of the three programs. It creates a single Button, labeled Quit. When the end-user presses the button, the program terminates.
Program Description

Lines 1-4 include the header files required by this program. Every program that uses the OPEN LOOK interface requires the first two general header files, plus a specific header file (in this case, `OblongButt.h`) for each widget class used.

Lines 5-11 implement a callback function that is associated with the `OblongButton` widget (line 26) and is invoked when the user selects the oblong button. For consistency, all callback functions define the same set of parameters, even though, as in this example, they are not always used. This particular callback function simply causes the application program to exit. All cleanup of OPEN LOOK resources is handled automatically by the system and the associated screen image is deleted.

Lines 12-16 are the standard declarations for a C program and lines 17-19 declare the data objects used. The `args` array will be used to contain argument lists to widget creation functions (line 22). Its size is arbitrary, but should be large enough to hold the largest argument list used in the program and should allow for possible future growth of argument lists.

Line 20 initializes the system and creates a toplevel shell widget. This widget does not have a visible image on the screen. It serves as a root for the widget hierarchy about to be created.

Line 22 generates the argument list to be used in creating the `OblongButton` widget. The array `args` stores the argument list, which consists of name-value pairs. The variable `n` is used to count the number of arguments present. In this program, the name of the argument is `XtNlabel`, and its value is the string `Quit`. The syntax shown in these lines is a widely used standard for creating argument lists. In this example, all arguments other than the button label assume standard default values.

Lines 23-25 create the `OblongButton` widget, make it a child of the toplevel shell widget, and assign it to the widget variable `quitButton`.

Line 26 adds the callback function defined above to the (previously NULL) list of functions to be called whenever the `quitButton` widget is selected by the user.

Line 27 takes the widget hierarchy defined above, with toplevel as its root, and creates the associated windows and displays the image on the screen.

Line 28 invokes the main event loop, which processes events, such as user inputs through the mouse and keyboard, and invokes the appropriate callback functions.
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xol/OpenLook.h>
#include <Xol/OblongButt.h>

void QuitCallback(widget, clientData, callData)
    Widget widget;
    caddr_t clientData, callData;
    {
        exit(0);
    }

int main(argc, argv)
    int argc;
    char **argv;
    {
        Widget toplevel, quitButton;
        Arg args[10];
        int n;

        toplevel = OlInitialize("top", "Top", NULL, 0, &argc, argv);
        n = 0;
        XtSetArg(args[n], XtNlabel, "Quit"); n++;

        quitButton = XtCreateManagedWidget("button",
            oblongButtonWidgetClass, toplevel, args, n);

        XtAddCallback(quitButton, XtNselect, QuitCallback, NULL);

        XtRealizeWidget(toplevel);
        XtMainLoop();
    }
Creating a Composite Widget

This program is a relatively easy program, although it's slightly more complex than the previous Quit Button widget program. It introduces the composite ControlArea widget and puts two Button children in it. The commentary describes how to modify the program to use gadgets instead of widgets.

The first child Button is the Quit button from the previous program. The second child Button changes its label when pressed, demonstrating the use of XtSetValues on an existing widget.

Program Description

Lines 1-5 include the header files required by this program. This program uses two widget classes, oblong button and control area, and has a header file for each.

Lines 6-25 implement a callback function, which will be associated with the toggle button widget and will be invoked when the user selects the toggle button.

Lines 8-9 define the standard callback parameters, although only one of them (widget) will be used.

Line 11 declares an array that will contain an argument list (limited to 1 argument).

Lines 15-23 implement a switch statement that chooses a new label for the widget and updates the variable containing the toggle value based on the current value of the counter. A switch statement is used so that the program can be easily extended to cycle among more than the two states currently implemented.

Line 24 updates the label field of the specified widget based on the value set in the argument list in the switch statement above. The updated label is reflected in the screen image of the widget (in this example, the toggle button).

Lines 26-32 implement the same callback function described in the previous program.
Annotated Sample Programs

1 #include <X11/Intrinsic.h>
2 #include <X11/StringDefs.h>
3 #include <Xol/OpenLook.h>
4 #include <Xol/OblongButt.h>
5 #include <Xol/ControlAre.h>

6 void
7 ToggleCallback(widget, clientData, callData)
8 Widget widget;
9 caddr_t clientData, callData;
10 {
11    Arg     args[1];
12    static int value = 1;
13    int n;
14
15    n = 0;
16
17    switch (value) {
18        case 1:
19            XtSetArg(args[n], XtNlabel, "Two"); n++;
20            value++;
21        break;
22        case 2:
23            XtSetArg(args[n], XtNlabel, "One"); n++;
24            value--;
25    }
26    XtSetValues (widget, args, n);
27 }

28 void
29 QuitCallback(widget, clientData, callData)
30 Widget widget;
31 caddr_t clientData, callData;
32 {
33    exit(0);
34 }
Line 37 declares and names the widgets that are used in this application.

Line 40 initializes the OPEN LOOK Graphical User Interface and creates the `toplevel` shell widget.

Lines 41-46 create a control area widget as a child of the `toplevel` widget. The argument lists in lines 42 and 43 specify that child widgets placed in the control area will have a layout consisting of a fixed number of columns, and that the number of columns (`XtNmeasure`) will be 1. By default, the widgets placed in this column will have the same size.

Lines 47-52 create an oblong button widget (`quitButton`) as a child of the control area widget and add the `QuitCallback` function to the `quitButton` widget’s list of callback functions. This is the same widget that was presented in the previous program example.

Lines 53-57 create a second oblong button widget (`toggleButton`) as a child of the control area widget and add the `ToggleCallback` function to `toggleButton`’s list of callback functions. Based on the definition of the control area widget above, this button will be placed under the `quitButton`, forming a single column, and both buttons will have the same size.

To transform the two button widgets into button gadgets, you simply change lines 50 and 56 to read “`oblongButtonGadgetClass`”. This results in the buttons using the control area window and not needing the code required to maintain their own windows. This is more efficient in both time and space.

Line 59 takes the widget hierarchy defined above, with `toplevel` as its root, creates the associated windows, and displays the image on the screen.

Line 60 invokes the main event loop, which processes events, such as user inputs through the mouse and keyboard, and invokes the appropriate callback functions.
33 int main(argc, argv)
34 int argc;
35 char **argv;
36 {
37     Widget toplevel, control, toggleButton, quitButton;
38     Arg args[10];
39     int n;
40     toplevel = OlInitialize("top", "Top", NULL, 0, &argc, argv);
41     n = 0;
42     XtSetArg(args[n], XtNlayoutType, OL_FixedCols); n++;
43     XtSetArg(args[n], XtNmeasure, 1); n++;
44     control = XtCreateManagedWidget( "control",
45         controlAreaWidgetClass,
46         toplevel, args, n);
47     n = 0;
48     XtSetArg(args[n], XtNlabel, "Quit"); n++;
49     quitButton = XtCreateManagedWidget( "qbutton",
50         oblongButtonWidgetClass,
51         control, args, n);
52     XtAddCallback(quitButton, XtNselect, QuitCallback, NULL);
53     n = 0;
54     XtSetArg(args[n], XtNlabel, "One"); n++;
55     toggleButton = XtCreateManagedWidget( "tbutton",
56         oblongButtonWidgetClass,
57         control, args, n);
58     XtAddCallback(toggleButton, XtNselect, ToggleCallback, NULL);
59     XtRealizeWidget(toplevel);
60     XtMainLoop();
61 }

Programming Using the OPEN LOOK Toolkit 3-13
Creating a Menu

This program creates an automatic popup widget, using the `MenuButton` widget. The menu has three children, one of which is the `Quit` Button used in the previous two programs. The two remaining children share the same two select and unselect callback functions, but pass different numbers to the functions.

Program Description

Lines 1-7 include the header files required by this program. This program uses four widget classes and contains a specific header file for each.

Lines 8-14 define the `QuitCallback` function from the previous examples.

Lines 15-21 define a callback function that will print a message to the client’s standard out when it is invoked. This function is associated with the selection event of multiple widgets. The `clientData` parameter will contain a pointer to an integer value (typecast to `caddr_t`) identifying which widget invoked it.

Lines 22-28 define a callback function that is similar to the preceding function, except that it is associated with an `unselect` event, that is, the clearing of a previous selection.
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xol/OpenLook.h>
#include <Xol/MenuButton.h>
#include <Xol/Menu.h>
#include <Xol/Exclusives.h>
#include <Xol/RectButton.h>

void Quitcallback(widget, clientData, callData)
    Widget widget;
    caddr_t clientData, callData;
    {
        exit(0);
    }

void Selectcallback(widget, clientData, callData)
    Widget widget;
    caddr_t clientData, callData;
    {
        printf("Button %d selected
", *clientData);
    }

void Unselectcallback(widget, clientData, callData)
    Widget widget;
    caddr_t clientData, callData;
    {
        printf("Button %d unselected\n", *clientData);
    }
Lines 34-38 declare the widgets and the argument list for this application.

Line 39 initializes the OPEN LOOK Graphical User Interface and creates a top level shell widget called `toplevel`.

Lines 40-42 create a menuButton widget as a child of `toplevel`, and the menuButton widget creates a composite child widget where menu items can be attached.

Lines 43-45 obtain the widget ID of the menuButton's composite child widget and store this id into the `menupane` variable.
Annotated Sample Programs

22 void
23 UnselectCallback(widget, clientData, callData)
24 Widget widget;
25 caddr_t clientData, callData;
26 {
27     printf("Button %d unselected", *clientData);
28 }

29 void
30 main(argc, argv)
31 int argc;
32 char **argv;
33 {
34     Widget toplevel, MenuButton;
35     Widget menupane, exclusives, button1, button2, button3;
36     Arg args[10];
37     int n;
38     int n1, n2;
39     toplevel = OLInitialize("top", "Top", NULL, 0, &argc, argv);
40     MenuButton = XtCreateManagedWidget("MenuButton",
41                              MenuButtonDownClass,
42                              toplevel, NULL, 0);
43     n = 0;
44     XtSetArg(args[n], XtNmenuPane, &menupane); n++;
45     XtGetValues(MenuButton, args, n);
Lines 47-52 create an exclusives widget as a child of `menupane`. This exclusives widget will appear when the mouse menu button is clicked on the menuButton created above. The argument list specifies that child widgets placed in the control area will have a layout consisting of a fixed number of columns, and that the number of columns (`XtNmeasure`) will be 1.

Lines 53-68 create three rectangular button widgets as children of the exclusives widget. These rectButton widgets provide the user-selectable choices in the pop-up menu of the menuButton.

Lines 53-58 create a rectangular button with the label "ONE" and associate two callback functions with it. The first, `SelectCallback`, is invoked when the associated button is selected. The second, `UnselectCallback`, is invoked when the button is unselected, that is, the selection is cleared. Both functions are passed a value for the `callData` parameter that identifies the call as being associated with button 1.

Lines 59-64 are similar to lines 52-56, except the new rectangular button has the label "TWO," and its callback function is passed a value for `callData` identifying button 2.

Line 69 takes the widget hierarchy under `toplevel` and creates the associated windows and displays the image on the screen.

Line 70 begins the main event loop.
n = 0;
XtSetArg(args[n],XtNlayout, OL_FIXEDCOLS); n++;
XtSetArg(args[n],XtNmeasure, 1); n++;
XtSetArg(args[n],XtNrecalculateSize,(XtArgVal) TRUE); n++;
exclusives = XtCreateManagedWidget("exclusives",
exclusivesWidgetClass,
memupane,args,n);
n1 = 1;
button1 = XtCreateManagedWidget("ONE",
rectButtonWidgetClass,
exclusives, NULL, 0);
XtAddCallback(button1, XtNselect, SelectCallback, &n1);
XtAddCallback(button1, XtNunselect, UnselectCallback, &n1);
n2 = 2;
button2 = XtCreateManagedWidget("TWO",
rectButtonWidgetClass,
exclusives, NULL, 0);
XtAddCallback(button2, XtNselect, SelectCallback, &n2);
XtAddCallback(button2, XtNunselect, UnselectCallback, &n2);
button3 = XtCreateManagedWidget("EXIT",
rectButtonWidgetClass,
exclusives, NULL, 0);
XtAddCallback(button3, XtNselect, QuitCallback, NULL);
XtRealizeWidget(toplevel);
XtMainLoop();
}
Excerpted Programming Examples from s_sampler

This section presents four code pieces selected from the comprehensive programming example, *s_sampler*, that appears in Chapter 6. The line numbers used in the excerpts match those in the appendix so that you can refer to the program as a whole.

The example in the appendix illustrates all widgets, gadgets, and flats that comprise the OPEN LOOK Toolkit. The code can be used as a model for developing OPEN LOOK applications. It is worth mentioning that, with a comment density in excess of 25%, the application sampler takes less than 2000 lines of code.

**Entering Data**

This section describes how to develop a textfield widget that permits the end-user to enter data. This may be the most common end-user operation and so is a critical example.

Lines 587 - 601 define the textfield widget callback routine. The callback routine is invoked whenever the end user *completes* the entry, which is indicated by pressing RETURN.

In this case, the callback simply outputs the text that was entered. The value here is to see how the text is referenced. In the same way, you can capture the text, convert it (in case it were numeric), and store it for use by other routines.

Lines 763 - 776 create the labeled data entry field. Defining the size and position of the field is done elsewhere in the example.
static void textfieldCB(widget, clientData, callData)
{
    Widget widget;
    XtPointer clientData, callData;
    OlTextFieldVerify *tfv = (OlTextFieldVerify *) callData;
    char buf[MAXBUF];
    Arg arg;

    sprintf(buf, "Footerpanel: TEXTFIELD User Input: %s", tfv->string);
    FooterMessage(footer_text, buf);
    XtSetArg(arg, XtNstring, (XtArgVal) "");
    XtSetValues(widget, &arg, 1);
}

/*
 * Make a caption as a label/prompt for the TEXTFIELD.
 */

    i = 0;
    XtSetArg(arg[i], XtNlabel, (XtArgVal) "Textfield: type & type <return> : ");
    widget = XtCreateManagedWidget("caption",
        captionWidgetClass, popupcall, arg, i);
    widget = XtCreateManagedWidget("textfield",
        textFieldWidgetClass, widget, NULL, 0);

/*
 * Callback to "read" user input when <return> typed.
 */
    XtAddCallback(widget, XtNverification, textfieldCB, NULL);
Using a Slider Widget

This section describes how to develop and use a slider widget.

The use of the slider widget here is both simple and dramatic. The slider is calibrated to return an integer value over the range of colors for the hardware. This value is returned to the callback as "callData." It is used in the \texttt{XtSetValue} call to set the background color of a stub widget that had been defined to sit physically above the slider widget on the screen. The effect, therefore, is to change the color of the stub widget as the end-user moves the slider. The result is a pretty demonstration of the use of color in graphic applications.

Line 583 accesses the color value established by the slider. Lines 584 and 585 change the value of the background color resource.

Lines 1236 - 1263 define the resources for the slider and associate the callback with its movement.

The position of the slider on the form is defined elsewhere in the application.

574 \texttt{static void}
575 \texttt{sliderCB(widget,clientData,callData)}
576 \texttt{Widget widget;}
577 \texttt{XtPointer clientData,callData;}
578 \{ \texttt{Arg arg;}
579
580 \texttt{/*}
581 \texttt{/* Slider returns current value.}
582 \texttt{/*}
583 \texttt{arg.value = (XtArgVal) *callData;}
584 \texttt{XtSetArg(arg,XtNbackground,arg.value);}
585 \texttt{XtSetValue(stub,&arg,1);}
586 \texttt{)}
1236 {
1237     Display *display = XtDisplay(toplevel);
1238     int screen = XDefaultScreen(display);
1239     int n, ncolors=2;
1240     Widget w;
1241     n = XDefaultDepth(display, screen);
1242     for(i=1; i<n; i++) {
1243         ncolors = ncolors*2;
1244     }
1245     ncolors = ncolors - 1;
1246     i = 0;
1247     XtSetArg(arg[i], XtNposition, (XtArgVal) OL_TOP); i++;
1248     XtSetArg(arg[i], XtNalignment, (XtArgVal) OL_CENTER); i++;
1249     XtSetArg(arg[i], XtNlabel, (XtArgVal) "Slider"); i++;
1250     slider_caption = XtCreateManagedWidget("slider_caption",
1251         captionWidgetClass, form, arg, i);
1252     i = 0;
1253     XtSetArg(arg[i], XtNWidth, (XtArgVal) N200_H_PIXELS); i++;
1254     XtSetArg(arg[i], XtNorientation, (XtArgVal) OL_HORIZONTAL); i++;
1255     XtSetArg(arg[i], XtNsliderMax, (XtArgVal) ncolors); i++;
1256     XtSetArg(arg[i], XtNgranularity, (XtArgVal) 1); i++;
1257     XtSetArg(arg[i], XtNticks, (XtArgVal) 1); i++;
1258     XtSetArg(arg[i], XtNtickUnit, (XtArgVal) OL_SLIDERVALUE); i++;
1259     XtSetArg(arg[i], XtNdragCBType, (XtArgVal) OL_RELEASE); i++;
1260     w = XtCreateManagedWidget("slider",
1261         sliderWidgetClass, slider_caption, arg, i);
1262     XtAddCallback(w, XtNSliderMoved, sliderCB, NULL);
1263 }
Defining and Using a Stub Widget

This section describes how to develop and use a stub widget.

The stub widget provides the application programmer with the flexibility of customizing a widget’s visual appearance and semantics. In this application, selecting the caption button “RAINBOW” paints the stub widget window by calling the function, `DrawAndPrint`. The stub widget itself refreshes the current contents of its window when necessary. Lines 1017-1034 illustrate the creation of the stub widget. Line 1027 registers the function `DrawAndPrint`, to repaint the widget when it is exposed. Lines 1033-1034 add an event handler to monitor the pointer entering and leaving the widget’s window. Lines 345-365 present the function, `StubEventHandler`. The stub widget also has its own cursor, which is set with the function, `SetStubCursor` in lines 331-341. Other related sections of code are `DrawAndPrint` (lines 143-223) and `GetColors` (lines 238-276). Please see Chapter 6.

```c
1017 /*
1018  * First two arguments scale widget to resolution of screen.
1019 */
1020   i = 0;
1021 XtSetArg(arg[i], XtNheight, (XtArgVal) N100_V_PIXELS); i++;
1022 XtSetArg(arg[i], XtNWidth, (XtArgVal) N100_H_PIXELS); i++;
1023 XtSetArg(arg[i], XtNbackground, skyblue_pixel); i++;
1024 /*
1025  * DrawAndPrint() will be called with an Expose event;
1026 */
1027   XtSetArg(arg[i], XtNexpose, DrawAndPrint); i++;
1028 stub = XtCreateManagedWidget("stub",stubWidgetClass,form,arg,i);
1029 /*
1030  * Add an event handler to track when the pointer enters and leaves the stub widget window.
1031 */
1032   XtAddEventHandler(stub,EnterWindowMask | LeaveWindowMask,
1033                       FALSE, StubEventHandler, (XtPointer)NULL);
```
345 static void
346 StubEventHandler(widget,clientData,event)
347 Widget widget;
348 XtPointer clientData;
349 XEvent *event;
350 {
351  XCrossingEvent *xce;
352  /*
353   * The xce pointer allows referencing to event specifics - see Xlib.h
354  */
355  if(event->type==EnterNotify || event->type==LeaveNotify)
356     xce = (XCrossingEvent *) &(event->xcrossing);
357  else
358     return;
359  if(event->type ==EnterNotify)
360     FooterMessage(footer_text, "Footerpanel: Pointer entered STUB widget");
361  else
362     FooterMessage(footer_text, "Footerpanel: Pointer left STUB widget");
363 }

...}
331 static void
332 SetStubCursor(widget)
333  Widget widget;
334 {
335   static Cursor cursor;
336  /*
337   * See OlCursor.c for other cursor possibilities.
338  */
339   cursor = GetOlQuestionCursor(XtScreen(widget));
340   XDefineCursor(XtDisplay(widget), XtWindow(widget), cursor);
341 }
Using a Form Widget

This last example shows how to create a form widget and how to position other widgets on the form. The form is a useful “backdrop” widget; it provides an empty area on which to position other widgets.

Lines 110 - 117 define some of the constraint resources used to position widgets on the form. These resources are specified for each widget and are later used to support positioning the widgets relative to each other. Both XtNxOffset and XtNyOffset are defined later in the application as a specific number of pixels. The XtNxOffset indicates that a child widget will be positioned a specific number of pixels to the right of a reference widget, defined by XtNxRefName. The XtNyOffset indicates that a child widget will be positioned a specific number of pixels below a reference widget, defined by XtNyRefName.

While XtNxRefWidget and XtNyRefWidget are also form constraint resources, XtNxRefName and XtNyRefName are used since they allow specifying the relative positions of widgets before any of the widgets are created.

The XtNxAddWidth and XtNyAddHeight resources determine whether the reference widget's width or height, that is, XtNxRefWidget's width or XtNyRefWidget's height, should be added to the offset.

...  
110 static Arg genericARGS[] = {

    111  { XtNxRefName, NULL },
    112  { XtNyRefName, NULL },
    113  { XtNxOffset,(XtArgVal) 0 },  /* to be initialized below */
    114  { XtNyOffset,(XtArgVal) 0 },  /* to be initialized below */
    115  { XtNxAddWidth,(XtArgVal) TRUE },
    116  { XtNyAddHeight,(XtArgVal) TRUE },
    117  },
...
The `SetPosition` routine sets the resource values in the current widget that will be used by the Form widget to place the current widget relative to other widgets already placed on the Form. The way to read the definition of `SetPosition` is "position widget with its upper left hand corner to the right of the upper left hand corner of xwidget and below the upper left hand corner of ywidget." How much to the right is determined by `XtNxOffset` (perhaps plus xwidget's width) and how much below is determined by `XtNyOffset` (perhaps plus ywidget's height). The figure below illustrates the effects of the `XtNxAddWidth` and `XtNyAddHeight` resources on placing new widget when both xwidget and ywidget are the same reference widget:

![Diagram showing the effects of `XtNxAddWidth` and `XtNyAddHeight` resources on placing new widget when both xwidget and ywidget are the same reference widget.]

- Both = True
  - `reference widget` to the right of the original form by `XOFFSET`.
- `XtNxAddWidth = FALSE`
  - `reference widget` and `NEW` widget on the right of the original form.
- `XtNyAddHeight = FALSE`
  - `reference widget` and `NEW` widget on the bottom of the original form.
- Both = FALSE
  - `NEW` widget on the bottom right of the original form.
Annotated Sample Programs

Lines 1470-1473 position the stub on left of the form below the caption.

```c
static void
SetPosition(widget, xwidget, ywidget)
  Widget widget;
  char *xwidget, *ywidget;
{
  static int nargs;

  if(nargs == 0)
    nargs = XtNumber(genericARGS);

  genericARGS[2].value = (XtArgVal) N10_H_PIXELS;
  genericARGS[3].value = (XtArgVal) N10_V_PIXELS;

  SetPosition(caption,"form","ca_caption");
  SetPosition(stub,"form","caption");
```
Using Flattened Widgets

This section presents pieces of code that will serve as examples of how to set­
up, create and use Flattened Widgets.

Specifying the Container Setting

The following code fragment illustrates how to create a flat exclusives widget setting.

In this example, each of the button sub-objects has the same client data ("test case") for the select callback procedure. Because of this, the XtNclientData resource can be specified for the container only. This allows each sub-object to inherit this value. If each sub-object wanted a different client data, the XtNclientData resource would be added to the other sub-object resources. This would automatically disable the inheriting of the container’s client data value.

To improve the readability of this example, required type casts of the fields in the FlatExclusives structure initialization have been deliberately omitted.

```
/* Application Defined Structure */

typedef struct {
    XtArgVallabel;       /* pointer to a string */
} FlatExclusives;

NOTE
Notice that all the fields in the application-defined structure, FlatExclusives, have the type XtArgVal. An alternate form for specifying the FlatExclusives type is:

    typedef XtArgVal FlatExclusives[#];

where '#' is the number fields per record.
Using Flattened Widgets

```c
String
exc_fields[] = { XtNlabel };

static void
cb()
{ /* This is the callback procedure...
    something interesting should go in here */ }

CreateObjects(parent)
    Widget parent;
{
    Arg      args[6];
    static FlatExclusives exc_items[] = { /* label for each button */
        { "Choice 1" },
        { "Choice 2" },
        { "Choice 3" }
    };

    XtSetArg(args[0], XtNitems, exc_items);
    XtSetArg(args[1], XtNnumItems, XtNumber(exc_items));
    XtSetArg(args[2], XtNitemFields, exc_fields);
    XtSetArg(args[3], XtNnumItemFields, XtNumber(exc_fields));
    XtSetArg(args[4], XtNselectProc, cb,);
    XtSetArg(args[5], XtNclientData, "test case");

    XtCreateManagedWidget("exclusives", flatExclusivesWidgetClass,
                          parent, args, 6);
} /* END OF CreateObjects() */
```

Callbacks and Flat Widgets

There are two differences in the way callbacks are handled for flat widgets as opposed to traditional widgets. The first difference is that sub-objects do not use XtCallback lists; instead, they use a single XtCallbackProc procedure.

Secondly, since the sub-objects of flattened widget containers are not true widget instances, the widget argument supplied to an application's callback procedure indicates the flat container widget that is ultimately responsible for managing the sub-object. For example, the flatExclusivesWidget id would be supplied as the widget id to the callback procedure for all sub-objects within the flat exclusives container. By maintaining this rule, the application always has the correct widget handy in the event that the application wishes to modify the item or its list from within the callback procedure.
The value of the XtNclientData resource is supplied as the client_data field to the callback procedure.

The call_data field is a pointer to a structure that the application can use to determine information about the sub-object associated with the current callback. The new structure is as follows:

```c
typedef struct {
    Cardinal   item_index; /* sub-object initiating callback */
    XtPointer  items;      /* Sub-object list head */
    Cardinal   num_items;  /* number of items */
    String *   item_fields; /* key of fields for list */
    Cardinal   num_item_fields; /* number of fields per item */
} OlFlatCallData;
```

where:

- `item_index` The index of the item (that is, the specific sub-object) responsible for initiating this invocation of the callback.
- `items` The head of item list that contains the sub-object initiating the callback.
- `num_items` The total number of items in the sub-object list.
- `item_fields` The list of resource names used to parse the records in the sub-objects list.
- `num_item_fields` The number of resource names contained in `item_fields`.

### Setting the State of a Sub-Object

The application can use two methods to change the state of an item: use the OlFlatSetValues procedure to modify one or more attributes of a sub-object, or directly modify the item list that the container and the application share.

The first approach is very similar to doing an XtSetValues request on a widget, except that the OlFlatSetValues routine requires the item index as well as the widget id, args and num_args. The routine is defined as:
The following code example uses this routine and illustrates how to change an item’s label from within a callback procedure. The example assumes the new label was specified as the client data.

```
Callback(widget, client_data, call_data)
    Widget widget; /* FlatExclusives Widget id */
    caddr_t client_data; /* the new static label */
    caddr_t call_data; /* OIFlatCallData structure pointer */

    OIFlatCallData * fcp = (OIFlatCallData *)call_data;
    Arg args[1];

    XtSetArg(args[0], XtNlabel, client_data);
    OIFlatSetValues(widget, fcp->item_index, args, 1);

} /* END OF Callback() */
```

Notice that the callback procedure did not have to know the number or the order of the item fields. The only requirement was that the XtNlabel resource is among the application-specified item fields, because if it were not, the above request would be ignored.

If the application does not use the above approach and modifies the item list directly, the application must ensure that all items within the list have valid states, since the container literally treats this type of modification as if the container were given a new list. For example, if an application wished to change a currently-set exclusive item, the application would have to unset the currently-set item and set the new item. If the application only set the new item, the container would generate a warning since the item list contains more than one set item.
The following example shows how a callback procedure changes the set item by modifying the item list. This example makes the first item be the set item whenever the last item is selected. Notice that once the list has been touched, the application must 'inform' the container of the modification. Also notice that in this example the callback needs to know the structure of the application to directly change its contents.

```c
/* Application Defined Structure from previous example */
typedef struct {
    XtArgVallabel; /* pointer to a string */
    XtArgValselect_proc; /* pointer to a callback procedure */
    XtArgValset; /* this item is currently set */
    XtArgValSensitive; /* this item is sensitive */
} FlatExclusives;

Callback(widget, client_data, call_data)
    XtPointer client_data; /* application’s client data */
    XtPointer call_data; /* OIFlatCallData structure pointer */
{
    OIFlatCallData * fcp = (OIFlatCallData *) call_data;

    if (fcp->num_items == (fcp->item_index + 1))
    {
        FlatExclusives * fexc_items = (FlatExclusives *) fcp->items;
        Arg args[1];
        /* Unset this item and
           * set the first one */
        fexc_items[fcp->item_index].set = FALSE;
        fexc_items[0].set = TRUE;
        /* Inform the container that the list
           * was modified */
        XtSetArg(args[0], XtNitemsTouched, TRUE);
        XtSetValues(widget, args, 1);
    }
} /* END OF Callback() */
```
Getting the State of a Sub-Object

Obtaining the state of a sub-object can also be achieved in two ways. The first is by using the `OlFlatGetValues` routine, specifying the index of the item to be queried.

```c
OlFlatGetValues(widget, item_index, args, num_args)
```

- `Widget  widget; /* flat widget id */`
- `Cardinal item_index; /* item to query */`
- `ArgList args; /* query resources */`
- `Cardinal num_args; /* number of query resources */`

If this approach is used, the application can query any sub-object resource even though it does not appear in the item fields. Take the initial example for instance, the application can query the `XtNfontColor` resource from any sub-object even though it does not appear in the FlatExclusives structure.

The second method for getting the state of a sub-object is by looking directly into the sub-object list since both the application and flat container share the same instance of the sub-object description.

Obtaining Help on a Sub-Object

The application can specify a unique help message for each sub-object in a similar fashion as help is registered for widgets, that is, through the `OlRegisterHelp` routine. Since sub-objects are not real widgets, but are extensions of the flat widget container, the help registration routine has a complex id:

```c
typedef struct {
    Widget  widget;  /* Flat Widget id */
    Cardinal item_index;  /* item to register help on */
} OlFlatHelpId;
```

The following example registers help on the eighth sub-object in the flat widget:
static String tag = "Item 8";
static String source = "Item 8’s help";
OlFlatHelpId help_id;

help_id.widget = flat_widget;
help_id.item_index = 7;

OlRegisterHelp(OL_FLAT_HELP, (XtPointer *)&help_id,
               tag, OL_STRING_SOURCE, source);
Programming Caveats

Naming Conventions

This may belabor the obvious - but you must pay attention to the naming conventions or you will quickly get lost inside your own code.

Notice that each of the examples related the callback name to the widget name in an organized way. You will find it pays dividends to establish those kinds of conventions for yourself. For example, if you define a Quit OblongButton widget, you may find it useful to define the callback routine "QuitCallback" or "QuitCB".

Macro Alerts

Do not use auto-increment or auto-decrement in any intrinsic or OPEN LOOK function calls.

For example, use

```c
XtSetArg(arg[i],...); i++;
```

rather than

```c
XtSetArg(arg[i++],...);
```

Callback Restrictions

Be careful not to use a "long jump" (setjmp(3C)) within a callback function. A long jump is used to bypass the normal function return structure (usually when an error condition occurs). This will not work from callbacks. The widget requires a return from the callback to complete updates to internal state tables, and using a "long jump" skips this return, causing the widget to have incorrect state information. This, in turn, causes incorrect visuals or program failures.
Global Name Space Restrictions

The OPEN LOOK widget library uses a particular convention when naming external C procedures, variables, and structure. It will always begin with “01” and may be optionally preceded by an underscore “_.” Similarly, preprocessor symbols (#define’s) will begin with “01” or “OL” and may be optionally preceded by an underscore. The global name space is therefore restricted in this way and you should not choose external names which begin with these prefixes.

Debugging Hints

Use the OPEN LOOK® Graphical User Interface Style Guide. The cleaner and more user-oriented the screen layout is, the easier it will be to debug.

You can consider developing applications in one of these ways:

- **One Object at a Time**
  
  First place the object, then check out the callback routine. The advantage of this is that it is very simple; one step at a time. The difficulty is that often one callback is really a function of other on-screen activities.

- **Place All Objects**
  
  First, place all objects on the screen, then debug the callbacks. The advantage of this approach is that placement and callback functions are easily separable functions and that placement is often a developmental checkpoint. Also, the end-user interface is both logically and functionally distinct from the actual application.

- **One Grouping at a Time**
  
  Develop a composite and all its constituents, including placement and callbacks. The advantage of this is that it often corresponds to a functional module of the application and, therefore, resembles the “traditional” style of modular development.

In any case, creating a modular development style is probably the single most valuable aid to program debugging.
4 X Window System, Version 11, Conventions for OPEN LOOK

Introduction 4-1

General Considerations 4-2
Methods of Communication 4-2
- Properties 4-2
- Events 4-2
Restrictions 4-3
Nomenclature 4-3
Protocol Notes 4-4
- Extensibility 4-4
- Efficiency 4-4
Property Notation 4-4

Window Properties 4-6
Window Decorations 4-6
Standard Decorations 4-6
Customizing Decorations 4-8
The Pin State 4-10
Window Colors 4-11
Busy Windows 4-12
Focus Warping 4-13

Relationship to Inter-Client Conventions 4-14
WM_NORMAL_HINTS 4-14
WM_HINTS 4-15
WM_PROTOCOLS 4-15
- WM_SAVE_YOURSELF 4-15

Table of Contents i
Table of Contents

- WM_DELETE_WINDOW 4-15
- WM_TAKE_FOCUS 4-16
  Window Groups 4-16
  Input Focus 4-19

Workspace and File Manager Conventions 4-21

Miscellaneous Implementation Issues 4-27
  Pinnable Menus and Override-Redirect 4-27
  Full Size Window 4-27
Introduction

The OPEN LOOK graphical user interface specifies the behavior and appearance of the entire system. The X Window System, version 11 (X11) is composed of a server and several clients. Some clients are dedicated to performing particular tasks, such as window management, while others are application programs. Therefore, the responsibility for implementing OPEN LOOK under X11 must be shared among several different clients. This chapter specifies how X11 clients must cooperate in order to implement OPEN LOOK.

The X11 protocol was designed using the principle of mechanism, not policy. The protocol provides only the tools with which to build an environment, but it doesn’t determine how these tools are to be used. If there were no policy governing the use of the X11 protocol, applications that worked correctly in isolation may fail to work when they share an environment with other applications. Therefore, conventions are necessary so that applications can coexist and interoperate. The standard conventions are described in the X11 Inter-Client Communication Conventions Manual (ICCCM). All X11 clients are required to conform to these conventions in order to guarantee interoperability.

An OPEN LOOK environment requires the existence of an additional set of conventions beyond those described in the ICCCM. The purpose of this chapter, then, is to detail the conventions necessary to guarantee that OPEN LOOK applications written by different vendors, using different toolkits and languages, will interoperate. This is a private set of conventions that must be supported only by OPEN LOOK applications. OPEN LOOK applications must support the conventions of the ICCCM as well as those outlined in this chapter.
General Considerations

Methods of Communication

The X11 protocol provides two principal means by which clients can communicate with each other: properties and events. Both mechanisms are described in the X11 protocol as having uninterpreted data, that is, data that is transmitted along with but not interpreted by the protocol. Clients use the uninterpreted data fields in properties and events to communicate amongst themselves.

Properties

Properties are uninterpreted data that are named, typed, and associated with a window. Properties are thus useful for storing pieces of a window’s state. For communication between an application and a window manager, the properties will be placed on the application’s top-level window(s). Unless otherwise specified, all properties will have format 32. This is necessary in order to avoid byte-order and structure-packing problems.

In X11, any client can write to any property. However, multiple clients writing to the same property raises the possibility of race conditions. Therefore, this chapter will usually designate a client that is the owner of each property. Only the owner of a property will be allowed to write to it.

Events

Events are typically generated by the server to notify a client that user input has occurred. However, there is a facility whereby clients may generate events and cause them to be sent to other clients. Events generated in this manner are called synthetic events because they were synthesized by a client, not necessarily in response to any real user action.

A special type of event called a Client Message is never created by the server; it can be synthesized only by a client. Client Messages have enough room for a small amount of uninterpreted data and are thus useful for sending datagram-like messages between clients. Client Messages are typically used for notification that an event has occurred; they are not used to transmit state information.
General Considerations

Restrictions

The rules for using properties and Client Messages as described in this chapter define a mini-protocol that exists entirely within the X11 protocol. When trying to communicate with other clients, any client can assume that all other clients understand the X11 protocol. However, OPEN LOOK clients cannot assume that other clients will understand the OPEN LOOK mini-protocol. OPEN LOOK clients must be prepared to deal with this situation. This principle has been embedded into the design of the mini-protocol by allowing the mini-protocol to fail gracefully if one of the communicating parties doesn’t understand it.

This case may arise if an OPEN LOOK window manager is managing a non-OPEN LOOK application, or if an OPEN LOOK application is being managed by a non-OPEN LOOK window manager. OPEN LOOK applications must be able to operate (perhaps with reduced, but acceptable functionality) without the presence of an OPEN LOOK window manager. By the same token, an OPEN LOOK window manager must not depend on all of its applications implementing this protocol. This restriction precludes a style of transaction where, for example, an application sends a message to the window manager and waits for a reply. If the window manager doesn’t implement the OPEN LOOK Interface, the application will never receive this reply. The transactions under this protocol must be completely asynchronous.

Nomenclature

All properties and Client Message types are named by X11 atoms. The X11 Protocol document (X Windows System Protocol, Release 4, Predefined Atoms) states that atom names private to a particular vendor or organization should have unique prefixes that begin with an underscore (“_”). Using the prefix __OPEN_LOOK__ is the obvious choice, but it makes all of the atoms too long. Therefore, all atoms unique to the OPEN LOOK interface are prefixed with _OL_.

Several terms in this chapter are used in specific ways that don’t necessarily correspond to usage elsewhere. A manager is a dedicated client that manages a shared resource. Typically there should be exactly one manager of each type. In this chapter, the term client typically means any ordinary client that is not a manager. Each client is further divided into two parts, the toolkit and the application.
The terms *window manager* and *session manager* are used as defined in the ICCCM. (Note that the window and session managers might be two separate clients or merged into a single client.) A *workspace manager* is OPEN LOOK usage, and is equivalent to a session manager. The *file manager* is an OPEN LOOK-specific application that provides a graphic view of the file system.

**Protocol Notes**

**Extensibility**

This protocol is designed for extensibility. Atoms are used in fields wherever possible so that the range of values is not limited. Lists of atoms are preferred to bitmasks for specifying options. The _OL_PROTOCOLS property is a list of atoms that indicates which sub-protocols the client supports. This list is extensible.

**Efficiency**

Each X11 protocol request can update only one property at a time. Furthermore, each time a property is changed, the server generates a PropertyNotify event. As more properties are added, correspondingly more requests are required to update them and more events are generated. Therefore, adding a new property to the conventions is a very expensive step. If additional data is necessary, it should be added to existing properties (in an upwardly-compatible way) in preference to adding new properties.

**Property Notation**

The *Name*, *Type*, and *Format* headings are self-explanatory. The *Owner* heading indicates which party is responsible for maintaining the contents of the property. The *Reader* heading indicates who is responsible for reading and acting on the property. The *Effect* heading indicates when changes to this property take effect. Typical values are *immediate*, which means that the reader should track changes at all times; and *exit Withdrawn*, which means that the window manager reads the property only when the window leaves the Withdrawn state.
The names of fields within the property are used only for reference purposes within this chapter. They have no relevance to the X11 protocol. The Default for a particular field indicates what the reader should assume if the field isn’t present or if the entire property is absent.
Window Properties

Window Decorations

These properties are used to communicate between clients and the window manager about how the clients' top-level windows should be decorated. In addition, the window manager sets some properties to inform the client of certain pieces of state, such as the pushpin, that are under the user's control. OPEN LOOK currently specifies that there be no window background next to a scroll bar, if the application has one. There currently is no method for a client to tell a window manager where its scroll bar is.

See the discussion on the OPEN LOOK VendorShell in Appendix A: Introduction to General Resources for more convenient ways of accessing window properties.

Standard Decorations

The client sets _OL_WIN_ATTR property on each top-level window to tell the window manager the window's type, along with other decoration options.

<table>
<thead>
<tr>
<th>_OL_WIN_ATTR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong>  _OL_WIN_ATTR</td>
</tr>
<tr>
<td><strong>Format:</strong>  32</td>
</tr>
<tr>
<td><strong>Length:</strong>  5</td>
</tr>
</tbody>
</table>
### Window Properties

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| flags       | CARD32 | 1 = win_type  
2 = menu_type  
4 = pin_state  
8 = cancel   |         | this is a bitmask that indicates which fields are present. |
| win_type    | XA_ATOM | _OL_WT_BASE      |         | base window                                           |
|             |       | _OL_WT_CMD       |         | command window                                        |
|             |       | _OL_WT_HELP      |         | help window                                           |
|             |       | _OL_WT_NOTICE    |         | notice window\[1\]                                    |
|             |       | _OL_WT_OTHER     |         | client-specified\[2\]                                 |
| menu_type   | XA_ATOM | _OL_MENU_FULL    | [3]     | full menu                                             |
|             |       | _OL_MENU_LIMITED  |         | limited menu                                          |
|             |       | _OL_NONE         |         | no menu                                               |
| pin_state   | CARD32 | 0                | 0       | pin is out                                            |
|             |       | 1                |         | pin is in                                             |
| cancel      | CARD32 | 0                | 0       | dismiss                                               |
|             |       | 1                |         | cancel                                                |

[1] This protocol allows the window manager to support notice frames as ordinary windows. Toolkits aren’t required to use window manager windows for their notices. For example, a toolkit might choose to implement notices by grabbing the server and mapping an override-redirect window. The window manager doesn’t have to know about these at all.

[2] If the *win_type* field contains _OL_WT_OTHER, the window manager will provide no decorations by default. The application can add decorations as it wishes by specifying them in the _OL_DECOR_ADD property (see below).

[3] The default menu type is implied by the window type. For base windows, the default menu is a full menu. For pop-ups, the default is a limited menu. For _OL_WT_OTHER, the default is no menu at all.

The following table indicates the default decorations that occur on a window depending on its type.
Window Properties

<table>
<thead>
<tr>
<th>window type</th>
<th>header</th>
<th>close</th>
<th>pin</th>
<th>resize</th>
<th>menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>_OL_WT_BASE</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>Full</td>
</tr>
<tr>
<td>_OL_WT_CMD</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>Limited</td>
</tr>
<tr>
<td>_OL_WT_NOTICE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>_OL_WT_HELP</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>Limited</td>
</tr>
<tr>
<td>_OL_WT_OTHER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>

An "X" indicates the presence of the decoration; no "X" indicates its absence.

A full menu contains the following entries: Close, Full Size, Properties, Back, Refresh, Move, Resize, and Quit. A limited menu contains the following entries: Dismiss/Cancel, Back, Refresh, Move, Resize, and Owner?. The Dismiss menu item changes to Cancel if the client has requested it in the cancel field.

Clients should take care not to make gross changes to decorations while the window is mapped, such as changing a base window into a notice, because this would result in flickering that users would likely find objectionable. If a client really needs to reuse a window for a different purpose, it should unmapping the window, make the changes, and then remap the window.

Customizing Decorations

There are certain cases where the client requires different decorations from those provided by default. To add or delete decorations from the default set provided for a window, the client can create one or both of the _OL_DECOR_ADD and _OL_DECOR_DEL properties. The type of each property should be ATOM. Each property is a variable-length list of atoms that indicates which decorations should be added to (in the case of _OL_DECOR_ADD) or deleted from (_OL_DECOR_DEL) the default set of decorations on this window.

For example, resize corners are present by default on most windows. An application could request that a window not be resizable by putting the _OL_DECOR_RESIZE atom in the _OL_DECOR_DEL property on that window.
You must use the Xlib function, `xChangeProperty`, in order to place a property on a window and to change the value of a property on a window.

<table>
<thead>
<tr>
<th>Type:</th>
<th>ATOM</th>
<th>Owner:</th>
<th>client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format:</td>
<td>32</td>
<td>Reader:</td>
<td>window manager</td>
</tr>
<tr>
<td>Length:</td>
<td>variable</td>
<td>Effect:</td>
<td>immediate</td>
</tr>
<tr>
<td>Atom Description</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| _OL_DECOR_CLOSE   | close box       |
| _OL_DECOR_RESIZE  | resize corners  |
| _OL_DECOR_HEADER  | window header   |
| _OL_DECOR_PIN     | pushpin         |

The preferred implementation of footers is for the client to manage the footer itself. A typical implementation of a window footer would make it be a small pane at the very bottom of a stack of panes inside the client's top-level window. The advantage of this method is that it allows the client maximum flexibility in managing the footer. For example, the client could make the footer scrollable or be several lines high.

Clients that implement footers should take care not to select input on any of the mouse buttons in this window, because this area is logically part of the window background. Mouse events should be allowed to fall through to be handled by the window manager.

In certain cases, the application may wish to specify exactly what decorations occur on a window. To do this, it would specify a window type of _OL_WT_OTHER in the `win_type` field of the _OL_WIN_ATTR property, and put the appropriate decoration atoms in the _OL_DECOR_ADD property.

If the application puts the same atom into both the _OL_DECOR_ADD and _OL_DECOR_DEL properties, the behavior is undefined. The behavior is also undefined if the application requests a combination of decorations that doesn’t make sense, such as a closed box and a pushpin.
The Pin State

If the current window has a pushpin, the window manager will create and maintain the property called _OL_PIN_STATE. The window manager is responsible for updating this property to reflect the state of the pin whenever the user changes it. The initial contents of this property are taken from _OL_WIN_ATTR.pin_state, if it exists.

<table>
<thead>
<tr>
<th>_OL_PIN_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong></td>
</tr>
<tr>
<td><strong>Owner:</strong></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
</tr>
<tr>
<td><strong>Reader:</strong></td>
</tr>
<tr>
<td><strong>Length:</strong></td>
</tr>
<tr>
<td><strong>Effect:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>pin is out</td>
</tr>
<tr>
<td>1</td>
<td>pin is in</td>
</tr>
</tbody>
</table>

The client should inspect the state of the pin whenever the user clicks in a button on a pop-up window. If the window is not pinned, the client should withdraw the window after completing the function successfully. The client doesn't have to withdraw the window if the operation wasn't successful. The client should not inspect the state of the pin upon receipt of WM_DELETE_WINDOW.

The client shouldn't set this property to try to move the pin itself. It should instead change the _OL_WIN_ATTR.pin_state field. The window manager should respond by changing _OL_PIN_STATE.

If the client has requested a pin, but the _OL_PIN_STATE property does not exist, the client can use whatever state it last requested for the pin. For example, suppose the client requested the pin's initial state be in by setting _OL_WIN_ATTR.pin_state appropriately. If a non-OPEN LOOK window manager is running, it will not set the _OL_PIN_STATE property. If this is the case, the client can make the decision to withdraw the window based on the pin state that it last requested.
Window Properties

In the case where the client requested to have the window pinned, there would be no way to unpin the window. It might be argued that this is a serious problem, because the window could never be removed. This is not the case. The foreign window manager would presumably have its own user interface for closing (iconifying) or deleting (with `WM_DELETE_WINDOW`) the window. These operations are independent of the state of the pin, so they will work as expected.

Window Colors

The client can specify different colors for the window background, foreground, and border. The foreground consists of the header text and window mark. The border consists of the inside border of the decorator (the one that thickens for selection highlighting) and the resize corners. The client specifies these colors as RGB triples in the `_OL_WIN_COLORS` property.

Border is ignored for this property only.

```
| Type: _OL_WIN_COLORS | Owner: client |
| Format: 32           | Reader: window manager |
| Length: 10           | Effect: immediate |
```
Window Properties

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>CARD32</td>
<td>1 = foreground&lt;br&gt;2 = background&lt;br&gt;4 = border</td>
<td>This is a bitmask that indicates which fields are present.</td>
</tr>
<tr>
<td>fg_red</td>
<td>CARD32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fg_green</td>
<td>CARD32</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>fg_blue</td>
<td>CARD32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bg_red</td>
<td>CARD32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bg_green</td>
<td>CARD32</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>bg_blue</td>
<td>CARD32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bd_red</td>
<td>CARD32</td>
<td></td>
<td>RGB values for border (ignored by this property)</td>
</tr>
<tr>
<td>bd_green</td>
<td>CARD32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bd_blue</td>
<td>CARD32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Busy Windows

When it wants to put a window in the busy state, the client should set the _OL_WIN_BUSY property on that window. If this property is not present, the window manager will assume that the window is not busy.

<table>
<thead>
<tr>
<th>_OL_WIN_BUSY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: INTEGER</td>
</tr>
<tr>
<td>Format: 32</td>
</tr>
<tr>
<td>Length: 1</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
When a window becomes busy, the window manager should gray out the window header. The window manager does not trap all keyboard and mouse input to the window. This is the responsibility of the application that owns that window. The window should beep in response to any input. (It can do this with one of the flavors of a grab or by mapping an Input Only window.) When a client is about to become unbusy, it should synchronize with the server and flush all input immediately prior to setting this property. This will help ensure that OPEN LOOK clients running in a non-OPEN LOOK environment will ignore input while they are busy.

**Focus Warping**

When an application brings up a pop-up window that is eligible for keyboard input, such as command window or property sheet, the OPEN LOOK Interface requires that the input focus be transferred there. This is the responsibility of the window manager.

The window manager should first determine whether the window is eligible to receive the input focus by checking which focus model the client has chosen for this window. Clearly, windows using the No Input model should not have the focus transferred to them. For the other models, the window manager should then query and save away the window that currently has the input focus. The window manager should then manipulate the focus as if the user had clicked SELECT on the window background. This action will be some combination of sending a `WM_TAKE_FOCUS` message or setting the focus to that window, depending on the focus model.

When the pop-up goes away, the window manager should attempt to restore the input focus to its previous location, again using the same focus action it would take as if the user had clicked SELECT on the window background.
Relationship to Inter-Client Conventions

All X11 clients should conform to the X11 inter-client conventions. These conventions are quite flexible in how window managers are required to deal with requests from clients. In particular, most of the window manager properties are called hints because the window manager is free to ignore them as it sees fit. For most of OPEN LOOK, however, it makes sense for window managers to honor application hints.

Refer to the X11 Inter-Client Communication Conventions Manual for the detailed formats of WM_NORMAL_HINTS, WM_HINTS, and WM_PROTOCOLS described below.

WM_NORMAL_HINTS

The window manager should honor the size of the window as created by the application.

If the USPosition flag is set, use the initial position of the window. If the USPosition flag is not set, the window manager should disregard the initial position of the window, instead positioning the window according to the standard window layout specified by OPEN LOOK.

The window manager should use the OPEN LOOK standard window layout policy even if the PPosition flag is set. The rationale for doing this is that quite a few programs always set the PPosition flag.

The window manager should attempt to honor the minimum, maximum, base, and incremental sizes. The maximum size, if present, should be used when the user zooms the window with the Full Size menu item. The base and incremental sizes will be used when the window manager needs to calculate the size of the window in rows and columns, such as on the window property sheet.
**WM_HINTS**

Window managers should use the icon window, if one is provided by the client. If none is provided, then the manager should use the icon pixmap and mask, if they are provided.

The window manager should honor the icon position, if it is provided. If it isn’t, the window manager should position the icon using OPEN LOOK’s default icon positioning strategy. Clients should generally not set the icon position fields except under certain circumstances, such as when the user gives command-line options for positioning.

**WM_PROTOCOLS**

**WM_SAVE_YOURSELF**

OPEN LOOK clients should elect for the `WM_SAVE_YOURSELF` protocol. In response to this message, they should behave exactly as the ICCCM specifies; that is, they should write a `WM_COMMAND` property and go into a quiescent state. Clients should not attempt user interaction in response to a `WM_SAVE_YOURSELF` message. Clients should also not exit of their own accord after receipt of the `WM_SAVE_YOURSELF` message.

**WM_DELETE_WINDOW**

Assuming the client has elected to receive `WM_DELETE_WINDOW` messages, the window manager should send this message when one of the following situations occurs:

- The user selects the Quit item on any base window menu.
- The user selects the Dismiss/Cancel item on a pop-up window menu.
- The user removes the pin from a pop-up window.

For clients that do not request `WM_DELETE_WINDOW`, the window manager should issue a Kill Client when the user selects Quit from the window menu. If a client requests a pop-up window but not `WM_DELETE_WINDOW`, the window manager should simply unmap the window when the user pulls the pin or selects the Dismiss menu item.
Relationship to Inter-Client Conventions

OPEN LOOK clients should elect to participate in the WM_DELETE_WINDOW protocol if they need to intercept or ask for user confirmation when the user requests to dismiss a window or quit an application. If a client receives WM_DELETE_WINDOW on a pop-up window, the client should withdraw the window. If a client receives WM_DELETE_WINDOW on a base window, the client should withdraw the base window and its pop-ups (perhaps after requesting confirmation). It is up to the application whether to exit entirely or just withdraw the base window family in this situation.

WM_TAKE_FOCUS

OPEN LOOK clients that require keyboard input should participate in the WM_TAKE_FOCUS protocol. This is explained further on in the section on input focus.

Window Groups

OPEN LOOK applications should always fill in the window_group field of the WM_HINTS property on each window that they expect to be managed by an OPEN LOOK window manager. Base windows should be designated as group leaders; that is, they should have their own window ID in the window_group field. Pop-ups (command windows, property sheets, help windows) should belong to the window group of the base window with which they are associated.

Notice Widgets, PopupWindows and Menus set their own window group so, in these cases, the application does not have to set the window group.

Windows (both group leaders and followers) can be in one of three states, according to the Inter-Client Conventions: Withdrawn, Iconic, and Normal. Two sources can instigate transitions between these states: the user (via the window manager) and the client. The OPEN LOOK Interface specifies that pop-up windows follow the state transitions of their associated base window. That is, if you close and reopen a base window, its pop-up windows disappear and reappear along with the base window. The OPEN LOOK interface further specifies that a pop-up window cannot appear on the screen unless its base window is also visible.
The following table lists the permitted combinations of base window (group leader) and pop-up window (follower) states:

<table>
<thead>
<tr>
<th>leader is in</th>
<th>followers can be in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawn</td>
<td>only Withdrawn</td>
</tr>
<tr>
<td>Iconic</td>
<td>Iconic or Withdrawn</td>
</tr>
<tr>
<td>Normal</td>
<td>Normal or Withdrawn</td>
</tr>
</tbody>
</table>

The basic idea is that windows stay in Withdrawn until the client moves them to another state. The Withdrawn state means that the client never wants the window to be displayed unless it is explicitly moved out of the Withdrawn state. While a group leader is not in Withdrawn, all followers that aren’t in Withdrawn track the state transitions of the group leader.

When a group leader is not in the Withdrawn state, its followers must be in the same state or in Withdrawn. When the leader is in Normal, the followers can either be in Normal (present on the screen) or in Withdrawn (not present). When a group leader is in Iconic, all of its followers must either be in Iconic or Withdrawn. Only one icon is displayed for the entire group: the group leader’s. All followers are invisible to the user, regardless of whether they are in Iconic or Withdrawn. The significance of a follower being in Iconic (as opposed to Withdrawn) is that it will reappear on the screen — in the Normal state — when its leader is moved to the Normal state, whereas the Withdrawn followers remain Withdrawn.

In order to simplify things, it is convenient to disallow certain state transitions on group follower windows (pop-ups) when the base window is in certain states. The following table shows state transitions that the window manager should perform on the group follower windows when the group leader window undergoes a transition.
### Relationship to Inter-Client Conventions

<table>
<thead>
<tr>
<th>Leader changed from</th>
<th>to</th>
<th>Followers that were</th>
<th>change to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawn</td>
<td>Normal</td>
<td>[1] Withdrawn</td>
<td></td>
</tr>
<tr>
<td>Withdrawn</td>
<td>Iconic</td>
<td>[1] Withdrawn</td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>Withdrawn</td>
<td>(any) Withdrawn</td>
<td></td>
</tr>
<tr>
<td>Iconic</td>
<td>Withdrawn</td>
<td>(any) Withdrawn</td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>Iconic</td>
<td>Normal</td>
<td>Iconic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Withdrawn</td>
<td>Withdrawn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iconic</td>
<td>[2]</td>
</tr>
<tr>
<td>Iconic</td>
<td>Normal</td>
<td>Iconic</td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Withdrawn</td>
<td>Withdrawn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal</td>
<td>[3]</td>
</tr>
</tbody>
</table>

1. If the leader was in Withdrawn, all followers must have been in Withdrawn. Moving the leader out of Withdrawn leaves the followers as they were. The window manager will never implicitly move any window from the Withdrawn state; the client must do so explicitly.

2. If the leader was in Normal, no follower could have been in Iconic.

3. If the leader was in Iconic, no follower could have been in Normal.

If a client attempts a state transition on a follower that would result in an illegal combination, the window manager should ignore the request. For example, suppose a client attempts to change a follower from Normal to Iconic without changing the leader. This request should be ignored. The group leader must be in Normal because the follower was in Normal. Changing the follower to Iconic isn’t allowed unless the leader changes to Iconic simultaneously. To do that, the client should request the transition on the leader, not the follower.

These state transitions can be initiated either by the window manager or the client. However, the burden of maintaining the states consistently lies on the window manager. For example, if the client iconifies its base window, the window manager is responsible for moving all Normal pop-up windows into Iconic. Similarly, if the user iconifies a base window, the window manager is again responsible for iconifying the pop-ups.
Input Focus

The OPEN LOOK Interface specifies that focus is transferred with mouse clicks. Since arbitrary windows (not just top-level windows) may have the input focus, it is impossible for the window manager to do all the focus management itself. Therefore, OPEN LOOK clients should use the Globally Active model of input focus described in the Inter-Client Conventions document. This model corresponds to \texttt{WM_HINTS.input} having the value False and the presence of the \texttt{WM_TAKE_FOCUS} atom in the \texttt{WM_PROTOCOLS} property.

Window managers in general, not just OPEN LOOK window managers, will send a \texttt{WM_TAKE_FOCUS} message to the client when they think the client should take the input focus. OPEN LOOK clients should respond by setting the input focus to the last subwindow that had the focus. If no subwindow ever had the focus, the client should set the focus to the default focus location. In addition, when the user clicks the SELECT mouse button in a client's subwindow, the client should set the input focus to that window.

\begin{verse}
\textbf{NOTE} The timestamp of the event that caused the focus change should be passed to the client in the \texttt{WM_PROTOCOLS} client message.
\end{verse}

OPEN LOOK applications will generally not want to use the Locally Active model of focus, because this leads to unnecessary transfers of focus. In the Locally Active model, the window manager assigns the focus to the window, after which the application is free to move the focus around within its subwindows. Under this model, clicking in a scroll bar might transfer the focus. This is incorrect.

In rare cases, an OPEN LOOK application might want to use the Passive model. In this model, only the top-level window is allowed to have the focus. The window manager will assign the focus to this window as appropriate. If an OPEN LOOK application has only one top-level window, the application can use this model. However, most OPEN LOOK applications will want to have more than one window eligible to receive the focus, or their focus window will not be the top-level window, so the Passive model will be unsuitable for them.
If an OPEN LOOK application doesn't handle keyboard input, it should choose the No Input model of input focus. This corresponds to \texttt{WM\_HINTS.input} having the value False and the absence of the \texttt{WM\_TAKE\_FOCUS} atom in \texttt{WM\_PROTOCOLS}.

If the window that has the focus disappears (is dismissed or iconified) and the window manager does not restore the focus to its previous location (either because the window manager wasn't able to or wasn't supposed to), the input focus will likely end up as None. In this case, window manager should set the input focus to somewhere known (perhaps a root window) and beep whenever a keystroke occurs. (This is required by the OPEN LOOK specification.) The window manager can detect when this situation occurs by keeping track of which window has the input focus (by selecting for Focus In and Focus Out messages).
Workspace and File Manager Conventions

The following OPEN LOOK Communications Conventions have been estab­lished for the OPEN LOOK Workspace Manager and OPEN LOOK File Manager when communicating with the OPEN LOOK Window Manager and other OPEN LOOK clients.

To initiate the execution of a process, or other action by the Workspace Manager, an appropriately formatted request is appended to the _OL_WSM_QUEUE property of the Root Window. If the request is for process execution, then the success or failure of the request is reported by the setting of the _OL_WSM_REPLY property on the window specified in the request. Similarly, the _OL_FM_QUEUE and _OL_FM_REPLY properties are used to communicate file service requests and replies. The QUEUE properties may have multiple requests appended; however, the REPLY properties will contain only a single reply at any given time.

The enqueue routine should add requests to the queue using PropModeAppend. This way, new requests are added to the end of the queue without disturbing any prior contents.

On the first request, the dequeue routine should allot buffer space for the queue, move the entire contents of the queue into the buffer, specify True for the delete argument to XGetWindowProperty, and return the first request in the queue. For subsequent requests, the dequeue routine should parcel out requests from its buffer until the buffer is empty. At that time, the dequeue routine would again read the entire contents of the queue and continue request processing.

<table>
<thead>
<tr>
<th>Property</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_OL_WSM_REPLY</td>
<td>&lt;type&gt;&lt;serial&gt;:&lt;sysname&gt;:&lt;nodename&gt;:&lt;errno&gt; &lt;pid&gt;:</td>
</tr>
<tr>
<td>_OL_FM_REPLY</td>
<td>&lt;type&gt;&lt;serial&gt;:&lt;sysname&gt;:&lt;nodename&gt;:&lt;path&gt; &lt;message&gt;:</td>
</tr>
</tbody>
</table>
The _OL_WSM_QUEUE and _OL_FM_QUEUE formats are each a single long sequence. In the table above, they were split across two lines due to their length.

The format of each of the requests and replies is an ASCII character sequence containing a fixed number of fields. The first field, `<type>`, is a fixed length one character field. The remaining fields are variable length character sequences separated by the ASCII unit separator (0x1f), which is represented in the table above by a colon. If the field is numeric, then the characters in the field are all ASCII digits.

The next several paragraphs define valid values for the `type` field for both Workspace Manager and File Manager requests. The remainder of this section then describes the other fields that are present in the various request and reply formats.

**Field: `<type>`**

The `<type>` field contains a binary numeric value identifying the type of the message. Valid values are given in the following two tables.

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_OL_WSM_QUEUE</td>
<td>1</td>
<td>WSM_EXECUTE</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>WSM_TERMINATE</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>WSM_SAVE_YOURSELF</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>WSM_EXIT</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>WSM_MERGE_RESOURCES</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>WSM_DELETE_RESOURCES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_OL_WSM_REPLY</td>
<td>1</td>
<td>WSM_SUCCESS</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>WSM_FORK_FAILURE</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>WSM_EXEC_FAILURE</td>
</tr>
</tbody>
</table>
The properties are specified in the *command* field of `_OL_WSM_QUEUE` as described below:

- **WSM_EXECUTE** is used by a client to request that a program be executed. These are the only requests that receive a reply.

- **WSM_TERMINATE** and **WSM_SAVE_YOURSELF** are used by the Window Manager to request that a window and its associated program(s) be terminated; the second form is used if the SaveYourselfMessage bit is set.

- **WSM_EXIT** is used by the Window Manager to denote the end of a sequence of **WSM_TERMINATES** and **WSM_SAVE_YOURSELFS** at session termination.

- **WSM_MERGE_RESOURCES** and **WSM_DELETE_RESOURCES** are used by a client to request that properties be merged into, or deleted from, the `.Xdefaults` file.

- **WSM_SUCCESS** indicates that the request was carried out successfully, while **WSM_FORK_FAILURE** and **WSM_EXEC_FAILURE** indicate failures that occurred in attempting to fork or execute the specified process.

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_OL_FM_QUEUE</td>
<td>1</td>
<td>FM_ACTIVATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FM_BROWSE</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>FM_COPY</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>FM_MOVE</td>
</tr>
<tr>
<td>_OL_FM_REPLY</td>
<td>9</td>
<td>FM_CANCEL</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>FM_ACCEPT</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>FM_INVALID</td>
</tr>
</tbody>
</table>

- **FM_ACTIVATE** is used by the Workspace Manager to request a stand-alone File Manager window, while **FM_BROWSE** is used by other clients to request a File Manager window in client service mode. All requests receive a reply.

- **FM_COPY** and **FM_MOVE** are used in drop mode to indicate that the user requested a copy or a move operation.

- **FM_CANCEL** is used in client service mode to indicate that a user has selected the cancel button.
Workspace and File Manager Conventions

- **FM_ACCEPT** is used in client service mode to indicate that the request was accepted and acted upon.
- **FM_INVALID** is used to indicate that the corresponding request was invalid.

**Field: <window>**

The **<window>** field contains the window id of the window whose _OL_WSM_REPLY_ or _OL_FM_REPLY_ property should be set in response to the current request.

**Field: <serial>**

The **<serial>** field contains a unique serial number that is supplied by the requesting routine for each request it appends. The same serial number is returned in the subsequent reply so that the reply may be associated with its corresponding request.

**Fields: <sysname> and <nodename>**

The **<sysname>** and **<nodename>** fields contain the system and node (as given by the `uname(1)` command) of the machine originating the request or reply.

**Fields: <uid> and <gid>**

The **<uid>** and **<gid>** fields are reserved for future use.

**Field: <applname>**

The **<applname>** field contains the name of the application making the request.

**Field: <command>**

For **WSM_EXECUTE**, the **<command>** field specifies a command to be executed by `/bin/sh`. 
For **WSM_MERGE_RESOURCES** and **WSM_DELETE_RESOURCES**, the
*<command>* field specifies the resource settings as

\[
\text{name}:\text{value} \quad \text{. . . name}:\text{value}
\]

where *name* is the name of the resource, : is the ASCII colon character, *value* is
the value assigned to the resource, \(\backslash n\) is the ASCII new-line character, and . . .
indicates that there may be one or more occurrences of name-value pairs in this
field.

For **WSM_DELETE_RESOURCES**, *value* can be null (that is, 0 characters). For types
other than **WSM_MERGE_RESOURCES** and **WSM_DELETE_RESOURCES**, this field is
ignored.

**Field**: **<atom>**

The **<atoms>** field is reserved for future use and is currently ignored.

**Field**: **<errno| pid>**

The **<errno| pid>** field contains an error number from the *fork*(2) or *exec*(2) sys­
tem calls if the the call failed, and the process id of the shell process if it suc­
cceeded.

**Field**: **<windowgroup>**

The **<windowgroup>** field contains the window group id associated with the win­
dow specified in the *window* field.

**Field**: **<directory>**

The **<directory>** field contains the full or relative pathname of a file system direc­
tory. It may contain shell patterns.

**Field**: **<pattern>**

The **<pattern>** field contains a file name or shell pattern that translates into one
or more file names.

**Field**: **<label>**

The **<label>** field contains the label that will be used for the top button of the *file*
menu.
Field: `<path message>`

If the request succeeded, the `<path message>` field contains a list of file names of the form `dirname filename ... filename`, where `dirname` is the full path name of the directory and `filename ... filename` is a list of blank-separated file names within the directory. If the request failed, the field contains an error message.
Miscellaneous Implementation Issues

Pinnable Menus and Override-Redirect

Currently, the X11 protocol has a deficiency where a window manager cannot tell when an application has changed the override-redirect window attribute. This situation comes up in the obvious implementation of pinned menus. Typically, a menu is manipulated with override-redirect set to True. However, when the menu is pinned, the owner of the menu would like the window manager to start managing the window. The obvious way to do this is for the client to simply set override-redirect to False. Unfortunately, the X11 protocol doesn’t provide for automatic notification of the window manager in this case. A similar situation occurs when override-redirect is turned on. In this case, the window manager will simply stop receiving Request events on this window. Typical window managers will still have resources allocated for this window; these should be reclaimed.

In both of these situations, we need clients to explicitly tell the window manager that override-redirect has changed. One possible solution is to have the client unmap the window before changing override-redirect and map the window again afterwards. This will always work with all window managers. However, it forces the client to repaint a window that, from the user’s point of view, should have stayed on the screen.

Another approach might be for the client to use synthetic Map Notify and UnmapNotify events to notify the window manager after the client has modified override-redirect. Unfortunately, this technique doesn’t work well visually. Therefore, clients should unmap the window before changing its override-redirect attribute.

Full Size Window

When the user selects Full Size from the window menu, the window manager should use the max_width and max_height entries from the WM_NORMAL_HINTS properties. As per the OPEN LOOK specification, the origin of the window may have to be moved so that as much as possible of the resized window fits on the screen. If the client has not provided the maximum size fields, or if it hasn’t provided the WM_NORMAL_HINTS property, the window manager should set its own policy for determining the dimensions of a full-sized window. Typically, on a large monitor with a landscape-style aspect ratio, the full height should be the height of the screen, and the width should stay unchanged from...
the current window width. On smaller monitors, the dimensions should be the full height and width of the screen.

When the user selects *Restore Size* from the window menu, the window manager should return the window to the size and location it had before the user selected *Full Size*. 
# 5 Mouseless Operations

## Overview

5-1

## Keyboard Traversal

5-2

## Keyboard Accelerators

Overview 5-5
- Examples 5-5
- Side Effects 5-5
Visual Appearance 5-6

## Mnemonics

Overview 5-7
- Mnemonic Modifier Prefix 5-7
- Visual Appearance 5-7
Widget Activation/Association 5-8
- Widget Activation 5-8
- Widget Association 5-9
Overview

Mouseless operations is a generic term which encompasses all features at the user’s and programmer’s disposal for operating an OPEN LOOK application in an environment without a mouse. There are four major areas in mouseless operations:

- Keyboard Traversal
- Keyboard Accelerators
- Keyboard Mnemonics
- Widget Activation and Association

In addition to the above areas, the mouseless operation within the toolkit depends heavily on the widget’s classing scheme and the event handling mechanism.

Each of the areas are discussed separately in the following sections.

NOTE

Throughout this section words that are capitalized and begin with “OL_” are toolkit tokens used to represent OPEN LOOK commands. For example, OL_NEXT_FIELD is the toolkit token equivalent of the functional specification for the NextField command.
Keyboard Traversal

Keyboard traversal is the ability to use the key sequences to move the keyboard input focus visual to any control or text-input widget within a top level window or to move input focus from one top level window to another. Under normal conditions, there is one top level window on the screen which takes input focus. This window easily is identified as the window with its header colored with the input-window color. Within the input focus window, one of the controls or text widgets will have the input focus. If a control has input focus, the entire control is highlighted with the input-focus color; or if a text widget has focus, the text widget’s caret is colored with the input focus color and is blinking.

Once input focus is within a top level window, traversal between controls and text widgets is achieved with the **OL_NEXT_FIELD** and **OL_PREV_FIELD** keyboard commands. By default, these commands are bound to the TAB (and Ctrl-TAB) and Shift-TAB keys (and Ctrl-Shift-TAB), respectively. As these keys are pressed, the input focus visual moves to the next control or text widget which will to accept it.

---

**NOTE**  
Most read-only text areas such as captions do not accept focus.

After traversing to a desired control, the user can activate that control by pressing the **OL_SELECTKEY** command (which is typically bound to the Spacebar). Activating the control with the **OL_SELECTKEY** command yields the same results as if the user had activated the control with a **OL_SELECT** mouse button click on the control.

Whenever focus is within a container widget such as an Exclusives or Nonexclusives setting or ScrollingList, the **OL_MOVERIGHT**, **OLMOVELEFT**, **OL_MOVEUP** and **OL_MOVEDOWN** commands move focus between the rows and columns of the contained sub-objects. By default, these commands are bound to the arrow keys. Similarly, when input focus is within a text widget, the **CHARFWD**, **CHARBAK**, **ROWUP** and **ROWDOWN** commands move the input caret among the characters. By default, these commands also are bound to the arrow keys. The **OL_NEXT_FIELD** and **OL_PREV_FIELD** keys are used to move input focus in and out of container widgets.

Whenever the last control in a top level window has focus and the **OL_NEXT_FIELD** key is pressed or whenever a sub-object located on the container widget’s border has focus and the direction of the traversal command points outside the container, traversal wrapping occurs. For the control in the top
level window, traversal wrapping causes focus to move to the first control in the top level window. For the sub-object located on the border of its container, focus remains in the container widget but is set to a sub-object in an extreme row or column opposite the sub-object with focus. The following figure illustrates intra-container widget focus traversal for three different exclusives settings. For the following settings, each sub-object displaying an arrow glyph indicates the new focus location if a move command (OL_MOVERIGHT, OL_MOVELEFT, OL_MOVEUP or OLMOVEDOWN) matching the glyph’s direction is pressed while the focus is on the start sub-object.

**Figure 5-1: Sub-Object Traversal within a Container Object**

![Diagram of sub-object traversal](image)

*From an interior sub-object*

*From an edge sub-object*
Traversal among a large number of sub-objects within a container widget can become tedious. An example of this scenario is the File Manager’s Directory Pane that often displays directories containing many files. With many files, the required number of keystrokes needed to reach an arbitrary set of files is typically large. To alleviate this problem, accelerated focus movement between sub-objects is available through the `OL_MULTIRIGHT`, `OL_MULTILEFT`, `OL_MULTIDOWN` and `OL_MULTIUP` commands. Pressing one of these keys has the same effect as pressing one of the `OL_MOVERIGHT`, `OL_MOVELEFT`, `OL_MOVEDOWN` or `OL_MOVEUP` multiple times.

Besides moving focus between controls within a top level window, keyboard traversal also moves focus between top level windows within an application or to top level windows of other applications. The `OL_NEXTWINDOW` and `OL_PREVWINDOW` commands move focus between windows within an application while the `OL_NEXTPAPP` and `OL_PREVAPP` commands move focus between applications. It should be noted that inter and intra application traversal requires the OPEN LOOK window manager to be running.
Keyboard Accelerators

Overview

All controls have the capability of having one or more attached accelerators. An accelerator is a one keystroke sequence that activates a control, giving the same result as if the control were visible and the OL_SELECT mouse button were clicked while the pointer was over that control.

An accelerator for an application can be activated if any window within that application has input focus, even if the control is not in the window with focus. Therefore, the main advantage of a keyboard accelerator over using a mouse is that the control does not have to be visible, allowing the user to limit the number of steps required to activate a particular control.

NOTE
Accelerators do not install key passive grabs on any window by default. If an application wants accelerators to install passive grabs, the application must set the Boolean shell resource XtNacceleratorsDoGrabs to TRUE. If this resource is set to TRUE, a passive grab for each accelerator is installed on each shell within the widget’s window group.

Examples

Here are two examples of how accelerators work:

- The Quit button on a window menu can be activated even though the window menu is not popped up.
- A top level window having a popup window descendant with a cancel button can be activated if an accelerator is bound to that button.

Side Effects

Since accelerators are global to an application, they must be unique for the entire application. If the application attempts to bind more than one control to the same accelerator key sequence, a warning is generated and the new accelerator is ignored. An application can override any OPEN LOOK standard command (for example, OL_DEFAULTACTION, OL_CANCEL, and so on) simply by binding an accelerator to the same key sequence as the OPEN LOOK command.
Visual Appearance

A workspace Miscellaneous Property Sheet setting indicates whether a string representation of an accelerator key sequence should appear in a control’s label. When the accelerator text is visible, it appears to the right of the label. The following figure shows a Quit button with and without its accelerator text.

Note that even though the accelerator visual may not be shown (by selecting the On–Don’t Show setting), the accelerator still functions.

Figure 5-2: Oblong Button with and without an Accelerator Visual
Mnemonics

Overview

Mnemonics provide a mechanism for traversing to a control and activating it. Once the control is activated, the control displays the appropriate keyboard input focus visual since all subsequent keyboard input is directed at that control.

A single keystroke is used to activate a mnemonic. But unlike accelerators which can be operated when any top level window has keyboard focus, a mnemonic’s keystroke only has meaning if keyboard focus is within the top level window containing the desired control and mnemonic. By restricting mnemonics to a top level window, different top level windows can reuse mnemonics provided they are unique to their top level window and that no top level window has an accelerator with the same key sequence. Any keystrokes for mnemonics that exist outside the focus top level window are ignored.

Mnemonics never install passive key grabs on any window.

Mnemonic Modifier Prefix

A modifier prefix is required for mnemonics when the control is not on a menu. The modifier prefix is settable from the Workspace Miscellaneous Property Sheet and has the default binding of Alt. For example, since a property sheet’s Apply button uses the letter “A” for its mnemonic, the keystroke Alt <a> is required to activate the mnemonic. But, if a control on a menu had a mnemonic “A,” the unmodified keystroke <a> activates that menu control. Note that mnemonics are case insensitive.

Visual Appearance

A workspace Miscellaneous Property Sheet setting indicates the type of the mnemonic visual feedback:

- On-Underline
- On-Highlight
- On-Don’t Show
- Off

Mouseless Operations 5-7
Mnemonics

The figure below and to the left shows two Quit buttons with the mnemonic visual set to Off, while the figure on the right shows the same Quit buttons with the mnemonic visual set to On-Underline. (All the Quit buttons also have a visible accelerator.)

Figure 5-3: Oblong Button with and without a Mnemonic Visual

If the On-Don't Show visual preference is selected, the mnemonic visual is not shown, but the mnemonic keystroke still activates the control.

Widget Activation/Association

Widget Activation

All widgets (and gadgets) in the toolkit support programmatic activation through a convenience routine, $\text{OlActivateWidget}$. Indirect widget activation is a fundamental feature of the toolkit’s event handling scheme. In mouseless operations, an widget’s callbacks often must be activated from the keyboard or programmatically. In addition to callback activation, many widgets have several other types of activation. For example, scrollbars have activation types corresponding to each of its six scrolling operations. See the individual widget manual pages for their activation types.

```c
Boolean OlActivateWidget(widget, activation_type, activation_data)

    Widget    widget;
    OLVirtualName activation_type;
    XtPointer activation_data;
```

Each widget has a class activation procedure for handling programmatic activation requests. When a widget receives a valid activation request, its class activation procedure does the necessary actions and returns TRUE, indicating that the request was granted. If a widget cannot satisfy the activation request (for
example, an activation type or the widget is busy), the widget’s class procedure
returns FALSE. If the returned code is FALSE, OlActivateWidget attempts to
activate any widgets that are associated with the widget that returned FALSE.
This sequence happens recursively until one of the widgets returns TRUE or
there are no more associated widgets.

**Widget Association**

Widgets are associated with other widgets by calling OlAssociateWidget. A
good example of widget association is that of the ScrolledWindow widget since
it has two child Scrollbar widgets which it associates with itself at creation time.
Since the scrollbars are associated with the ScrolledWindow widget, calling
OlActivateWidget using the scrolled window widget id and the command
OL_PAGERIGHT, causes the contents of the scrolled window to move one pane to
the right.

Also, when the application adds a child widget to the scrolled window widget,
the scrolled window widget associates itself with that child. This means that
calling OlActivateWidget on that child with a scrolling command will cause
the view to scroll since the scrollbars are now indirectly associated with that
child.

Since the association feature is used by the toolkit’s event handling mechanism,
the child of the scrolled window will scroll automatically if the child has focus
and the user presses Alt+].

The left bracket modified by the Alt key is the default keyboard binding for the
OL_SCROLLRIGHT command.

The scrolling occurs because the toolkit receives the keypress event (which
arrived on the scrolled window widget’s child) and attempts to activate that
child by calling OlActivateWidget with the activation type OL_SCROLLRIGHT.
Since the child doesn’t know that activation type, OlActivateWidget attempts
to activate the widgets associated with the child. This happens recursively and
eventually, the scrollbar widget is activated and scrolls the view.
# Internationalization

## Introduction to Internationalization
- Extended UNIX Code Set (EUC) 6-1
- Multibyte Processing and Wide Character Format 6-2

## The Internationalization of the OPEN LOOK Toolkit
- Changes to the Toolkit 6-5
- Localizing OPEN LOOK Applications 6-6
- Internationalizing A Client 6-7
- Locale Announcement 6-8
- The ol/locale_def File 6-9
  - Default Locale Definition File 6-9
  - The Example Japanese Locale Definition File 6-10
- Dynamic Help Message Retrieval 6-12
  - General Message Handling Design 6-12
  - The Class Defines 6-13
- Internationalizing 6-14
- Internationalizing the Strings in an Application 6-15
  - Creating the Application app-defaults Files 6-15
  - Processing Text within an Application 6-16
- Input Method 6-20
  - The Localization Package for the Input Method 6-22
  - Structures 6-23
  - Functions 6-28
- Other Changes 6-30
  - TextEdit Widget 6-30
  - xterm 6-31
Introduction to Internationalization

The UNIX operating system has been used extensively in many countries. The applications developed for it have traditionally provided messages only in English and have operated using English language conventions.

In order for applications to operate correctly in any language, no assumptions may be made about language, code set or local conventions. All such information about “locale” must be stored externally to the application. To do this for both the UNIX system and the OPEN LOOK interface, library functions had to be enhanced and new functions provided to support localizing facilities.

UNIX System V Release 3.1 and later has removed the dependency of the UNIX system on the 7-bit US ASCII code set and includes new extensions which provide support for applications and commands for the non-English speaking user.

The key international capabilities of OPEN LOOK GUI are the following:

- Support of full 8-bit and mult-byte code sets. Commands can handle code sets in which all 8 bits are used.
- Support of alternative date and time formats.
- Enhanced support for character classification and conversion- functions which, for any code set, convert characters from upper to lower case or classify characters as alphabetic, printable, upper or lower case, and so on.
- ANSI C internationalization enhancements, Extended UNIX Characters (EUC) and multibyte processing.
- UNIX System commands which support the use of EUC and multibyte processing.

The OPEN LOOK Release 4i Toolkit provides tools to internationalize applications that make no assumptions about language, code set or local conventions.

Extended UNIX Code Set (EUC)

To enable the use of languages which require characters with encodings other than ASCII, Release 4i provides support for up to four code sets concurrently, at both file and process level.

The external code set represents the set of characters that can be presented to the computer system. The internal code set scheme is called the “Extended UNIX Code” or EUC. EUC comprises a primary code set (code set 0), which is always assigned to the US 7-bit ASCII character set, and three supplementary code sets. The choice of the supplementary code sets is at the system
administrator’s discretion and EUC can support multiple languages concurrently. EUC is provided mainly to support the huge number of ideograms needed for I/O in an Asian Language Environment.

For a given character, its EUC code set is distinguished by the value set of the most significant bit (MSB) of the EUC representation and by single-shift characters. The code sets used at any time (that is, within a single process) are determined by the selected locale. The default locale is the C locale. (See the second part of this chapter for information about choosing locale.) The primary code set (code set 0) is always 7-bit US ASCII. Each byte of any character in supplementary codes (code sets 1, 2, and 3) has the high-order bit set; code sets 2 and 3 are distinguished from code set 1 and each other by their use of a “special shift” byte before each character. SS2 is represented in hexadecimal by \texttt{0x8e}, SS3 by \texttt{0x8f}.

<table>
<thead>
<tr>
<th>Code Set</th>
<th>EUC Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\texttt{0xxxxxxx}</td>
</tr>
<tr>
<td>1</td>
<td>1xxxxxxx [ 1xxxxxxx [ ... ]]</td>
</tr>
<tr>
<td>2</td>
<td>SS2 1xxxxxxx [ 1xxxxxxx [ ... ]]</td>
</tr>
<tr>
<td>3</td>
<td>SS3 1xxxxxxx [ 1xxxxxxx [ ... ]]</td>
</tr>
</tbody>
</table>

Figure 6-1: EUC Code Set Representations

Multibyte Processing and Wide Character Format

To work within the constraints of usual computer architectures, characters are encoded as sequences of bytes, or “multibyte characters.” A multibyte character is character encoded using one or more bytes. An Ascii character is the simplest example of a multibyte character. Because multibyte characters are of varying widths, the sequence of bytes needed to encode a character must be self-identifying: regardless of the supplementary code set used, each byte of a multibyte character will have the high-order bit set; if code sets 2 or 3 are used, each multibyte character will also be preceded by a shift byte.

UNIX System V Release 3.1 introduced a new data type for C programs (\texttt{wchar_t}) which allows all the characters from different code sets, including the primary set, to be represented by codes or wide characters of a uniform length. \texttt{wchar_t} lets you manipulate variable width characters as uniformly sized data.
objects called “wide characters.” Use of the \texttt{wchar\_t} data type often simplifies code that deals with characters because the code need not concern itself with the memory width of every character.

For each wide character there is a corresponding multibyte character and vice versa; the wide character that corresponds to a regular single-byte Ascii character is required to have the same value as its single-byte value, including the null character.

Since there can be thousands or tens of thousands of characters in an Asian-language set, systems should use a 16-bit or 32-bit sized integral value to hold all members. Implementations provide corresponding 16-bit and 32-bit libraries with functions that you can use to manage multibyte and wide characters.

\textbf{Figure 6-2: EUC and Corresponding 16-bit Wide-character Representation}

<table>
<thead>
<tr>
<th>Code Set</th>
<th>EUC Code Representation</th>
<th>Wide-character Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00000000</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>1</td>
<td>1x0000000</td>
<td>1000000001xxxxxxx</td>
</tr>
<tr>
<td></td>
<td>1x0000000 1xxxxxxx</td>
<td>1xxxxxxx1xxxxxxx</td>
</tr>
<tr>
<td>2</td>
<td>SS2 1x0000000</td>
<td>0000000001xxxxxxx</td>
</tr>
<tr>
<td></td>
<td>SS2 1xxxxxxx 1xxxxxxx</td>
<td>0xxxxxxx1xxxxxxx</td>
</tr>
<tr>
<td>3</td>
<td>SS3 1xxxxxxx</td>
<td>1000000000xxxxxxx</td>
</tr>
<tr>
<td></td>
<td>SS3 1xxxxxxx 1xxxxxxx</td>
<td>1xxxxxxx0xxxxxxx</td>
</tr>
</tbody>
</table>
Figure 6-3: EUC and Corresponding 32-bit Wide-character Representation

<table>
<thead>
<tr>
<th>Code Set</th>
<th>EUC Code Representation</th>
<th>Wide-character Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xxxxxxxx</td>
<td>00000000000000000000000000000000</td>
</tr>
<tr>
<td>1</td>
<td>1xxxxxxxx</td>
<td>00110000000000000000000000000000</td>
</tr>
<tr>
<td></td>
<td>1xxxxxxxx1xxxxxxxx</td>
<td>00110000000000000000000000000000</td>
</tr>
<tr>
<td></td>
<td>1xxxxxxxx1xxxxxxxx1xxxxxxx</td>
<td>00110000000000000000000000000000</td>
</tr>
<tr>
<td>2</td>
<td>SS2 1xxxxxxxx</td>
<td>00010000000000000000000000000000</td>
</tr>
<tr>
<td></td>
<td>SS2 1xxxxxxxx1xxxxxxxx</td>
<td>00010000000000000000000000000000</td>
</tr>
<tr>
<td></td>
<td>SS2 1xxxxxxxx1xxxxxxxx1xxxxxxx</td>
<td>00010000000000000000000000000000</td>
</tr>
<tr>
<td>3</td>
<td>SS3 1xxxxxxxx</td>
<td>00100000000000000000000000000000</td>
</tr>
<tr>
<td></td>
<td>SS3 1xxxxxxxx1xxxxxxxx</td>
<td>00100000000000000000000000000000</td>
</tr>
<tr>
<td></td>
<td>SS3 1xxxxxxxx1xxxxxxxx1xxxxxxx</td>
<td>00100000000000000000000000000000</td>
</tr>
</tbody>
</table>
The Internationalization of the OPEN LOOK Toolkit

A number of internal changes have been made to the OPEN LOOK Toolkit in order to make it compatible with internationalization requirements. The Toolkit uses several UNIX-based facilities and a multilingual messaging system which permits all help messages, labels and error messages to be displayed in the end user’s language. The Toolkit also has many routines that have been specifically designed to support an OPEN LOOK Input Method. The OPEN LOOK Input Method permits the use of a traditional keyboard to handle non-English character requirements.

Internationalization can be divided into two different concepts:

- Internationalization: writing a program that makes no assumptions about local customs by separating data from the program logic.
- Localization: providing the data specific to a language, cultural conventions, and code sets.

In order to support the former concept the OPEN LOOK Toolkit has been enhanced to include several routines which retrieve character strings in the end user's language. The entire widget set was modified to employ these routines. In addition, all the code set dependencies, such as the assumption that characters are always one byte long, were removed. The locale specific data for a particular language, is to be supplied by the various companies that are localizing the Toolkit for a particular language group.

Changes to the Toolkit

Several changes or adaptations had to be made to the OPEN LOOK Toolkit in order to support the concepts of internationalization and localization. They are as follows:

1. **Locale Announcement**: The behavior of an X application in an Internationalized environment is governed by information such as, language, character code set and so on. In order to operate correctly in a "localized" environment, an application needs to communicate this information to the underlying operating system. The OPEN LOOK Toolkit provides a locale announcement mechanism by which applications can ask the operating system to configure the appropriate environment.
2. **Text Drawing and Font Grouping Facility**: Different languages require different fonts for drawing characters. In addition, a string in certain languages may contain characters from more than one character set. More than one font may be needed to draw such strings. Since existing X11R4 functions are designed to handle strings from one code set only, a mechanism was designed that allows applications to specify a group of fonts to facilitate drawing of strings with characters from multiple code sets.

3. **Compound Text Translation Facility for ICCC**: The OPEN LOOK Toolkit provides a mechanism to translate EUC encoded strings to Compound Text format (and vice versa) for Inter-Client Communication.

4. **Implementation of an Input Method**: An Input Method enables mapping of keystrokes to characters, possibly with additional dictionaries or other linguistic help. The OPEN LOOK Toolkit provides necessary support for Input Method implementation.

5. **Localized Help Messages and Message Switching**: The help registration facility in the OPEN LOOK Toolkit provides a way for an application to display help messages in the local language. In addition, all references to strings in the Toolkit and clients are language independent.

6. **End User System Clients**: The OPEN LOOK clients in particular, the Work Space Manager, the Window Manager, the File Manager and xterm have been modified to operate in a localized environment. This involves Language and Locale Announcement by the Work Space Manager, supporting EUC file names in the File Manager and using localized text input mode in xterm.

### Localizing OPEN LOOK Applications

The following section summarizes the changes necessary for localizing the Toolkit for a particular locale.

1. Translate OPEN LOOK Toolkit messages (for example `xol_msgs`) to the locale language.
2. Install the message catalogue in the file
   
   /usr/X/lib/locale/locale_name/messages/xol_msgs.

3. Translate all OPEN LOOK client messages files (for example xterm_msgs) to the locale language.

4. Install the client messages in

   /usr/X/lib/local/locale_name/xterm_msgs.

5. Create a locale definition file with resource values for the locale.

6. Install the locale definition file in

   /usr/X/lib/locale/locale_name/ol_locale_def

7. Write an input method library for the locale language (optional).

8. Install the library in /usr/X/lib/locale/locale_name/libname.so.

**Internationalizing A Client**

Use the following steps to internationalize a client:

1. Replace hardcoded ASCII strings with calls to OlGetMessage.

2. Create a default message file containing default (English) versions of messages.

3. Install default messages under /usr/X/lib/app-defaults/appname_msgs (where appname_msgs is the class name used by the client in calls to OlGetMessage, OlVaDisplayWarningMessage).

4. Replace the use of the XFontstruct data type with the OlFontList data type (see the following sections in this Chapter that describe the OlFontList data type).

5. Replace intrinsic-based text metric and drawing calls with the following OPEN LOOK Toolkit routines:
Intrinsics Routine | OPEN LOOK Routine
---|---
XTextWidth | OlTextWidth
font -> ascent | OlMaxFontInfo
XDrawString | OlDrawString

6. Link additional libraries `-lw` and `-ldl` (wide character and dynamic linking) when building applications. See the OPEN LOOK GUI Programmer’s Guide for a complete description of the routines listed above.

**Locale Announcement**

The “localized” behavior of an X application in the International environment is governed by locale specific information such as language, character code set; and cultural conventions such as numeric, date and time formats. This information affects the resource file, text input, and fonts used by applications. For an X application to run correctly in a “localized” environment, it needs to announce the locale under which it intends to operate.

The OPEN LOOK Toolkit performs the locale announcement with the following limitations:

- Multiple locales in the same application are not supported.
- All resource names (as opposed to values) are encoded in the Latin-1 code set.
- The Window Manager always runs in the same locale as the Workspace Manager.
- Applications must make specific provisions to respond dynamically to a change in the value of locale specific resources.
The **ol_locale_def File**

To configure a localized environment, locale specific information is needed. The system administrator must provide such information in a locale definition file under the directory `/usr/X/lib/locale/locale_name/ol_locale_def`.

The information contained in this file is used to determine the localized environment and to look up default values for certain locale specific resources. The format of this file is shown below:

The entries `xnlLanguage`, `displaying`, `inputlang`, `numeric` and `time format` correspond to the `setlocale(3C)` categories `LC_ALL`, `LC_TYPE`, `LC_MSGS`, `LC`, `NUMERIC`, and `LC_TIME`, respectively. Each entry consists of the locale name (to be passed to `setlocale(3C)` by the OPEN LOOK Toolkit) and a quoted format string (to be used by the workspace manager to identify the locale category to the end user in the setlocale property sheet.

The representation for the date, the time and numeric for a locale correspond to formats found in files `LC_TIME` and `LC_NUMERIC` under `/lib/locale/locale_name`. For example, if the date format in the file `LC_TIME` is `"%m%d%y"` then the format string for time format should be `mm/dd/yy`.

Also, note that, format strings specified in a locale definition file are intended to be used by the Work Space Manager only. Actual values of the resources `time format` and `numeric` are specified as a locale name for use with `setlocale()`.

**Default Locale Definition File**

The following default locale definition file `ol_locale_def` is installed in the directory `/usr/X/lib/locale/C`.

---

**NOTE**

The representation for the date, the time and numeric for a locale correspond to formats found in files `LC_TIME` and `LC_NUMERIC` under `/lib/locale/locale_name`. For example, if the date format in the file `LC_TIME` is `"%m%d%y"` then the format string for time format should be `mm/dd/yy`.

Also, note that, format strings specified in a locale definition file are intended to be used by the Work Space Manager only. Actual values of the resources `time format` and `numeric` are specified as a locale name for use with `setlocale()`.
The Internationalization of the OPEN LOOK Toolkit

*xnlLanguage:  C "English"
*displayLang: C "English"
*inputLang: C "English"
*numeric: C "10,000"
*timeFormat: C "5:30PM" "12/31/90"
*fontGroup:  
*fontGroupDef:  
*inputMethod:  
*imStatus: False
*frontEndString: 

The Example Japanese Locale Definition File

The following is an example locale definition resource file for the Japan locale that has been included as part of the Toolkit.

*xnlLanguage: japan "Japan"
*displayLang: japan "Japanese", C "English"
*inputLang: japan "Japanese", English "English"
*numeric: japan "10000", fr "10.000"
*timeFormat: japan "5:30PM 31/12/90", C "17:30 12/31/90"
*fontGroup: mincyo, gothic
*fontGroupDef: mincyo=lucida/kanji/kana/hojo, \n     gothic=goal/gokan/gokana/gokanji
*inputMethod: libolim
*imStatus: True
*frontEndString: 

The resources specified in the local definition file are described below:

xmlLanguage
This contains the name of the current locale under which an application operates.

timeFormat
This resource specifies the name of the locale that supports the desired formats for time and date. The locale name specified by this resource corresponds to the LC_TIME category of setlocale().
inputLang This resource contains the name of the locale that provides the necessary environment for processing text for the desired language.

displayLang This resource specifies the name of the locale in which the messages should be displayed. It corresponds to the LC_MESSAGES category of the setlocale() function.

numeric This resource specifies the name of the locale that supports desired format for specifying a numeric value. The value of this resource corresponds to the LC_NUMERIC category of the setlocale() function.

inputMethod This resource specifies the name of the Input Method used by the application for handling text input in local language. It determines the appropriate localization library name used to establish a connection with the specified Input Method.

imStatus Some languages need to display status information in text input mode. The Input Methods developed for such languages need a window (or widget) to display such information. This resource specifies a Boolean value to indicate whether or not an Input Method requires a status window.

fontGroup Text in some languages may be composed of characters from different code sets and hence requires more than one font to draw it. The fontGroup resource provides an alias for the set of fonts that are used to draw text in those languages.

fontGroupDef This resource provides a way of associating a single name with set of fonts (known as a fontGroup). A single fontGroupDef may contain definitions for multiple fontGroups and separated by commas. It is used in constructing a list of XFontStruct structures for use by text drawing utilities.

frontEndString The string that is atomized by the Input Method and used to identify IM messages sent to the client.
Dynamic Help Message Retrieval

To display help messages in the local language, the help widget was modified such that it reads a localized help message from the locale specific help message file. This file(s) resides under the following sub-directory:

```
/usr/X/lib/locale/locale_name/help/app-name.
```

General Message Handling Design

For upward compatibility, error and warning handling have been implemented using the current functions, `OlVaDisplayErrorMsg()` and `OlVaDisplayWarningMsg()`. These functions, as well as others that they call, were modified to open one or more error databases, retrieve specific localized messages and (if desired) a custom message prefix, provide defaults for retrieval failures, and print formatted messages as before. An option is also provided for an application to silence a Toolkit error or warning message. All Toolkit and client code using other forms of error and warning handling have been converted to use one of these two functions. There is a default (locale C) error database (`libolc.a`) for the Toolkit, one for each supported client, one for the client library, and the capability for any other application to create an error database for itself.

There is also a general message retrieval function `OlGetMessage()` so that the Toolkit or any application can retrieve a localized message and use it for its own specific purpose.

Visible widget string resources fall into two categories: static and dynamic. An example of the former is a button label that remains constant for the duration of a program’s execution while an example of the latter is a footer message which changes periodically at run-time. An application can use its `app-defaulrs` file to set static string resources but must use `OlGetMessage()` for dynamically changing string resources.

The message retrieval functions (`OlGetMessage()`, `OlVaDisplay()`) take name, type, and class arguments to identify a message. The class identifies a file and is used to differentiate among the Toolkit, supported clients, and other applications.
The Class Defines

The reserved class defines for the clients are the following:

```c
#define OleCO1ClientOlamMSgs "olam_msgs"
#define OleCO1ClientOlamErrs "olam_errs"
#define OleCO1ClientOlfmMSgs "olfm_msgs"
#define OleCO1ClientOlfmErrs "olfm_errs"
#define OleCO1ClientOl pixmapMsgs "olpix_msgs"
#define OleCO1ClientOl pixmapErrs "olpix_errs"
#define OleCO1ClientOlprintscreenMsgs "olps_msgs"
#define OleCO1ClientOlprintscreenErrs "olps_errs"
#define OleCO1ClientOlwmMsgs "olwm_msgs"
#define OleCO1ClientOlwmErrs "olwm_errs"
#define OleCO1ClientOlwsmMsgs "olwsm_msgs"
#define OleCO1ClientOlwsmErrs "olwsm_errs"
#define OleCO1ClientXtermMsgs "xterm_msgs"
#define OleCO1ClientXtermErrs "xterm_errs"
```

An application is free to use any other defined type for class, using the above file name format. The same is true for a message name and type, though it is suggested that the name and type be used in an efficient and self-documenting way.

Other reserved strings are **_stop** where "****" can be any OPEN LOOK supported client.

In addition to the files used by the message retrieval functions, the standard application **app-defaults** file is created as usual and contains any static widget string resources requiring localization. For each locale, the files are placed in the `/usr/X/lib/locale/locale_name/app-defaults` directory.

OPEN LOOK Toolkit files are placed in the `/usr/X/lib/locale/locale_name/messages` directory.
Internationalizing

There are three types of strings in the OPEN LOOK source code which need internationalizing: error/warning messages, dynamically changing strings or non-resource strings such as footer messages and widget resource strings such as button labels and mnemonics. String conversion is addressed in a number of ways:

- For both the OPEN LOOK Toolkit and clients, warning and error messages are internationalized using `OlVaDisplayWarningMsg()` and `OlVaDisplayErrorMsg()` calls. Both of these functions take variable argument lists.

- For both the OPEN LOOK Toolkit and clients, dynamically changing strings are internationalized with `OlGetMessage()` calls. Message classes used include `xol_msg` and `xterm_msgs`.

- For the OPEN LOOK Toolkit, application level resource strings are localized using `OlGetMessage()`.

- For clients only, static widget resource strings can be internationalized using the client’s `app-defaults file`.

- Mnemonics should be in the `app-defaults file` also and internationalized, if desired.

  A mnemonic is represented by a one byte character.

- File names, function names, and resource names are examples of strings which do not need to be internationalized.

- The UNIX utility `exstr` is used to determine what strings a source file contains. Refer to the UNIX System V Release 4 Programmer’s Reference Manual for information on using the `exstr(1)` utility.
Internationalizing the Strings in an Application

You can internationalize the application strings by performing the following:

1. Run the UNIX utility `exstr` on the application source code to see what strings exist in the program. See the UNIX System V Release 4 Programmer’s Reference Manual for information about `exstr(1)`.

2. Convert all errors and warnings to `OLVaDisplayErrorMsg()` and `OLVaDisplayWarningMsg()` calls. See the OPEN LOOK GUI Programmer’s Guide and the information provided below.

3. Convert all miscellaneous strings (including string arguments to `OLVaDisplayErrorMsg()` and `OLVaDisplayWarningMsg()`) to `OLGetMessage()` calls.

   All three functions require a class, name, and type parameter; this is explained below.

4. English or default messages can be compiled into the application or reside in a default database, created by the application programmer, as explained below.

5. Define widget string resources and mnemonics in the standard `app-defaults/file`.

Creating the Application app-defaults Files

OPEN LOOK associates a message class, name, and type with each message. Class is currently used to differentiate among the OPEN LOOK Toolkit, the supported OPEN LOOK clients, and other OPEN LOOK applications. A name and type pair is used to map to a unique message within a database file. Thus an entry in an Resource database file for OPEN LOOK has the following format:

   *name.type: <default or local error message string>
The Internationalization of the OPEN LOOK Toolkit

Processing Text within an Application

Languages other than English may require different code set schemes and fonts. In addition, a text string may contain characters from different code sets. Drawing of such text strings requires knowledge of the code set scheme used for a language and the availability of specific fonts for that language. A facility has been provided in the Internationalized OPEN LOOK Toolkit to handle the drawing of encoded strings. See the "Structures" section below.

NOTE

If an application is using a wchar_t string and it calls the function OlDrawString(), then it is the responsibility of the application to convert a wchar_t string to a multibyte string (using wcstombs(3X)) before passing it to the function.

Structures

The following data structures are used to parse an encoded string and to query and store code set and font related information to facilitate drawing of encoded strings. Please see the OPEN LOOK GUI Programmer's Guide for more details.

OlStrSegment

The String Segment Information structure OlStrSegment is primarily intended to be used by a parsing utility. The parsing utility parses an encoded string and returns a sub-string in which all characters belong to the same character code set. The structure definition is as follows:

Typedef struct _OlStrSegment {
    unsigned short code_set; /* EUC code set number */
    int len; /* length of the string segment */
    unsigned char *str; /* string segment */
} OlStrSegment;

OlFontList

The font list structure OlFontList caches the XFontStructs for locale-specific groups of fonts specified via the fontGroup resource. The element num contains the number of entries in the list. This structure is intended to be used with text drawing routines where the supplied graphics context (gc) needs to be updated with the font ID Font of the locale specific font before initiating an actual drawing request.
typedef struct __OlFontList {
    int     num;  /* number of XFontStruct in the list */
    XFontStruct ** fontl; /* list of XFontStruct for local fonts */
    int      *cswidth; /* list of char. width in csname array */
    char     ** csname; /* list of char. code set names */
    char     *fgrpdef;
} OlFontList;

Functions

OlDrawString()

The **OlDrawString** function is a general purpose text drawing utility meant to replace the Xlib function **XDrawString()** for drawing an EUC encoded string.

```c
int OlDrawString(display, drawable, fontlist, gc, x, y, string, len)
Display *display; /* display pointer */
Drawable drawable; /* drawable resource, e.g. window */
OlFontList *fontlist; /* list XFontStruct of localized fonts */
GC gc; /* gc to draw */
int x, y; /* Initial position for drawing string */
unsigned char *string; /* encoded string to be drawn */
int len; /* length of the string in bytes */
```

OlDrawImageString()

The **OlDrawImageString** function is a general purpose text drawing utility meant to replace the Xlib function **XDrawImageString()** for drawing an EUC encoded string.
OlDrawImageString(dpy, drawable, fontlist, gc, x, y, string, len)

Display *display; /* display pointer */
Drawable drawable; /* drawable resource, e.g. window */
OIFontList *fontlist; /* list XFontStruct of localized fonts */
GC gc; /* gc to draw */
int x, y; /* Initial position for drawing string */
unsigned char *string; /* encoded string to be drawn */
int len; /* length of the string in bytes */

OlGetNextStrSegment()

The OlGetNextStrSegment() function is a parsing utility that parses an EUC encoded string and returns a pre-allocated structure that contains a sub-string and necessary information about the sub-string to enable caller to identify/draw it.

int OlGetNextStrSegment(fontlist, parse, str, len)
OIFontList *fontlist;
OILStrSegment *parse;
unsigned char **str;
int *len;

OlCvtFontGroupToFontStructList()

This converter prepares a list of XFontStruct for a specified fontGroup and returns a pointer to OlFontList structure. The returned font list is intended to be used by an application with a call to function OlDrawlmageString. If OlFontList is not NULL, then widgets can use the OPEN LOOK text drawing utilities (for example, OlDrawImageString()).

static Boolean OlCvtFontGroupToFontStructList (display, args, num_args, from, to, CacheRef)
Display *display;
XrmValue *args;
Cardinal *num_args;
XrmValue *from;
XrmValue *to;
XtPointer *cache_ref;
The 01 TextWidth function computes and returns the length of the given string in pixels. The functionality is similar to XTextwidth except that the string may be EUC encoded in multibyte format. The sub-string returned by 01GetNextStrSegment() is used to compute the length of string in pixels.

```c
int OlTextWidth(fontlist, string, len)
FontList    *fontlist;
unsigned char *string;
int len;
```

OlFontInfo

This structure keeps track of the maximum values of various font properties for XFontStruct in the OlFontList structure. It is initialized and returned by the function OlMaxFontInfo(). The information is used by Button and Caption widget drawing routines and may be used by an application to determine maximum ascent, descent, height and width prior to calling drawing utilities. The structure definition is as follows:

```c
typedef Struct _OlFontInfo
{
    int ascent;
    int descent;
    int height;
    int width;
} OlFontInfo;
```

OlMaxFontInfo()

This routine is a general purpose routine that returns a OlFontInfo structure that contains maximum values for various font properties, given an OlFontList.

```c
OlFontInfo
OlMaxFontInfo(fontlist)
OlFontList * fontlist;
```
**Input Method**

Unlike English, some languages require several key strokes to enter one character. Moreover, the same set of key strokes may map to more than one character. For GUIs to behave correctly with these languages in text input mode, a mechanism must exist that maps key strokes to characters, possibly with some linguistic help and user interaction. Such a mechanism is widely known as the Input Method. The implementation of an Input Method may vary depending upon the language. However, all Input Methods perform two major tasks:

1. composing characters from key strokes (commonly known as pre-editing)
2. echoing user key strokes somewhere on screen as an acknowledgement of user action.

Besides variations in the Input Method implementation due to language specific requirements, a particular implementation may also be influenced by other factors such as, its placement of pre-edit window where user key strokes are echoed. The following paragraphs discuss various alternatives for pre-editing.

Based upon the placement of the pre-edit window, the pre-editing phase may be classified into four categories:

**on-the-spot**

In on-the-spot pre-editing, scrolling data lines displayed in the same window as the completed text. This is achieved by moving and scrolling data in the window along the caret every time new key strokes are entered or characters are composed.

**over-the-spot**

The over-the-spot pre-editing implements a separate window to display pre-edit key strokes. This window overlaps the text insertion window to provide a visual feedback which is similar to the on-the-spot. Although, scrolling of data is not required in this category, the pre-edit window itself may need to be moved, so as not to obscure newly completed text.

**off-the-spot**

off-the-spot pre-editing is a simplified version of over-the-spot, in which a pre-edit window is placed directly under the text insertion window. Doing so, does not require moving of pre-edit window since both pre-edit characters and composed characters are visible at the same time (in different windows).
root window

The last category, root window is similar to off-the-spot except that it employs one pre-edit window (usually, at bottom of the screen) for all applications.

Figure 6-4: Four kinds of Input Methods

The OPEN LOOK Toolkit provides the hooks for the over-the-spot method.
Additionally, an Input Method may be implemented either as a *front-end* or a *back-end* localized library. In the *front-end* implementation, Input Method intercepts all key (press) events coming from X server, filters some special keys (for example, "Compose" key), echoes user key strokes in pre-edit window, and sends composed characters to the application text insertion window, usually when a special *compose-key* is entered by the user. In the *back-end* implementation, user key strokes go to the application from X server as usual and the application passes the keys to an Input Method to receive composed character. Both methods may use a localized library or a server to compose characters from user key strokes.

**The Localization Package for the Input Method**

A set of locale dependent functions for the Input Method have been defined. The following list of routines should be provided by any Input Method that is designed to work with the OPEN LOOK Toolkit.

- OlCloseIm
- OlOpenIm
- OlResetlc
- OlImoflc
- OlCreate lc
- OlDestroyic
- OlGetImValues
- OlSetlcFocus
- OlUnsetlcFocus
- OlGetlcValues
- OlSetlcValues
- OlLookupImString
- OlDisplayOfIm
- OlLocaleOfIm

A library containing these functions for a particular locale should be installed in `/usr/X/lib/locale/locale_name`. The name of the library must be in the form `im_name.so`. At runtime, the Toolkit will automatically link in the input method library using the UNIX System V Release 4 dynamic linking facility.
If you wish to use an alternate path for a dynamic Input Method library, then set the shell variable `OLIMLIBPATH` to the directory path that contains the IM library.

**Structures**

The Toolkit requires additional data structures to support Input Methods. These data structures communicate information between the client and the Input Method. They are initialized and manipulated by Input Method functions.

Some of the fields provided in the following structure may not be used for a particular pre-edit implementation. This design assumes a shared pre-edit window, that can be used either as an *under-the-spot* or an *over-the-spot*.

**OlImStyles**

The `OlImStyles` structure contains supported style for the pre-edit window. A convenience function `OlGetImValues` returns a pointer to this structure.

```c
typedef unsigned short OlImStyle;
typedef struct _OlImStyles {
    short styles_count;
    OlImstyle * supported_styles;
} OlImStyles;
```

**OlIm**

The Input Method data structure `OlIm` contains global information about an Input Method. It is created by the Input Method function `OlOpenIm` and destroyed by the `OlCloseIm` function when no longer needed. There is one instance of this structure per application (if the locale under which the application runs, requires one). The structure definition is as follows.
typedef struct _OlIm {
    struct _OlIC *iclist;  /* Input context list */
    OlImStyles  im_styles;  /* supported pre-edit types */
    OlImValuesList imvalues;
    char        *appl_name;  /* application name */
    char        *appl_class; /* application class */
    long        version;    /* OPEN LOOK version */
    void        *imtype;    /* hook for IM specific data */
} OlIm;

OlIC

The Input Context structure OlIC contains necessary information about a text window that is needed when the user enters into pre-edit mode. There is one such structure for each text window in an application. It is used to advise the Input Method on the placement of pre-edit window, status window, their attributes, fonts to use to draw text in the pre-edit window and so on. This structure is initialized by the Input Method OlCreatIC function and is destroyed by the OlDestroy() function when no longer needed. The structure definition is as follows.

    struct _OlIC {
      Window        cl_win;  /* client window ID */
      XRectangle    cl_area; /* client area for pre-edit window */
      Window        focus_win; /* focus window ID */
      OlICWindowAttr s_attr; /* attributes for status window */
      XRectangle    s_area;   /* area for status window */
      OlICWindowAttr pre_attr; /* attributes for pre-edit window */
      XRectangle    pre_area; /* area for pre-edit window */
      OlImStyle     style;   /* styles needed for this IC */
      XPoint        spot;    /* cur. cursor location in text win. */
      struct _OlIm  *im;     /* ptr. to OlIm structure */
      struct _OlIC  *nextic; /* ptr. to next in list */
      void          *ictype; /* IM specific hook */
    } OlIC;

OlICValues

The OlICValues structure defines a list of input context attribute names and value pairs. It is used by the following:
1. an application to communicate to the Input Method the desired attributes for its input contexts

2. by TextEdit widget or xterm to alter/retrieve attribute values

The end of the list is indicated by a NULL value for the attribute name.

```c
typedef struct _OlICValues {
    char * attr_name;
    void * attr_value;
} OlICValues;

typedef OlICValues * OlICValuesList;
```

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlclientArea</td>
<td>XRectangle</td>
</tr>
<tr>
<td>OlclientWindow</td>
<td>Window</td>
</tr>
<tr>
<td>OlclientAttributes</td>
<td>pointer to OlImWinAttributes</td>
</tr>
<tr>
<td>OlstatusArea</td>
<td>XRectAngle</td>
</tr>
<tr>
<td>OlstatusAttributes</td>
<td>pointer to OLImWinAttributes</td>
</tr>
<tr>
<td>OlcursorLocation</td>
<td>XPoint</td>
</tr>
<tr>
<td>OlImFontList</td>
<td>pointer to OlFontList</td>
</tr>
<tr>
<td>OlLineSpacing</td>
<td>unsigned short (number of pixels)</td>
</tr>
</tbody>
</table>

**OlImFunctions**

This structure contains addresses of locale dependent Input Method functions. The structure is initialized by an Input Method stub function `OlSetupInputMethod()`, at the initialization time (and subsequently when an application wants to switch Input Method based on a change in its current locale). On UNIX System V Release 4 systems, the localization package is provided as a dynamic shared library, and in that case the functions' addresses are initialized using the dynamic library linker functions from section (3X). The structure definition is as follows:
The Internationalization of the OPEN LOOK Toolkit

typedef struct _OImFunctions {
  OIm   *(*OIOpenIm)();
  void  (*OICloseIm)();
  OIC   *(*OICreateIC)();
  int   *(*OLookupImString)();
  void  (*OLDestroyIC)();
  void  (*OLSetICFocus)();
  void  (*OLUnsetICFocus)();
  char  *(*OLGetICValues)();
  char  *(*OLSetICValues)();
  char  *(*OResetIC)();
  OIm   *(*OLImOfIC)();
  Display *(*OLDisplayOfIm)();
  char  *(*OLocaleOfIm)();
  void  (*OLGetImValues)();
} OImFunctions;

The following list briefly describes each of the above functions. More information is offered in the next subsection and in the OPEN LOOK GUI Programmer’s Guide.

OIOpenIm() is responsible for opening the locale dependent Input Method and initializing the OIm structure.

OICloseIm() is responsible for closing the Input Method and destroying data structure associated with it.

OICreateIC() is responsible for creating an “input context” within the Input Method. The function also creates and initializes an OIC structure.

OLDestroyIC() destroys an input context created by OICreateIC() function and removes any data structures associated with it.

OLookupImString() is responsible for obtaining a string and/or keysym associated with user key strokes in pre-edit mode.
OlSetICFocus()
sets and keeps track of Input Focus for an input context.

OlUnsetICFocus()
unsets the Input Focus from an input context.

OlGetICValues()
is a convenience function to retrieve input context attributes.

OlSetICValues()
is a convenience function to set/alter input context attributes and description.

OlImOfIC()
returns a pointer to an OlIM structure associated with the specified Input Context.

OlResetIC()
resets the input context to its initial state.

OlDisplayOfIm()
returns a pointer to the display corresponding to the given Input Method.

OlLocaleOfIm()
returns a locale name string under which the specified Input Method runs.

OlGetImValues()
returns a list of properties and features supported by the Input Method.

OlICWindowAttr
This structure contains some window attributes that are needed by an Input Method to implement Toolkit-defined look and feel of the pre-edit window (and perhaps status window). The line_spacing field specifies the space in pixels between lines in a text window. This information may be used by the Input Method for proper placement of a pre-edit window. The cursorcolor and colormap information may remain unused. The structure definition is as follows.
The OImCallback structure provides support for the use of on-the-spot Input Methods. It is used by an application to pass the addresses of pre-editing callback functions to the input method. See OISetICValues (the attribute of OICallbacks) for a description of the callback functions to be provided. Note that the TextEdit widget supports over-the-spot Input Methods and does not use pre-edit callbacks.

```c
typedef struct _OImCallback {
    OImValues   client_data;
    OImProc     callback;
} OImCallback;
```

```c
typedef struct _OICWindowAttr {
    Pixel        background; /* background pixel */
    Pixel        foreground; /* background pixel */
    Colormap     colormap;  /* colormap for pre-edit to use */
    Colormap     std_colormap;
   Pixmap       back_pixmap; /* background pixmap for IM to use */
    OILFontList  fontlist;   /* font list for text drawing */
    int          spacing;    /* line spacing for text lines in PE */
    Cursor       cursor;     /* cursor to use for PE window */
    OImCallback  callback[NUM_IM_CALLBACKS];
} OICWindowAttr;
```

## Functions

All of the functions described in this section are provided by the localization library for the Input Method. These functions are used by the TextEdit widget and the xterm client.

**OlCreatelc()**

The OlCreatelc() is used to register a client text insertion window with Input Method. It is responsible for creating a context within an Input Method and initializing necessary data structures for later use by other Input Method functions that manipulate a pre-edit window or its contents.

```c
OIC *OlCreatelc(im, icvalues_list)
OIm      *im;
OICValues *icvalues;
```
OlDestroyic() The OlDestroyic function is responsible for destroying the input context associated with a text insertion window that was created by the OlCreateic function. The Input Method destroys all internal data structures for the input context.

```c
void OlDestroyic(ic)
OlIc * ic;
```

OlSeticFocus()

The OlSeticFocus() function should set the internal state of an input method to indicate that the text insertion window associated with the ic argument has received the input focus.

```c
void OlSeticFocus(ic)
OlIc * ic;
```

OlUnseticFocus()

The OlUnseticFocus function should set the internal state of an input method and indicate that the last insertion window associated with the ic argument has lost input focus.

```c
void OlUnseticFocus(ic)
OlIc * ic;
```

OlGeticValues()

The caller of the OlGeticValues function passes a OlIcValues list with a set of attribute names whose values are desired. The function fills in the values for those attributes and returns it to the caller. The input method may unmap the status window for the input context and ungrab the keyboard.

```c
char* OlGeticValues(ic, icvalues)
OlIc * ic;
OlIcValues icvalues;
```

OlSeticValues()

The OlSeticValues function is responsible for setting/altering the fields of the OlIc structure based on the attribute values passed. The caller of this function passes an OlIcValues list containing a list of (attribute, value-name,) pairs. The function updates the OlIc structure appropriately.
The Internationalization of the OPEN LOOK Toolkit

char
OlSetICValues(ic, icvalues)
OlIC * ic;
OlICValues icvalues;

OlLookupImString()

The OlLookupImString function is similar to the Xlib function, XLookupString() except that it takes an extra ic argument and may or may not return any string for the given key( press) event. It serves as an entry-point into the Input Method. The function sets up the necessary environment for pre-edit mode. It returns a composed character when available.

int
OlLookupImString(event, ic, buffer_return, buffer_len,
    keysym_return, status_return)
KeyEvent *event;
OlIC * input_context;
char *buffer_return;
int buffer_len;
KeySym * keysym_return;
OlImStatus * status_return;

Other Changes

The following changes are here for your information only. The changes are for the most part internal to the Toolkit.

TextEdit Widget

The TextEdit widget provides full text editing functionality in the OPEN LOOK widget set. The TextEdit widget supports the input and storage of multibyte characters. It hides details of the Input Method from the application programmer. All OPEN LOOK utilities that manipulate TextEdit widget data have been modified to handle multibyte characters. (Backwards compatibility has been preserved.) The TextEdit widget supports over-the-spot input methods. (Both front-end and back-end input methods are supported.)
xterm

xterm provides terminal emulation capabilities in the OPEN LOOK environment. It provides users with a UNIX shell environment and text editing ability. Because xterm is a self-contained application in the sense that it does not rely on a TextEdit widget for text processing, it provides the same Input Method mechanism provided by the TextEdit widget. Being an OPEN LOOK application, xterm uses the OlInitialize() routine and hence it automatically inherits the locale announcement mechanism.
7 Extensive Widget Sampler Program

Design Objectives 7-1

Program Description 7-3
Design Objectives

The sampler program includes all of the widgets, gadgets, and flattened widgets that comprise the OPEN LOOK toolkit.

The program is of value both to the applications programmer and to the end-user. The end-user can get a practical sense of what it is like to work with each of the widgets. The applications programmer can use it as a springboard for initial development since it presents a step-by-step approach to creating and using each one of the OPEN LOOK widgets.

The program is densely commented and, therefore, is not annotated.

Most of the widget definitions use at least a subset of possible resources to demonstrate some of its functionality. Often, where a particular coding objective could be achieved in several ways, both are used as illustration - for example, in naming widgets and setting widget resources.

Excerpts from this example appear as a section in Chapter 3 of this document. The excerpts include the creation and use of the textfield widget to enter data, the slider widget, the stub widget and the placement of widgets on the form. The excerpts are heavily annotated.

The source code of the program is included in the product release and can be found in /usr/X/lib/tutorial/Xol/s_sampler.c.

The following figure shows the output of the widget sampler.
Design Objectives

Figure 7-1: The Open Look Widget Sampler

![Diagram of Open Look Widget Sampler](image-url)

This text is in a scrolled window. You can move left and right with the two scrollbars.

OPEN LOOK GUI Programmer's Guide
Program Description

1 /* ***********************************************
2 /* OPEN LOOK WIDGET SAMPLER : prototype widgets and use of form
3 /*
4 /* Copyright (c) 1989 AT&T
5 /* Copyright (c) 1988 Hewlett-Packard Company
6 /* Copyright (c) 1988 Massachusetts Institute of Technology
7 /************************************************

8 #include <stdio.h>

9 /*
10 * Headers required for all OPEN LOOK applications.
11 */

12 #include <X11/Intrinsic.h>
13 #include <X11/StringDefs.h>
14 #include <Xol/OpenLook.h>

15 /*
16 * Headers required for creating widget instances.
17 */

18 #include <Xol/AbbrevMenu.h>
19 #include <Xol/BulletinBo.h>
20 #include <Xol/Button.h>
21 #include <Xol/Caption.h>
22 #include <Xol/CheckBox.h>
23 #include <Xol/ControlAre.h>
24 #include <Xol/Exclusives.h>
25 #include <Xol/FooterPane.h>
26 #include <Xol/Form.h>
27 #include <Xol/Gauge.h>
28 #include <Xol/MenuButton.h>
29 #include <Xol/Nonexclusi.h>
30 #include <Xol/Notice.h>
31 #include <Xol/OblongButt.h>
32 #include <Xol/PopupWindo.h>
33 #include <Xol/RectButton.h>
34 #include <Xol/Scrollbar.h>
Program Description

```c
#include <Xol/ScrolledWi.h>
#include <Xol/ScrollingL.h>
#include <X11/Shell.h>
#include <Xol/Slider.h>
#include <Xol/StaticText.h>
#include <Xol/Stub.h>
#include <Xol/TextEdit.h>
#include <Xol/TextField.h>

/*
 * Headers required for creating flat widget instances.
 */
#include <Xol/FExclusive.h>
#include <Xol/FNonexclusive.h>
#include <Xol/FCheckBox.h>

/*
 * icon.xpm is the pixmap for the application’s icon
 */
#include "icon.xpm"

/*
 * Defines.
 */
#define MAXTEXT 1000
#define MAXBUF 100

/*
 * Ol_PointToPixel scales to the current screen resolution.
 */
#define N1_H_PIXEL Ol_PointToPixel(OL_HORIZONTAL, 1)
#define N1_V_PIXEL Ol_PointToPixel(OL_VERTICAL, 1)
#define N10_H_PIXELS Ol_PointToPixel(OL_HORIZONTAL, 10)
#define N10_V_PIXELS Ol_PointToPixel(OL_VERTICAL, 10)
#define N50_V PIXELS Ol_PointToPixel(OL_VERTICAL, 50)
#define N100_H_PIXELS Ol_PointToPixel(OL_HORIZONTAL, 100)
```

OPEN LOOK GUI Programmer's Guide
Program Description

```c
#define N100_V_PIXELS Ol_PointToPixel(OL_VERTICAL, 100)
#define N150_H_PIXELS Ol_PointToPixel(OL_HORIZONTAL, 150)
#define N150_V_PIXELS Ol_PointToPixel(OL_VERTICAL, 150)
#define N200_H_PIXELS Ol_PointToPixel(OL_HORIZONTAL, 200)

/*
* Global variables.
*/

static Pixel red_pixel, blue_pixel, purple_pixel, green_pixel,
yellow_pixel, orange_pixel, skyblue_pixel;

static int i;

static Boolean rainbow = FALSE;

/*
* Note: In many cases, a widget is labelled by making it the
* child of a caption, which itself is a child of the form.
*/

static Widget toplevel,
controlarea, cabutton,
form,
gauge,
abbrevmenu_caption, /* caption for abbreviatedmenubutton */
bb_caption, /* caption for bulletinboard */
ca_caption, /* caption for controlarea */
caption, /* "True" caption */
ct_caption, /* caption for checkbox */
f_caption, /* caption for flats */
fm_caption, /* caption for form */
gd_caption, /* caption for gadgets */
g_caption, /* caption for gauge */
slist_caption, /* caption for scrollinglist */
slider_caption, /* caption for slider */
te_caption, /* caption for textedit */
sw_caption, /* caption for scrolledwindow */
nonexclusives, nebutton2, nebutton3,
noticeshell, noticebox,
```

Extensive Widget Sampler Program
Program Description

101 popupshell10,
102 popupshell11,
103 popupshell12,
104 scrollbar,
105 stub,
106 footer_text;

107 /*
108 * Form constraint resources used for most widgets in application.
109 */

110 static Arg genericARGS[] = {
111     { XtNxRefName, NULL },
112     { XtNyRefName, NULL },
113     { XtNxOffset, (XtArgVal) 0 }, /* to be initialized below */
114     { XtNyOffset, (XtArgVal) 0 }, /* to be initialized below */
115     { XtNxAddWidth, (XtArgVal) TRUE },
116     { XtNyAddHeight, (XtArgVal) TRUE },
117 }

118 /*
119 * UTILITY FUNCTIONS
120 */

121 /*
122 * Function to vary value displayed in gauge widget.
123 */

124 static void
125 ChangeGauge(widget)
126        Widget widget;
127 {
128     Arg arg;
129     int current_value;
130     XtSetArg(arg, XtNsIderValue, &current_value);
131     XtGetValues(widget, &arg, 1);
132     current_value+=10;

7-6 OPEN LOOK GUI Programmer's Guide
Program Description

133    if(current_value>100)
134        current_value=0;
135    /*
136    * The following convenience routine is faster than a SetValues.
137    */
138    OlSetGaugeValue(gauge,current_value);
139    }
140    /*
141    * Xt and Xlib examples to give functionality to the stub widget.
142    */
143    static void
144    DrawAndPrint(widget,xevent,region)
145    Widget widget;
146    XEvent *xevent; /* not used */
147    Region region; /* not used */
148    {
149        Display *display;
150        static Window window;
151        static Boolean done1 = FALSE, done2 = FALSE;
152        static GC gc[6];
153        XGCValues values;
154        int i, ix5, ix10, coord, axis, finish = 180*64;
155        static int hpixel,vpixel;
156        Font fid = (Font)0;
157        char *fontname = "*12bluci";
158        XtGCMask mask;
159
160        display = XtDisplay(widget);
161        window = XtWindow(widget);
162
163        if(!done1) {
164            hpixel = (int) N1_H_PIXEL;
165            vpixel = (int) N1_V_PIXEL;
166            if(hpixel==0)
167                hpixel=1;
168            if(vpixel==0)
169                vpixel=1;
Program Description

Programmer's Guide

7-8

```c
168     done1 = TRUE;
169 }
170     if(rainbow) {
171     values.line_width = 5*hpixel;
172     if(!done2) {
173     mask = (XtGCMask) (GCForeground | GCLineWidth);
174     values.foreground = red_pixel;
175     gc[0]= XtGetGC(widget,mask, &values);
176     values.foreground = orange_pixel;
177     gc[1]= XtGetGC(widget,mask, &values);
178     values.foreground = yellow_pixel;
179     gc[2]= XtGetGC(widget,mask, &values);
180     values.foreground = green_pixel;
181     gc[3]= XtGetGC(widget,mask, &values);
182     values.foreground = blue_pixel;
183     gc[4]= XtGetGC(widget,mask, &values);
184     values.foreground = purple_pixel;
185     gc[5]= XtGetGC(widget,mask, &values);
186     }
187 }
188     for(i=0;i<6;i++) {
189     ix5 = i*5;
190     ix10 = i*10;
191     axis = 95 - ix10;
192     coord = ix5 + 5;
193     XDrawArc(display, window, gc[i],
194     coord*hpixel,
195     coord*vpixel,
196     (unsigned int) (axis*hpixel),
197     (unsigned int) (axis*vpixel),
198     0,finish);
199 }
200 }
```
else
    XClearWindow(display,window);

/*
 * Get font for this screen resolution.
 */

fontname[0] = OlGetResolution(XtScreen(widget));
fid = XLoadFont(display,fontname);

/*
 * If successful, label the widget.
 */

if(fid != (Font) 0) {
    values.font = fid;
    gc[0] = XtGetGC(widget,(XtGCMask) GCFont, &values);
    XDrawString(display,window,gc[0],5*hpixel,
    70*vpixel,"STUB WIDGET",11);
    XtReleaseGC(stub,gc[0]);
    XUnloadFont(display,fid);
}

/*
 * This function merely varies the value displayed in the gauge.
 */

ChangeGauge(gauge);

/*
 * Function for changing the footer message.
 */

static void FooterMessage(footer_text,buf)
Widget footer_text;
char buf[];
{
    Arg arg;
    XtSetArg(arg,XtNstring,buf);
Program Description

```c
XtSetValues(footer_text,&arg,1);

/*****************************************************************************/
/* Function for getting color values for the display. */

static void
GetColors
{
    XtrmValue fromValue,toValue;

    static char *colors[] = {
        "purple","blue","green",
        "yellow","orange","red","skyblue"
    };

    int ncolors = 7;
    for(i=0;i<7;i++) {
        fromValue.size = sizeof(colors[i]);
        fromValue.addr = colors[i];
        XtConvert(toplevel,
                  XtRString,&fromValue,XtRPixel,&toValue);
        switch(i) {
          case 0:
            purple_pixel = *((Pixel *)toValue.addr);
            break;
          case 1:
            blue_pixel = *((Pixel *)toValue.addr);
            break;
          case 2:
            green_pixel = *((Pixel *)toValue.addr);
            break;
          case 3:
            yellow_pixel = *((Pixel *)toValue.addr);
            break;
          case 4:
            orange_pixel = *((Pixel *)toValue.addr);
            break;
          case 5:
            break;
          case 6:
            break;
        }
    }
```
Program Description

```c
    case 5:
        red_pixel = *((Pixel *)toValue.addr);
        break;
    case 6:
        skyblue_pixel= *((Pixel *)toValue.addr);
        break;
    }
}

    /*
    * Function for positioning widgets on the form.
    * 
    * Note: XtNx[y]RefName are used instead of XtNx[y]RefWidget so that resources for each widget can be set in an .Xdefaults file. In addition, this also allows specifying the placement of any widget on the form without first having to place the corresponding reference widget(s) on the form. (This would NOT be the case with the XtNx[y]RefWidget resource).
    */

    static void
    SetPosition(widget, xwidget, ywidget)
    Widget widget;
    char *xwidget, *ywidget;
    {
        static int nargs;

        if(nargs == 0)
            nargs = XtNumber(genericARGS);

        genericARGS[0].value = (XtArgVal) xwidget;
        genericARGS[1].value = (XtArgVal) ywidget;
        XtSetValues(widget,genericARGS,nargs);
    }

    /*
    * Function for creating a custom icon for the application.
    */
```
static void SetProgramIcon(toplevel)

    Widget toplevel;

{
    Pixmap program_icon;
    Arg arg[2];
    int i;
    Screen *screen;

    screen = XtScreen(toplevel);

    /* 
    * "icon_" values from icon.xpm 
    */

    program_icon = XCreatePixmapFromData(XtDisplay(toplevel), 
                                          RootWindowOfScreen(screen), 
                                          DefaultColormapOfScreen(screen), 
                                          icon_width, icon_height, 
                                          DefaultDepthOfScreen(screen), 
                                          icon_ncolors, 
                                          icon_chars_per_pixel, 
                                          icon_colors, 
                                          icon_pixels);

    i=0;
    XtSetArg(arg[i], XtIconPixmap, ( XtArgVal) program_icon); i++;
    XtSetArg(arg[i], XtIconName, ( XtArgVal) "s_sampler"); i++;
    XtSetValues(toplevel, arg, i);
}

/* 
* Function for creating a special cursor for the stub widget. 
*/

static void SetStubCursor(widget)

    Widget widget;

{
    static Cursor cursor;
Program Description

336  /*
337   * See OlCorsor.c for other cursor possibilities.
338   */
339
340   cursor = GetOlQuestionCursor(XtScreen(widget));
341   XDefineCursor(XtDisplay(widget), XtWindow(widget), cursor);
342 }
343
344  /*
345   * Event handler example to give functionality to the stub widget.
346   */
347
348 static void
349   StubEventHandler(widget, clientData, event)
350   Widget widget;
351   XtPointer clientData;
352   XEvent *event;
353 {
354   XCrossingEvent *xce;
355
356   /* The xce pointer allows referencing to event specifics - see Xlib.h */
357
358   if(event->type==EnterNotify || event->type==LeaveNotify)
359     xce = (XCrossingEvent *) &(event->xcrossing);
360   else
361     return;
362
363   if(event->type == EnterNotify)
364     FooterMessage(footer_text, "Footerpanel: Pointer entered STUB widget");
365   else
366     FooterMessage(footer_text, "Footerpanel: Pointer left STUB widget");
367 }
368
369  /*
370   * CALLBACKS FOR WIDGETS
371   */

Extensive Widget Sampler Program
/* With this callback, each widget passes its index as
   clientData and thus maps to its own footerpanel message. */

static void genericCB(widget,clientData,callData)
    Widget widget;
    XtPointer clientData,callData;
{
    int n = (int) clientData;

    switch(n) {
        case 1:
            FooterMessage(footer_text,
                "Footerpanel: OBLONGBUTTON callback");
            break;
        case 2:
        case 3:
        case 4:
            {
                char buf[MAXBUF];
                sprintf(buf,
                    "Footerpanel: [NON] EXCLUSIVES callback for button %d",n-1);
                FooterMessage(footer_text,buf);
            }
            break;
        case 5:
            FooterMessage(footer_text,
                "Footerpanel: POPUP apply callback");
            break;
        case 6:

400    FooterMessage(footer_text,
401          "Footerpanel: POPUP setdefaults callback");
402    break;

403    case 7:
404    FooterMessage(footer_text,
405          "Footerpanel: POPUP reset callback");
406    break;

407    case 8:
408    FooterMessage(footer_text,
409          "Footerpanel: POPUP resetfactory callback");
410    break;

411    case 9:
412    FooterMessage(footer_text,
413          "Footerpanel: OBLONGBUTTONGADGET callback");
414    break;

415    case 10:
416    FooterMessage(footer_text,
417          "Footerpanel: FLAT select callback");
418    break;
419  }
420 }

421    static void
422    checkboxCB(widget,clientData,callData)
423    Widget widget;
424    XtPointer clientData, callData;
425 {
426    Arg arg;
427    OlDefine position;
428    /*
429    * Get the current value.

Program Description

430  */
431  XtSetArg(arg,XtNposition,&position);
432  XtGetValues(widget,&arg,1);
433  /*
434  * Toggle it.
435  */
436  if(position == (OlDefine) OL_LEFT)
437    XtSetArg(arg,XtNposition,OL_RIGHT);
438  else
439    XtSetArg(arg,XtNposition,OL_LEFT);
440    XtSetValues(widget,&arg,1);
441  }
442  /*
443  * The MenuSelectCB callback is used by the pulldown menu
444  * popup menu, and abbreviatedmenusbutton widget and gadget.
445  */
446  static void
447  menuSelectCB(widget,clientData,callData)
448  Widget widget;
449  XtPointer clientData,callData;
450  {
451    int n = (int) clientData;
452    char buf[MAXBUF];
453    sprintf(buf,"Footerpanel: Button %d selected",n);
454    FooterMessage(footer_text,buf);
455  }
456  
457  static void
458  nonexclusivesCB(widget,clientData,callData)
459  Widget widget;
460  XtPointer clientData,callData;
461  {
462    Widget parent = (Widget) clientData;
463    */
* If only "More" button, create "Fewer" button.

```c
if(widget == nebutton2 && nebutton3 == (Widget) 0) {
    nebutton3 = XtCreateManagedWidget
    ("Fewer", rectButtonWidgetClass, parent, NULL, 0);
    XtAddCallback(nebutton3, XtNselect, nonexclusivesCB, (XtPointer) nonexclusives);
    XtAddCallback(nebutton3, XtNselect, genericCB, (XtPointer) 4);
}
```

* If all three buttons, delete "Fewer" button.

```c
else if(widget == nebutton3 && nebutton3 != (Widget) 0) {
    XtDestroyWidget(nebutton3);
    nebutton3 = (Widget) 0;
}
```

```c
static void noticeCB1(widget, callData, clientData)
    Widget widget; /* w = emanating button: where notice does popup */
    XtPointer callData, clientData;
    { 
        Arg arg;
        
        XtSetArg(arg, XtNemanateWidget, (XtArgVal)widget);
        XtSetValues(noticebox, &arg, 1);
        XtPopup(noticeshell, XtGrabExclusive);
    }
```

```c
static void noticeCB2(widget, clientData, callData)
    Widget widget;
    XtPointer clientData, callData;
    { 
```
static void popupCB(widget, clientData, callData)
{
    XtPointer clientData, callData;
    XtPopup(clientData, XtGrabNone);
}

/*
 * The rainbowCB callback is for the RAINBOW button.
 */

static void rainbowCB(widget, clientData, callData)
{
    XtPointer clientData, callData;
    if (rainbow)
        rainbow = FALSE;
    else
        rainbow = TRUE;

    DrawAndPrint(stub, (XEvent *)0, (Region)0);
}

/*
 * This callback is used by all three scrollbars.
 */

static void scrollbarCB(widget, clientData, callData)
{
    XtPointer clientData, callData;
    OlScrollbarVerify *sbv = (OlScrollbarVerify *) callData;
    int n = (int) clientData;
}
532    char buf[MAXBUF];

533    sbv->ok = TRUE;

534    switch(n) {
535        case 0:
536            sprintf(buf,
537                    "Footerpane1: Form scrollbar moved to %d%%",
538                    sbv->new_location);
539            break;
540        case 1:
541            sprintf(buf,
542                    "Footerpane1: Scrolledwindow vertical scrollbar moved");
543            break;
544        case 2:
545            sprintf(buf,
546                    "Footerpane1: Scrolledwindow horizontal scrollbar moved");
547            break;
548    }
549    FooterMessage(footer_text,buf);

550    /*
551     * Update the form's scrollbar page indicator.
552     */
553    if(widget==scrollbar)
554        sbv->new_page = (int) (sbv->new_location/10) + 1;
555
556    static void
557    scrollinglistCB(widget,clientData,callData)
558        Widget widget;
XtPointer clientData, callData;
{
    OIListToken token = (OIListToken) callData;
    OIListItem *selected_listitem;
    char buf[MAXBUF];

    /*
     * This macro identifies the item selected.
     */

    selected_listitem = OIListItemPointer(token);

    sprintf(buf,
        "Footerpanel: SCROLLING LIST : Item label %s Data stored %d",
        selected_listitem->label, selected_listitem->user_data);
    FooterMessage(footer_text, buf);
}

static void sliderCB(widget, clientData, callData)
{
    Widget widget;
    XtPointer clientData, callData;
    
    Arg arg;

    /*
     * Slider returns current value.
     */
    arg.value = (XtArgVal) *callData;
    XtSetArg(arg, XtNbackground, arg.value);
    XtSetValues(stub, &arg, l);
}

static void textfieldCB(widget, clientData, callData)
{
    Widget widget;
    XtPointer clientData, callData;

    OITextFieldVerify *tfv = (OITextFieldVerify *) callData;

char buf[MAXBUF];
Arg arg;

sprintf(buf,"Footerpane1: TEXTFIELD User Input: %s, 
tfv->string); 
FooterMessage(footer_text,buf);

XtSetArg(arg,XtNstring, (XtArgVa1) ....);
XtSetVa1ues(widget,&arg,1);

/*
* THE WIDGET TREE:
*
* *
* Toplevel
* *
* |
* Footerpanel
* / x
* Form Statictext
* /
* Remaining OPEN LOOK widgets, gadgets, and flats
*/

/*
* MAIN
*/

int main(argc,argv)
int argc;
char *argv[];
{

Arg arg[10];

/*
* Initialize the environment.
*/

{
toplevel = OIInitialize( "s_sampler",
                     "s_sampler",
                     (XrmOptionDescRec *) NULL,
                     (Cardinal) 0,
                     &argc,
                     argv);

i=0;
XtSetArg(arg[i],XtNtitle,(XtArgVal) "s_sampler"); i++;
XtSetValues(toplevel,arg,i);

/*
 * Get colors to use later.
 */
GetColors;

/*
 * Set the pixmap for program icon window
 */
SetProgramIcon(toplevel);

/*
 * Set FORM resources for the specific screen.
 */

genericARGS[2].value = (XtArgVal) N10_H_PIXELS;
genericARGS[3].value = (XtArgVal) N10_V_PIXELS;
}

/*
 * Make all the widgets and then do placement on the form last.
 */

/*
 * FOOTER_PANEL: This will be the child of toplevel, with
 * the FORM as the topchild and STATICTEXT as the footer child.
 */

{ Widget footerpanel;
footerpanel = XtCreateManagedWidget("footerpanel",
    footerPanelWidgetClass,toplevel,NULL,0);

/*
* FORM: Form's caption is the top child of the footerpanel.
*/

i = 0;
XtSetArg(arg[i],XtNposition,(XtArgVal) OL_TOP); i++;
XtSetArg(arg[i],XtNalignment,(XtArgVal) OL_CENTER); i++;
XtSetArg(arg[i],XtNlabel,(XtArgVal) "Form"); i++;
f_caption = XtCreateManagedWidget("fm_caption",
    captionWidgetClass,footerpanel,arg,i);

text = XtCreateManagedWidget("statictext",
    staticTextWidgetClass,footerpanel,arg,i);

/* Register help for the form, using the file "form.help"
*/
(void) OlRegisterHelp( OL_WIDGET_HELP, (XtPointer) form,
    "Form Widget",OL_DISK_SOURCE,"/usr/X/lib/tutorial/Xt/form.help");

/*
* Make STATICTEXT as the footer child of the footerpanel.
*/

i = 0;
XtSetArg(arg[i],XtNstring,
    (XtArgVal) "Static text (read only) for the Footer panel . . . ");
i++;
footer_text = XtCreateManagedWidget("statictext",
    staticTextWidgetClass,footerpanel,arg,i);

/*
* All the remaining widgets will be on the form.
*/
Program Description

688 /*
689 * CAPTION: to label the controlarea, latter to be created next.
690 */

691 i = 0;
692 XtSetArg(arg[i], XtNxResizable, (XtArgVal) TRUE); i++;
693 XtSetArg(arg[i], XtNxAttachRight, (XtArgVal) TRUE); i++;
694 XtSetArg(arg[i], XtNlabel, (XtArgVal) "Controlarea: "); i++;
695 ca_caption = XtCreateManagedWidget("ca_caption",
696 captionWidgetClass, form, arg, i);

697 /*
698 * CONTROLAREA: the control area contains a README oblongbutton with
699 * with a popup statictext message, a popup command window, a popup
700 * property window, and an EXIT button with a notice widget which pops
701 * up when the button is selected.
702 */

703 {
    Widget widget, popupca01;
    static char stext[MAXTEXT];
    FILE *fp = (FILE *) NULL;

    controlarea = XtCreateManagedWidget("controlarea",
    controlAreaWidgetClass, ca_caption, NULL, 0);

    widget = XtCreateManagedWidget("README",
    oblongButtonWidgetClass, controlarea, NULL, 0);
    i=0;
    XtSetArg(arg[i], XtNtitle, "README"); i++;
    popupshell0 = XtCreatePopupShell("popupshell0",
    popupWindowShellWidgetItem, widget, arg, i);

    XtAddCallback(widget, XtNselect, popupCB, popupshell0);

    i = 0;
    XtSetArg(arg[i], XtNupperControlArea, &popupca01); i++;
    XtGetValues (popupshell0, arg, i);
Program Description

719 /*
720 * STATICTEXT: This is used as a README for the application.
721 */

722 if((fp=fopen("/usr/X/lib/tutorial/Xol/statictext.text", "r"))){
723 fread(stext,sizeof(char),MAXTEXT,fp);
724 fclose(fp);
725 }
726 else
727 strcpy(stext,"Statictext message for README0);

728 i = 0;
729 XtSetArg(arg[i], XtNstring, stext); i++;
730 XtCreateManagedWidget("statictext",
731 staticTextWidgetClass,popupca01,arg,i);
732 }

733 /*
734 * POPUP: Used for Command Window.
735 */

736 {
737 Widget widget,popupcall,popupca12,popupfooter1, popupbutton1;
738
739 popupbutton1 = XtCreateManagedWidget("Popup: Command Window",
740 oblongButtonWidgetClass,controlarea,NULL,0);
741
742 /*
743 * Make the popup shell first.
744 */
745 i = 0;
746 XtSetArg(arg[i],XtNtitle,"Command Window"); i++;
747 popupshell1 = XtCreatePopupShell("POUPSSHHELL",
748 popupWindowShellWidgetClass,popupbutton1,arg,i);
749
750 /*
751 * Add callback to popup button now that we have popupshell widget ID.
752 */
753 XtAddCallback(popupbutton1, XtNselect, popupCB, popupshell1);
Program Description

751 /*
752 * The popup window automatically makes three children: upper and
753 * lower control areas and footer: get widget IDs to populate them.
754 */
755    i = 0;
756    XtSetArg(arg[i], XtNupperControlArea, &popupcall); i++;
757    XtSetArg(arg[i], XtNlowerControlArea, &popupcall2); i++;
758    XtSetArg(arg[i], XtNfooterPane1, &popupfooter1); i++;
759    XtGetValues (popupshell1, arg, i);

760 /*
761 * Populate popup upper control area.
762 */

763 /*
764 * Make a caption as a label/prompt for the TEXTFIELD.
765 */

766 i=0;
767    XtSetArg(arg[i], XtNlabe1, (XtArgVal) "'l'extfield: type & type <return> :"); i++;
768    widget = XtCreateManagedWidget("caption",
769        captionWidgetClass,popupcall, arg,i);
770    widget = XtCreateManagedWidget("textfield",
771        textFieldWidgetClass,widget,NDLL,0);

771 /*
772 * Callback to "read" user input when <return> typed.
773 */

774    XtAddCallback(widget,XtNverification,textfieldCB,NULL);

775 /*
776 * Populate popup lower controlarea with buttons: one must be a default.
777 */

778 i = 0;
779    XtSetArg(arg[i], XtNdefault,(XtArgVal) TRUE); i++;
780    XtCreateManagedWidget("Option 1",......
783 oblongButtonWidgetClass, popupca12, arg, i);

784 XtCreateManagedWidget("Option 2",
785 oblongButtonWidgetClass, popupca12, NULL, 0);

786 /*
787 * Add text to the popup footer.
788 */
789 i = 0;
790 XtSetArg(arg[i], XtNborderWidth, 0); i++;
791 XtSetArg(arg[i], XtNstring, "Footer widget for messages"); i++;
792 XtCreateManagedWidget("footer",
793 staticTextWidgetClass, popupfooter1, arg, i);
794 }

795 /*
796 * End of POPUP: Command Window.
797 */

798 /*
799 * POPUP: Used for Property Sheet.
800 */

801 {
802     Widget exclusives, widget, popupca21, popupfooter2, popupbutton2;
803 }

804 /*
805 * XtCallbackRec used for widget SetValues.
806 */

807 static XtCallbackRec popup_applyCBR[] = {
808     { genericCB, (xtPointer) 5 },
809     { (XtCallbackProc) NULL, (xtPointer) NULL },
810 },

811 static XtCallbackRec popup_setdefaultsCBR[] = {
812     { genericCB, (xtPointer) 6 },
813     { (XtCallbackProc) NULL, (xtPointer) NULL },
814 },
static XtCallbackRec popup_resetCBR[] = {
    { genericCB, (XtPointer) 7 },
    { (XtCallbackProc) NULL, (XtPointer) NULL },
};

static XtCallbackRec popup_resetfactoryCBR[] = {
    { genericCB, (XtPointer) 8 },
    { (XtCallbackProc) NULL, (XtPointer) NULL },
};

popupbutton2 = XtCreateManagedWidget("Popup: Property Window",
    oblongButtonWidgetClass, controlarea, NULL, 0);

/*
* Make the popup shell first:
* NOTE: callbacks must be set for creation of the automatic buttons.
* In this example, mapping is to one function but does not have to be.
*/
i = 0;
XtSetArg(arg[0], XtNtitle, "Property Window"); i++;
XtSetArg(arg[i], XtNreset, (XtArgVal) popup_resetCBR); i++;
XtSetArg(arg[i], XtNapply, (XtArgVal) popup_applyCBR); i++;
XtSetArg(arg[i], XtNresetFactory, (XtArgVal) popup_resetfactoryCBR); i++;
XtSetArg(arg[i], XtNsetDefaults, (XtArgVal) popup_setdefaultsCBR); i++;
popupshell2 = XtCreatePopupShell("POPUP SHELL",
    popupWindowShellWidgetClass, popupbutton2, arg, i);

/*
* Add callback to popup button now that we have popupshell widget ID.
*/
XtAddCallback(popupbutton2, XtNselect, popupCB, popupshell2);

/*
* Get widget IDs of popup children needed.
* Note that lower control area ID not needed
* since automatic buttons are created by code above.
*/
i = 0;
XtSetArg(arg[i], XtNupperControlArea, &popupca21); i++;
XtSetArg(arg[i], XtNfooterPanel, &popupfooter2); i++;
XtGetValues (popupshe1l2, arg, i);

/*
 * Populate popup upper control area with EXCLUSIVES and NONEXCLUSIVE.
 * Note that there is no need to populate popup lower control area.
 */
i=0;
XtSetArg(arg[i], XtNlayoutType, OL_FIXEDCOLS); i++;
XtSetValues (popupca21, arg, i);

widget = XtCreateManagedWidget("Exclusives: ",
captionWidgetClass, popupca21, NULL, 0);
i = 0;
XtSetArg(arg[0], XtNlayoutType, (XtArgVal) OL_FIXEDROWS); i++;
XtSetArg(arg[0], XtNmeasure, (XtArgVal) 1); i++;
exclusives = XtCreateManagedWidget("exclusives",
exclusivesWidgetClass, widget, arg, i);
i = 0;
XtSetArg(arg[0], XtNlabel, (XtArgVal) "Choice 1"); i++;
widget = XtCreateManagedWidget("rbutton",
rectButtonWidgetClass, exclusives, arg, i);
XtAddCallback(widget, XtNselect, genericCB, (XtPointer) 2);

widget = XtCreateManagedWidget("Choice 2",
rectButtonWidgetClass, exclusives, NULL, 0);
XtAddCallback(widget, XtNselect, genericCB, (XtPointer) 3);

widget = XtCreateManagedWidget("Choice 3",
rectButtonWidgetClass, exclusives, NULL, 0);
XtAddCallback(widget, XtNselect, genericCB, (XtPointer) 4);

widget = XtCreateManagedWidget("Nonexclusives: ",
captionWidgetClass, popupca21, NULL, 0);
i = 0;
XtSetArg(arg[i], XtNlayoutType, (XtArgVal) OL_FIXEDROWS); i++;
Program Description

XtSetArg(arg[i], XtNmeasure, (XtArgVal) 1); i++;
nonexclusives = XtCreateManagedWidget("nonexclusives",
   nonexclusivesWidgetClass, widget, arg, i);

widget = XtCreateManagedWidget("Choice 1",
   rectButtonWidgetClass, nonexclusives, NULL, 0);
XtAddCallback(widget, xtNselect, genericCB, ( XtPointer) 2);

nebutton2 = XtCreateManagedWidget("More",
   rectButtonWidgetClass, nonexclusives, NULL, 0);
XtAddCallback(nebutton2, XtNselect, nonexclusivesCB, nonexclusives);
XtAddCallback(nebutton2, XtNselect, genericCB, (XtPointer) 3);

/*
 * Add text to the popup footer.
 */
i = 0;
XtSetArg(arg[i], XtNborderWidth, 0); i++;
XtSetArg(arg[i], XtNstring, "Footer widget for messages"); i++;
XtCreateManagedWidget("footer",
   staticTextWidgetClass, popupfooter2, arg, i);
}

/*
 * End of POPUP: Property Sheet.
 */

/*
 * TO MAKE A MENU:
 *
 * Widget Tree for creating menu:
 *
 *         ( parent widget )
 *
 *         |  menubutton ( visible/mouse sensitive symbol for menu )
 *
 *         |  pane ( used for placement of menu button set )
 *
 *  / x
 *
 *  / x

OPEN LOOK GUI Programmer's Guide
button1  button2

Create a menubutton and menu with a pushpin.

{  
  static Widget menubutton, menupane, widget;

  i = 0;
  XtSetArg(arg[i], XtNpushpin, (XtArgVal) OL_OUT); i++;
  XtSetArg(arg[i], XtNlabelType, (XtArgVal) OL_STRING); i++;
  XtSetArg(arg[i], XtNlabelJustify, (XtArgVal) OL_LEFT); i++;
  XtSetArg(arg[i], XtNrecomputeSize, (XtArgVal) TRUE); i++;
  menubutton = XtCreateManagedWidget("Menubutton",
       menuButtonWidgetClass, controlarea, arg, i);

  /*
  * Get the Widget id of the menupane of the menubutton.
  */
  i = 0;
  XtSetArg(arg[i], XtNmenuPane, (XtArgVal) &menupane); i++;
  XtGetValues(menubutton, arg, i);

  /*
  * Make two oblongbuttons on the menupane, with select callbacks.
  */
  widget = XtCreateManagedWidget("One",
       oblongButtonWidgetClass, menupane, NULL, 0);
  XtAddCallback(widget, XtNselect, menuSelectCB, (XtPointer) 1);

  widget = XtCreateManagedWidget("Two",
       oblongButtonWidgetClass, menupane, NULL, 0);
  XtAddCallback(widget, XtNselect, menuSelectCB, (XtPointer) 2);
}

/*
* End of MENU

Extensive Widget Sampler Program
948  */
949  /*
950  * NOTICE: attach to an EXIT button in the control area.
951  */
952 {
953  static Widget noticetext;
954  Widget widget;
955  static Arg noticeARGS[] = {
956      {XtNtextArea, (xtArgVal) &noticetext},
957      {XtNcontrolArea, (xtArgVal) &noticebox},
958  };
959  cabutton = XtCreateManagedWidget("Exit",
960      oblongButtonWidgetClass, controlarea, NULL, 0);
961  /*
962  * Attach notice popup callback to "EXIT" controlarea button.
963  */
964  XtAddCallback(cobutton, XtNselect, noticeCB1, NULL);
965  /*
966  * Create notice popup shell.
967  */
968  noticeshell = XtCreatePopupShell("notice",
969      noticeShellWidgetClass, cabutton, NULL, 0);
970  /*
971  * Get the widget ids of noticebox and textarea of noticebox.
972  */
973  XtGetValues(noticeshell, noticeARGS, XtNumber(noticeARGS));
974  /*
975  * Add text to text area of noticebox.
976  */
Program Description

```
976  */
977
978  i = 0;
979  XtSetArg(arg[i],
980      XtNstring,(XtArgVal)"NOTICE WIDGET:0nconfirm EXIT");
981  i++;
982  XtSetValues(noticetext,arg,i);

983  /*
984  * Add two buttons to noticebox.
985  */
986  */
987  /* First button has an exit callback.
988  */
989     widget = XtCreateManagedWidget("Okay",
990        oblongButtonWidgetClass,noticebax,NULL,O);
991     XtAddCallback(widget, XtNselect,noticeCB2,noticebax);
992  /*
993  * Second button is a no-op that pops down
994  * notice widget without exiting.
995  */
996     XtCreateManagedWidget("Cancel",
997        oblongButtonWidgetClass,noticebax,NULL,O);
998   }
999  /*
1000  * CAPTION
1001  */
1002  {
1003      static Widget widget;
1004      i = 0;
1005      XtSetArg(arg[i],XtNposition,(XtArgVal) OL_TOP); i++;
1006      XtSetArg(arg[i],XtNalignment,(XtArgVal) OL_CENTER); i++;
1007      XtSetArg(arg[i],XtNlabel,(XtArgVal) "Caption"); i++;
```
caption = XtCreateManagedWidget("caption",
    captionWidgetClass,form,arg,i);

widget = XtCreateManagedWidget("*** Rainbow ***",
    oblongButtonWidgetClass,caption,NULL,0);
XtAddCallback(widget, XtNselect, rainbowCB, NULL);
}

/*
 * STUB WIDGET: this will be used as a drawing canvas.
 */

/*
 * First two arguments scale widget to resolution of screen.
 */

i = 0;
XtSetArg(arg[i],XtNheight,(XtArgVal) N100_V_PIXELS); i++;
XtSetArg(arg[i],XtNWidth,(XtArgVal) N100_H_PIXELS); i++;
XtSetArg(arg[i],XtNbackground, skyblue_pixel); i++;

/*
 * DrawAndPrint will be called with an Expose event;
 */

XtSetArg(arg[i],XtNexpose, DrawAndPrint); i++;
stub = XtCreateManagedWidget("stub",stubWidgetClass,form,arg,i);

/*
 * Add an eventhandler to track when the pointer
 * enters and leaves the stub widget window.
 */

XtAddEventHandler(stub,EnterWindowMask | LeaveWindowMask,
    FALSE, StubEventHandler, (XtPointer)NULL);

/*
 * Set the special cursor for stub widget window
 * below once tree realized.
 */

*/

OPEN LOOK GUI Programmer's Guide
Program Description

* BULLETINBOARD: with ABBREVIATEDMENUBUTTON, CHECKBOX,
and TEXTEDIT widgets in it, each within a caption
* labelling the widget.
*/
{
    Widget widget, bulletinboard, abbnbutton, abbmenupane, checkbox;
    Dimension height, yvalue,
    xpad = (Dimension) N10_H_PIXELS,
    ypad = (Dimension) N10_V_PIXELS;

    static char textedit_string[] =
    "This text is in a textedit widget and can be edited."

    /*
    * BULLETINBOARD.
    */

    i = 0;
    XtSetArg(arg[i], XtNposition, (XtArgVal) OL_TOP); i++;
    XtSetArg(arg[i], XtNalignment, (XtArgVal) OL_CENTER); i++;
    XtSetArg(arg[i], XtNlabel, (XtArgVal) "Bulletinboard"); i++;
    bb_caption = XtCreateManagedWidget("bb_caption",
        captionWidgetClass, form, arg, i);

    bulletinboard = XtCreateManagedWidget("bulletinboard",
        bulletinBoardWidgetClass, bb_caption, NULL, 0);

    /*
    * Set some resources for the bulletinboard.
    */

    i = 0;
    XtSetArg(arg[i], XtNborderWidth, (XtArgVal) 2); i++;
    XtSetArg(arg[i], XtNborderColor, (XtArgVal) orange_pixel); i++;
    XtSetValues(bulletinboard, arg, i);
yvalue = ypad;

i = 0;
XtSetArg(arg[i], XtNx, (XtArgVal) xpad); i++;
XtSetArg(arg[i], XtNy, (XtArgVal) yvalue); i++;
XtSetArg(arg[i], XtNlabel, (XtArgVal) "Abbreviated menu button: ");
i++;
abbnenu_caption = XtCreateManagedWidget("abbnenu_caption",
captionWidgetClass,bulletinboard,arg,i);

abbbmenubutton = XtCreateManagedWidget("abbbmenubutton",
abbrevMenuButtonWidgetClass,abbnenu_caption,arg,i);

/*
* Get the Widget ID of the menupane of the abbreviated menu button.
*/

i = 0;
XtSetArg(arg[i], XtNmenuPane, (XtArgVal) &abbbmenupane); i++;
XtGetValues(abbbmenubutton, arg, i);

i = 0;
widget = XtCreateManagedWidget("Button 1",
oblongButtonWidgetClass,abbnenupane,arg,i);
XtAddCallback(widget,XtNselect,menuSelectCB,(XtPointer) 1);

i = 0;
widget = XtCreateManagedWidget("Button 2",
oblongButtonWidgetClass,abbnenupane,arg,i);
XtAddCallback(widget,XtNselect,menuSelectCB,(XtPointer) 2);

/*
* CHECKBOX.
*/

i=0;
XtSetArg(arg[i], XtNheight, &height); i++;
XtGetValues(abbbmenubutton, arg, i);
Program Description

```c
yvalue = yvalue + height + ypad;
i = 0;
XtSetArg(arg[i], XtNx, xpad); i++;
XtSetArg(arg[i], XtNy, yvalue); i++;
XtSetArg(arg[i], XtLabel, (XtArgVal) "Checkbox: "); i++;
cb_caption = XtCreateManagedWidget("cb_caption",
captionWidgetClass, bulletinboard, arg, i));

i = 0;
checkbox = XtCreateManagedWidget("CHECKBOX_LABEL",
checkBoxWidgetClass, cb_caption, arg, i);
XtAddCallback(checkbox, XtNselect, checkboxCB, NULL);

/*
* TEXTEDIT WIDGET.
*/
i=0;
XtSetArg(arg[i], XtNheight, &height); i++;
XtGetValues(checkbox, arg, i);

yvalue = yvalue + height + ypad;
i = 0;
XtSetArg(arg[i], XtNx, (XtArgVal) xpad); i++;
XtSetArg(arg[i], XtNy, (XtArgVal) yvalue); i++;
XtSetArg(arg[i], XtNalignment, (XtArgVal) OL_TOP); i++;
XtSetArg(arg[i], XtLabel, (XtArgVal) "Textedit:"); i++;
te_caption = XtCreateManagedWidget("te_caption",
captionWidgetClass, bulletinboard, arg, i));

i = 0;
XtSetArg(arg[i], XtNheight, (XtArgVal) N50_V_PIXELS); i++;
XtSetArg(arg[i], XtNwidth, (XtArgVal) N150_H_PIXELS); i++;
XtSetArg(arg[i], XtNsourcetype, (XtArgVal)OL_STRING_SOURCE); i++;
XtSetArg(arg[i], XtNsourcetype, (XtArgVal) textedit_string); i++;
XtCreateManagedWidget("textedit",
textEditWidgetClass, te_caption, arg, i));
```

Extensive Widget Sampler Program
/* SCROLLINGLIST */
{
    static Widget scrollinglist;
    static OListToken (*scrollinglistADDfn);
    static OListItem sl_item[10];

    int natoi;
    /*
    * Use a caption to label the scrollinglist.
    */

    i = 0;
    XtSetArg(arg[i], XtNposition, (XtArgVal) OL_TOP); i++;
    XtSetArg(arg[i], XtNalignment, (XtArgVal) OL_CENTER); i++;
    XtSetArg(arg[i], XtNlabel, (XtArgVal) "Scrollinglist"); i++;
    slist_caption = XtCreateManagedWidget("slist_caption",
                   captionWidgetClass, form, arg, i);

    i = 0;
    XtSetArg(arg[i], XtNviewHeight, (XtArgVal) 7); i++;
    scrollinglist = XtCreateManagedWidget("scrollinglist",
                   scrollingListWidgetClass,slist_caption,arg,i);

    /*
    * Get the pointer to the scrollinglist function in order to add items.
    */

    i = 0;
    XtSetArg(arg[i],
             XtNapplAddltem, (XtArgVal) &scrollinglistADDfn); i++;
    XtGetValues(scrollinglist,arg,i);

    natoi = (int) 'A';

    for(i=0; i<10; i++) {
        sl_item[i].label_type = (OlDefine) OL_STRING;
sl_item[i].label = 
    strcpy(XtMalloc((unsigned)7),"ITEM ");
sl_item[i].label[5] = natoi;
sl_item[i].mnemonic = tolower(natoi++);

/*
 * This field is for storing any type of data desired.
 */

 sl_item[i].user_data = (XtPointer) i ;
 (*scrollinglistADDfn)
 (scrollinglist,NULL,NULL,sl_item[i]);
 }


 XtAddCallback(scrollinglist,
 XtNUserMakeCurrent,scrollinglistCB,NULL);
 }

/*
 * SCROLLED WINDOW
 */

{
 static Widget scrolledwindow;

 static char swstring[] =
 "This text is in a
 static text widget
 in a scrolled window.
 You can move it up
 and down or left
 and right with the
 two scrollbars.
"

 i = 0;
 XtSetArg(arg[i],XtNposition,(XtArgVal) OL_TOP); i++;
XtSetArg(arg[i], XtNalignment, (XtArgVal) OL_CENTER); i++;
XtSetArg(arg[i], XtNlabel, (XtArgVal) "Scrolledwindow"); i++;
sw_caption = XtCreateManagedWidget("sw_caption",
    captionWidgetClass, form, arg, i);

i = 0;
XtSetArg(arg[i], XtNheight, N150_V_PIXELS); i++;
XtSetArg(arg[i], XtNwidth, N100_H_PIXELS); i++;
XtSetArg(arg[i], XtNvStepSize, 20); i++;
XtSetArg(arg[i], XtNhStepSize, 20); i++;
scrolledwindow = XtCreateManagedWidget("scrolledwindow",
    scrolledWindowWidgetClass, sw_caption, arg, i);
XtAddCallback(scrolledwindow,
    XtnVSliderMoved, scrollbarCB, (XtPointer)1);
XtAddCallback(scrolledwindow,
    XtnHSliderMoved, scrollbarCB, (XtPointer)2);

/*
   * Make a statictext widget in the scrolled window
   * to scroll with the scrollbars.
   */

i = 0;
XtSetArg(arg[i], XtNheight, N150_V_PIXELS); i++;
XtSetArg(arg[i], XtNstring, swstring); i++;
XtCreateManagedWidget("statictext",
    staticTextWidgetClass, scrolledwindow, arg, i);
}

/*
   * SLIDER
   */

/*
   * The slider will be used to change the background of the stub widget:
   * calculate the number of colors for the screen and set the slider
   * range from 0 to (N-1) with a granularity of 1; see the sliderCB
   * function for the rest of the code/functionality.
   */
Program Description

```
1236  {
1237    Display *display = XtDisplay(toplevel);
1238    int screen = XDefaultScreen(display);
1239    int n, ncolors=2;
1240    Widget w;
1241    n= XDefaultDepth(display,screen);
1242    for(i=1; i<n; i++) {
1243        ncolors = ncolors*2;
1244    }
1245    ncolors = ncolors -1 ;
1246    i = 0;
1247    XtSetArg(arg[i],XtNposition,(XtArgVal) OL_TOP); i++;
1248    XtSetArg(arg[i],XtNalignment,(XtArgVal) OL_CENTER); i++;
1249    XtSetArg(arg[i],XtNlabel,(XtArgVal) "Slider"); i++;
1250    slider_caption = XtCreateManagedWidget("slider_caption",
1251        captionWidgetClass,form,arg,i);
1252    i = 0;
1253    XtSetArg(arg[i],XtNwidth, (XtArgVal) N200_H_PIXELS); i++;
1254    XtSetArg(arg[i],XtNorientation, (XtArgVal) OL_HORIZONTAL); i++;
1255    XtSetArg(arg[i],XtNsliderMax, (XtArgVal) ncolors); i++;
1256    XtSetArg(arg[i],XtNgxgranularity, (XtArgVal) 1); i++;
1257    XtSetArg(arg[i],XtNticks, (XtArgVal) 1); i++;
1258    XtSetArg(arg[i],XtNtickUnit, (XtArgVal) OL_SLIDERVALUE); i++;
1259    XtSetArg(arg[i],XtNdragCBType, (XtArgVal) OL_RELEASE); i++;
1260    w = XtCreateManagedWidget("slider",
1261        sliderWidgetClass,slider_caption,arg,i);
1262    XtAddCallback(w,XtNslerMoved,sliderCB,NULL);
1263 }
1264 /*
1265  * GAUGE
1266  */
1267    i = 0;
1268    XtSetArg(arg[i],XtNposition,(XtArgVal) OL_TOP); i++;
```

Extensive Widget Sampler Program
XtSetArg(arg[i], XtNalignment, (XtArgVal) OL_CENTER); i++;
XtSetArg(arg[i], XtNlabel, (XtArgVal) "Gauge"); i++;

g_caption = XtCreateManagedWidget("g_caption",
    captionWidgetClass, form, arg, i);

i = 0;
XtSetArg(arg[i], XtNwidth, (XtArgVal) N200_H_PIXELS); i++;
XtSetArg(arg[i], XtNorientation, (XtArgVal) OL_HORIZONTAL); i++;
XtSetArg(arg[i], XtNsliderMax, (XtArgVal) 100); i++;
XtSetArg(arg[i], XtNgranularity, (XtArgVal) 10); i++;
XtSetArg(arg[i], XtNticks, (XtArgVal) 10); i++;
XtSetArg(arg[i], XtNtickUnit, (XtArgVal) OL_SLIDERVALUE); i++;
gauge = XtCreateManagedWidget("gauge",
    gaugeWidgetClass, g_caption, arg, i);

/*
* GADGETS: the oblongbutton gadget and menubutton
* gadget are displayed in a bulletinboard, the latter in a
* caption entitled, "Gadgets."
*/

{
    Widget carea, button, menubutton, menupane;

    i=0;
    XtSetArg(arg[i], XtNposition, (XtArgVal) OL_TOP); i++;
    XtSetArg(arg[i], XtNalignment, (XtArgVal) OL_CENTER); i++;
    XtSetArg(arg[i], XtNlabel, (XtArgVal) "Gadgets"); i++;
    gd_caption = XtCreateManagedWidget("gd_caption",
        captionWidgetClass, form, arg, i);

    i=0;
    XtSetArg(arg[i], XtNborderWidth, (XtArgVal) 2); i++;
    XtSetArg(arg[i], XtNborderColor, (XtArgVal) orange_pixel); i++;
    carea = XtCreateManagedWidget("controlarea",
        controlAreaWidgetClass, gd_caption, arg, i);

    i=0;
    XtSetArg(arg[i], XtNlabel, (XtArgVal) "Oblongbutton" ); i++;
    XtSetArg(arg[i], XtN, (XtArgVal) N10_H_PIXELS); i++;

7-42
OPEN LOOK GUI Programmer’s Guide
Program Description

XtSetArg(arg[i], XtNlabel, (XtArgVal) "Menubutton"); i++;
XtSetArg(arg[i], XtNx, (XtArgVal) N100_H_PIXELS);
XtSetArg(arg[i], XtNy, (XtArgVal) N10_V_PIXELS); i++;
memobutton = XtCreateManagedWidget("memobutton",
    menuButtonGadgetClass, carea, arg, i);
   
i = 0;
XtSetArg(arg[i], XtNmenuPane, (XtArgVal) &menupane); i++;
XtGetValues(menobutton, arg, i);
   
i = 0;
button = XtCreateManagedWidget("Button 1",
    oblongButtonGadgetClass, menupane, arg, i);
XtAddCallback(button, XtNselect, menuSelectCB, (XtPointer) 1);
   
i = 0;
button = XtCreateManagedWidget("Button 2",
    oblongButtonGadgetClass, menupane, arg, i);
XtAddCallback(button, XtNselect, menuSelectCB, (XtPointer) 2);
   
} /*

* FLAT WIDGETS: the flatexclusives, flatnonexclusives, and
* flatcheckbox are displayed in a bulletinboard, the latter
* in a caption entitled, "Flat Widgets."
*/

{ 
    Widget widget, carea;
    WidgetClass class;
    int num_subobjects = 3;
    Dimension hspace = N10_H_PIXELS,
    vspace = N10_V_PIXELS;
   
} /*

Extensive Widget Sampler Program
* For flatexclusives and flatnonexclusives: note "customized" fields.

```c
static String fields1[] = {
    XtNbackground, XtNlabel, XtNselectProc, XtNclientData
};
```

typedef struct {
    XtArgValbackground; /* item's background */
    XtArgVallabel; /* item's label */
    XtArgValselect; /* item's select callback */
    XtArgValclientData; /* clientData for callback */
} FlatData1;

```c
static FlatData1 *items1;
```

* For flatcheckbox: note "customized" fields.

```c
static String fields2[] = {
    XtNbackground, XtNlabel, XtNset,
    XtNselectProc, XtNclientData
};
```

typedef struct {
    XtArgValbackground; /* item's background */
    XtArgVallabel; /* item's label */
    XtArgValset; /* item's set status */
    XtArgValselect; /* item's select callback */
    XtArgValclientData; /* clientData for callback */
} FlatData2;

```c
static FlatData2 *items2;
```

```c
items1 = (FlatData1 *)XtMalloc((Cardinal)
    (num_subobjects*sizeof(FlatData1)));
```

```c
items2 = (FlatData2 *)XtMalloc((Cardinal)
    (num_subobjects*sizeof(FlatData2)));
```
Program Description

1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  

Extensive Widget Sampler Program 7-45
Program Description

```c
widget = XtCreateManagedWidget("Nonexclusives: ",
captionWidgetClass, carea, NULL, 0);

class = flatNonexclusivesWidgetClass;
XtCreateManagedWidget("flatnonexclusives", class, widget, arg, i);

items2[0].background = (XtArgVal) red_pixel;
items2[1].background = (XtArgVal) green_pixel;
items2[2].background = (XtArgVal) blue_pixel;
items2[0].label = (XtArgVal) "CheckBox1";
items2[1].label = (XtArgVal) "CheckBox2";
items2[2].label = (XtArgVal) "CheckBox3";
items2[0].set = items2[1].set = items2[2].set = (XtArgVal) TRUE;
items2[0].select = items2[1].select = items2[2].select
   = (XtArgVal) genericCB;
items2[0].clientData =
   items2[1].clientData =
   items2[2].clientData = (XtArgVal) 10;

class = flatCheckBoXWidgetClass;
XtSetArg(arg[i], XtNitems, items2); i++;
XtSetArg(arg[i], XtNnumItems, num_subobjects); i++;
XtSetArg(arg[i], XtNitemFields, fields2); i++;
XtSetArg(arg[i], XtNnumItemFields, XtNumber(fields2)); i++;
XtCreateManagedWidget("flatcheckbox", class, carea, arg, i);
```

```c
/* SCROLLBAR: attach to the right & bottom of the form.
 */

{ static Arg scrollbarARGS[] = {
   /* Widget resources. */
```
Program Description

1433    { XtNproportionLength, 10 },
1434    { XtNshowPage, OL_LEFT },

1435    /*
1436    * Form resources.
1437    */
1438    { XtNResizable,(XtArgVal) TRUE },
1439    { XtNAttachBottom,(XtArgVal)TRUE },
1440    { XtNAttachRight,(XtArgVal) TRUE },
1441    { XtNvOffset,(XtArgVal) TRUE },
1442  };

1443      scrollbar = XtCreateManagedWidget("scrollbar",
1444          scrollbarWidgetClass,form,scrollbarARGS,
1445          XNumber(scrollbarARGS));
1446      XtAddCallback(scrollbar,
1447          XtNssliderMoved,scrollbarCB,(XtPointer) 0);
1448  }

1449    /*
1450    * Position all the widgets on the form.  Note that reference
1451    * names are used.  Note that 10 horizontal and vertical pixels
1452    * are used in most cases; in a few cases special values are used.
1453    */
1454
1455    {  
1456        Arg arg;
1457        static Dimension width1, width2;
1458        static Position x1, x2;

1459        static Arg getArgs1[] = {
1460            { XtNwidth, (XtArgVal) &width1}
1461        };
1462
1463        static Arg getArgs2[] = {
1464            { XtNwidth, (XtArgVal) &width2}
1465        };

Extensive Widget Sampler Program 7-47
Program Description

1467  genericARGS[2].value = (XtArgVal) 0;
1468  genericARGS[3].value = (XtArgVal) 0;
1469  SetPosition(ca_caption,"form","form");
1470  genericARGS[2].value = (XtArgVal) N10_H_PIXELS;
1471  genericARGS[3].value = (XtArgVal) N10_V_PIXELS;
1472  SetPosition(caption,"form","ca_caption");
1473  SetPosition(stub,"form","caption");
1474  /*
1475  * Position bulletinboard to right of
1476  * caption or stub, whichever is wider.
1477  */
1478  XtSetArg(arg,XtNwidth,&:width1);
1479  XtGetValues(caption,&:arg,1);
1480  XtSetArg(arg,XtNwidth,&:width2);
1481  XtGetValues(stub,&:arg,1);
1482  if(width1>width2)
1483  SetPosition(bb_caption,"caption","ca_caption");
1484  else
1485  SetPosition(bb_caption,"stub","ca_caption");
1486  SetPosition(slender_caption,"form","stub");
1487  SetPosition(g_caption,"form","slider_caption");
1488  SetPosition(f_caption,"form","gd_caption");
1489  SetPosition(slist_caption,"bb_caption","ca_caption");
1490  SetPosition(sw_caption,"slist_caption","ca_caption");
1491  genericARGS[2].value = (XtArgVal) (7 * N10_H_PIXELS);
1492  SetPosition(gd_caption,"slider_caption","slist_caption");
1493  genericARGS[2].value = (XtArgVal) N10_H_PIXELS;
1494  /*
1495  * Position scrollbar to right of flat widgets or scrolledwindow,
1496  * whichever extends further to the right of the form.
1497  */
1498  genericARGS[3].value = (XtArgVal) 0;
XtGetValues(f_caption, getArgs1, XtNumber(getArgs1));
XtGetValues(sw_caption, getArgs2, XtNumber(getArgs2));

width1 = width1 + (Dimension) x1;
width2 = width2 + (Dimension) x2;

if (width1 > width2)
    SetPosition(scrollbar, "f_caption", "ca_caption");
else
    SetPosition(scrollbar, "sw_caption", "ca_caption");

genericARGS[3].value = (XtArgVal) N10_V_PIXELS;
}

*/
* Realize the widget tree.
*/

XtRealizeWidget(toplevel);

*/
* The special cursor for the stub widget can be set now
* that it has been realized as part of the widget tree.
*/

SetStubCursor(stub);

*/
* Turn control over to the Xt intrinsics and OPEN LOOK.
*/

XtMainLoop;

} /* MAIN */
Introduction to the Manual Pages

Introduction to General Resources

Application Resources
- XtNbackground
- XtNbeep
- XtNbeepVolume
- XtNcontrolName
- XtNlockName
- XtNmod1Name
- XtNmod2Name
- XtNmod4Name
- XtNmod5Name
- XtNshiftName
- XtNDontCare
- XtNdragRightDistance
- XtNfontColor
- XtNforeground
- XtNhelpDirectory
- XtNhelpModel
- XtNfont
- XtNfontColor
- XtNfontGroup
- XtNforeground
- XtNinputFocusColor
- XtNinputWindowColor
- XtNmenuMarkRegion
- XtNmnemonicPrefix
- XtNmouseDampingFactor
- XtNmouseStatus
- XtNmultiClickTimeout
- XtNmultiObjectCount
- XtNselectDoesPreview
### Table of Contents

- **XtNshowAccelerators** A-11
- **XtNshowMnemonics** A-11
- **XtNtextBackground** A-12
- **XtNtextFontColor** A-12
- **XtNthreeD** A-12

#### Core Resources
- **XtNancestorSensitive** A-13
- **XtNaccelerators** A-14
- **XtNbackground** A-14
- **XtNbackgroundPixmap** A-14
- **XtNborderColor** A-15
- **XtNborderPixmap** A-15
- **XtNborderWidth** A-15
- **XtNcolormap** A-16
- **XtNdepth** A-16
- **XtNdestroyCallback** A-16
- **XtNheight** A-16
- **XtNmanaged** A-17
- **XtNmappedWhenManaged** A-17
- **XtNscreen** A-17
- **XtNsensitive** A-17
- **XtNtranslations** A-18
- **XtNwidth** A-18
- **XtNx** A-18
- **XtNy** A-19

#### Flat Widget Layout
- **XtNgravity** A-19
- **XtNhPad/XtNvPad** A-21
- **XtNhSpace/XtNvSpace** A-21
- **XtNitemGravity** A-22
- **XtNitemMaxHeight/XtNitemMaxWidth** A-22
- **XtNitemMinHeight/XtNitemMinWidth** A-22
- **XtNlayoutHeight/XtNlayoutWidth** A-23
- **XtNlayoutType** A-23
- **XtNmeasure** A-24
- **XtNsameHeight** A-24
- **XtNsameWidth** A-25

#### Manager Widget Resources
- **XtNconsumeEvent** A-25
- **XtNreferenceName** A-26
- **XtNinputFocusColor** A-27
Table of Contents

- XtNreferenceWidget A-27
- XtNtraversalOn A-27
- XtNuserData A-27

Primitive Widget Resources
- XtNaccelerator A-28
- XtNacceleratorText A-28
- XtNconsumeEvent A-29
- XtNfont A-29
- XtNfontcolor A-29
- XtNforeground A-29
- XtNinputFocusColor A-29
- XtNmnemonic A-30
- XtNreferenceName A-30
- XtNreferenceWidget A-30
- XtNtraversalOn A-30
- XtNuserData A-31

Shell Resources
- Base Windows and Popup Windows A-31

Resource Description
- XtNallowShellResize A-34
- XtNbusy A-34
- XtNconsumeEvent A-34
- XtNcreatePopupChildProc A-35
- XtNfocusWidget A-35
- XtNgeometry A-35
- XtNheightInc/XtNwidthInc A-35
- XtNiconic A-36
- XtNiconMask A-36
- XtNiconName A-36
- XtNiconPixmap A-36
- XtNiconWindow A-36
- XtNiconX/XtNiconY A-37
- XtNinitialState A-37
- XtNinput A-37
- XtNmenuButton A-38
- XtNmenuType A-38
- XtNmaxAspectX/XtNmaxAspectY A-38
- XtNminAspectX/XtNminAspectY A-38
- XtNmaxHeight/XtNmaxWidth/XtNminHeight/XtNminWidth A-39
- XtNoverrideRedirect A-40
Table of Contents

- XtNpopupCallback A-40
- XtNpopupCallback A-40
- XtNpopdownCallback A-40
- XtNpushpin A-40
- XtNresizeCorners A-41
- XtNssaveUnder A-41
- XtNtitle A-41
- XtNtransient A-41
- XtNuserData A-42
- XtNwaitForWm A-42
- XtNwindowGroup A-42
- XtNwindowHeader A-42
- XtNwmProtocol A-42
- XtNwmProtocolInterested A-43
- XtNwmTimeout A-43
Introduction to the Manual Pages

This part of the OPEN LOOK® Programmer's Guide offers a detailed description of the various OPEN LOOK widgets, gadgets, and convenience routines that are available to programmers writing client applications for the OPEN LOOK interface. The reference manual is divided into 4 sections.

Section 1 introduces the widgets and routines available to the application programmer and Section 2 discusses General Resources. Resources are the visual and functional characteristics of a particular widget or gadget. Widgets inherit resources from common classes. Some of these super-classes are standard Intrinsics classes such as Core and Shell. Others are OPEN LOOK specific such as Flat, Manager, and Primitive. The section on General Resources is broken down into 6 sub-sections. Each sub-section lists resources of a particular class of widgets. The frequently used resources are grouped as follows:

- Application Resources
- Core Resources
- Common Flat Container Resources
- Manager Widget Resources
- Primitive Widget Resources
- Shell Resources

Section 3 contains the various Convenience Routines available to the application programmer. These routines are presented in alphabetical order without regard to upper or lower case distinctions. The introduction to Section 3 lists the routines by functional groups making it easier to find those routines which perform specific functions such as updating dynamic resources. Other routines work with specific widgets such as TextField and TextEdit. There are also some special Convenience Routines useful for performing such activities as creating widget trees, initializing the OPEN LOOK toolkit, or routines that support input focus. Examples of the functional groups are as follows:

- Cursor/Bitmap Utilities
- Dynamic Setting Utilities
- Regular Expression Utilities
Introduction to the Manual Pages

- Text Buffer Utilities
- Database Utilities
- and others

See the introduction to Section 3 for a full listing of these routines.

Section 4, offers a complete discussion of each widget in alphabetical order in formal manual page format. The widgets are presented in the order below:

- AbbreviatedMenuButton
- BulletinBoard
- Caption
- CheckBox
- ControlArea
- Exclusives
- FlatCheckBox
- FlatExclusives
- FlatNonexclusives
- FooterPanel
- Form
- Gauge
- Menu
- MenuButton (Widget and Gadget)
- Nonexclusives
- Notice
- OblongButton (Widget and Gadget)
- PopupWindow
- RectButton
- Scrollbar
- ScrolledWindow
- ScrollingList
- Slider
- Static Text
- Stub
- TextEdit
- TextField

Section 5, "Obsolete Routines" presents routines and Widgets that were a part of the toolkit but are no longer supported and being phased out. They are here for reference only.
Introduction to General Resources

The following section lists and describes the general resources available to several widgets within a class. We have grouped them here to avoid redundancy on each widget manual page. The resources/operations are grouped according to widget class:

- Application
- Core
- Flat Container
- Manager
- Primitive
- Shell

Application Resources

The OPEN LOOK toolkit uses several resources to determine the state of an OPEN LOOK application. These can be accessed using the OlGetApplicationValues function [see OlGetApplicationValues(3W)]. In order to maintain a consistent look and feel between applications running simultaneously on the same display device, applications do not set these resources directly since the Workspace Manager preference property sheets are responsible for maintaining their values by updating the appropriate resource files.
### Application Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>G</td>
</tr>
<tr>
<td>XtNbeep</td>
<td>XtCBeep</td>
<td>OIDefine</td>
<td>OL_BEEP_ALWAYS</td>
<td>G</td>
</tr>
<tr>
<td>XtNbeepVolume</td>
<td>XtCBeepVolume</td>
<td>int</td>
<td>0</td>
<td>G</td>
</tr>
<tr>
<td>XtNcontrolName</td>
<td>XtCControlName</td>
<td>String</td>
<td>&quot;Ctrl&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNdontCare</td>
<td>XtCDontCare</td>
<td>OIBitMask</td>
<td>LockMask</td>
<td>Mod2Mask</td>
</tr>
<tr>
<td>XtNdragRightDistance</td>
<td>XtCDragRightDistance</td>
<td>Dimension</td>
<td>20 (pixels)</td>
<td>G</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black</td>
<td>G</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>G</td>
</tr>
<tr>
<td>XtNhelpDirectory</td>
<td>XtCHelpDirectory</td>
<td>String</td>
<td>(calculated)</td>
<td>G</td>
</tr>
<tr>
<td>XtNhelpModel</td>
<td>XtCHelpModel</td>
<td>OIDefine</td>
<td>OL_POINTER</td>
<td>G</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>G</td>
</tr>
<tr>
<td>XtNinputWindowColor</td>
<td>XtCInputWindowColor</td>
<td>Pixel</td>
<td>Yellow</td>
<td>G</td>
</tr>
<tr>
<td>XtNlockName</td>
<td>XtCLockName</td>
<td>String</td>
<td>&quot;Lock&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNmenuMarkRegion</td>
<td>XtCMenuMarkRegion</td>
<td>Dimension</td>
<td>10 (pixels)</td>
<td>G</td>
</tr>
<tr>
<td>XtNmnemonicPrefix</td>
<td>XtCMnemonicPrefix</td>
<td>Modifiers</td>
<td>Alt</td>
<td>G</td>
</tr>
<tr>
<td>XtNmod1Name</td>
<td>XtCMod1Name</td>
<td>String</td>
<td>&quot;Alt&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNmod2Name</td>
<td>XtCMod2Name</td>
<td>String</td>
<td>&quot;Mod2&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNmod3Name</td>
<td>XtCMod3Name</td>
<td>String</td>
<td>&quot;Mod3&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNmod4Name</td>
<td>XtCMod4Name</td>
<td>String</td>
<td>&quot;Mod4&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNmod5Name</td>
<td>XtCMod5Name</td>
<td>String</td>
<td>&quot;Mod5&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNmouseDampingFactor</td>
<td>XtCmouseDampingFactor</td>
<td>Cardinal</td>
<td>4 (points)</td>
<td>G</td>
</tr>
<tr>
<td>XtNmouseStatus</td>
<td>XtCmouseStatus</td>
<td>Boolean</td>
<td>True</td>
<td>G</td>
</tr>
<tr>
<td>XtNmultiClickTimeout</td>
<td>XtCMultiClickTimeout</td>
<td>Cardinal</td>
<td>300 (millisec)</td>
<td>G</td>
</tr>
<tr>
<td>XtNmultiObjectCount</td>
<td>XtCMultiObjectCount</td>
<td>Cardinal</td>
<td>3</td>
<td>G</td>
</tr>
<tr>
<td>XtNselectDoesPreview</td>
<td>XtCSelectDoesPreview</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G</td>
</tr>
<tr>
<td>XtNshiftName</td>
<td>XtCShiftName</td>
<td>string</td>
<td>&quot;Shift&quot;</td>
<td>G</td>
</tr>
<tr>
<td>XtNshowMnemonics</td>
<td>XtCShowMnemonics</td>
<td>OIDefine</td>
<td>OL_UNDERLINE</td>
<td>G</td>
</tr>
<tr>
<td>XtNshowAccelerators</td>
<td>XtCShowAccelerators</td>
<td>OIDefine</td>
<td>OL_DISPLAY</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextBackground</td>
<td>XtCTextBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextFontColor</td>
<td>XtCtextFontColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>G</td>
</tr>
<tr>
<td>XtNthreeD</td>
<td>XtCThreeD</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G</td>
</tr>
</tbody>
</table>
XtNbackground
This resource reflects the default background color used by objects which are not scrolling lists or textEdit widgets.

XtNbeep
Range of values:

- OL_BEEP_NEVER/“never”
- OL_BEEP_ALWAYS/“always”
- OL_BEEP_NOTICES/“notices”

This resource determines the type of objects that can generate audible warnings to the user. OL_BEEP_NEVER implies no objects should generate audible warnings. OL_BEEP_ALWAYS implies any object can generate audible warnings. OL_BEEP_NOTICES implies only Notices should generate audible warnings.

XtNbeepVolume
Range of Values:

-100 to +100

This resource specifies a percentage of the keyboard’s normal beep that should be used when generating audible warnings to the user.
XtNcontrolName  
XtNlockName  
XtNmod1Name  
XtNmod2Name  
XtNmod4Name  
XtNmod5Name  
XtNshiftName  

These resources define the names used to display the accelerator in buttons and labels. They are provided as resources so that they can be easily changed for a particular keyboard. For example, the mod1Name is usually "Alt," but on some keyboards it may be labeled "Meta."

XtNdontCare  

Range of Values are determined by ORing the following bit masks:

- LockMask (typically the mask associated with the CAPS LOCK key)
- Mod1Mask
- Mod2Mask (typically the mask associated with the NUM LOCK key)
- Mod3Mask
- Mod4Mask
- Mod5Mask
- ShiftMask
- ControlMask

This resource specifies the modifier bits that are ignored when processing mouse button events. For example, assume the Mod2Mask bit is in the XtNdontCare bits. Now if the NUM LOCK key is in a set state and the user presses the SELECT mouse button, the press is interpreted as a SELECT button press because the Mod2Mask bit is ignored. But if the Mod2Mask bit is not in the XtNdontCare bits, the press is not interpreted as a SELECT button press because the internal event handling routine honors the Mod2Mask bit in the XEvent.
**XtNdragRightDistance**

This resource represents the number of pixels the pointer must be dragged over a MenuButton with the **MENU** mouse button depressed to post the MenuButton’s submenu. The direction of the drag is to the right. This resource only applies to MenuButtons on *press-drag-release* menus.

**XtNfontColor**

This resource reflects the default font color used by objects which are not scrolling lists or text widgets.

**XtNforeground**

This resource reflects the default foreground color used by objects which are not scrolling lists or textEdit widgets.

**XtNhelpDirectory**

This resource specifies the directory in which all help files for an application are located. It is locale-specific. The default value depends on the user’s **XFILESEARCHPATH** environment variable and system configuration, but a typical value is */usr/X/lib/locale/C/help/app-classname*, where everything before “C” is hardcoded somewhere in the **XFILESEARCHPATHDEFAULT** variable.

This resource is used by the **OlFindHelpFile** call and should normally only be set once per application instance.

**XtNhelpModel**

Range of Values:

```
OL_POINTER/"pointer"
OL_INPUTFOCUS/"inputfocus"
```

The OPEN LOOK help model defaults to follow the mouse pointer. So, when the **HELP** key is pressed, the item under the pointer is the subject of the help message. When this resource is set to **OL_INPUT_FOCUS**, the subject of the help message follows input focus.
**XtNfont**

Range of Values:

(any valid return from XLoadQueryFont)

Default:

(chosen to match the scale and screen resolution)

This resource identifies the font to be used to display the text of the widget.

The default value points to a cached font structure; an application should not expect to get this value with a call to XtGetValues and use it reliably thereafter.

**XtNfontColor**

Range of Values:

(any Pixel value valid for the current display)/(any name from the rgb.txt file)

This resource specifies the color for the font.

See the note about the interaction of this resource with other color resources under the description of the XtNbackground resource in Core Resources, “Introduction to General Resources”, Appendix A.

**XtNfontGroup**

Range of Values:

(valid fontgroupname from the ol_locale_def file)

This resource specifies a set of up to 4 fonts to be used when drawing internationalized text with the OPEN LOOK text drawing routines (for example, OlDrawString, OlTextWidth). The first font should always be an ASCII font.

**XtNforeground**

This resource defines the foreground color for the widget.

See the note about the interaction of this resource with other color resources under the description of the XtNbackground resource in Core Resources, “Manual Pages: Introduction”, Appendix A.
XtNinputFocusColor
This resource reflects the default color that controls display whenever the control has input focus.

XtNinputWindowColor
This resource determines the color of the OPEN LOOK window header when it has input focus.

XtNmenuMarkRegion
This resource represents the width (in pixels) of the MenuButton's menu mark region. If the pointer is moved into this region with the MENU mouse button depressed, the MenuButton's submenu is posted.

XtNmnemonicPrefix
This resource specifies the modifier key that must accompany the mnemonic character when activating an object from the keyboard if that object is not on a menu. Note: this value is not settable.

XtNmouseDampingFactor
This resource specifies the number of pixels the pointer can be moved before a drag operation is initiated.

XtNmouseStatus
This Boolean resource indicates whether there is a mouse on the server.

XtNmultiClickTimeout
This resource specifies the number of milliseconds that determines when two mouse button clicks is considered a multi-click, provided the pointer does not move beyond the XtNmouseDampingFactor value.
XtNmultiObjectCount

This resource determines the number of times the OL_MULTIRIGHT, OL_MULTILEFT, OL_MULTIUP and OL_MULTIDOWN keys repeat the OL_MOVERIGHT, OL_MOVELEFT, OL_MOVEUP and OL_MOVEDOWN keys, respectively.

XtNselectDoesPreview

This Boolean resource reflects the behavior of the SELECT mouse button when it is pressed over a MenuButton or an Abbreviated MenuButton. If its value is TRUE, pressing SELECT will cause the MenuButton to preview the submenu’s default item and releasing the SELECT button will activate the default item. If its value is FALSE, pressing SELECT will post the submenu.

XtNshowAccelerators

Range of Values:

OLDISPLAY/“display”
OLINACTIVE/“inactive”
OLNONE/“none”

When this resource is set to OL_DISPLAY, the keyboard accelerators on the controls will be displayed. When this resource is set to OL_NONE, the keyboard accelerators on the controls will not be displayed. Setting this resource to OL_INACTIVE will cause the keyboard accelerators to not be displayed and will cause the controls to ignore the accelerator action.

XtNshowMnemonics

Range of Values:

OLDISPLAY/“display”
OLHIGHLIGHT/“highlight”
OLINACTIVE/“inactive”
OLNONE/“none”
OLUNDERLINE/“underline”

This resource determines if the keyboard mnemonics on the controls should be displayed. Setting it to OL_UNDERLINE will cause the mnemonics to be displayed in the primitive children by drawing a line under the character in the font color. Setting it to OL_HIGHLIGHT will display the mnemonic character with the background and foreground colors reversed. When highlighting a character
that is displayed on a pixmap background, the mnemonic character will be
drawn in a solid color. The mnemonic accelerator will not be displayed if this
resource is set to \texttt{OL\_NONE}. The resource set to \texttt{OL\_INACTIVE} turns off the
mnemonic display as well as making the mnemonic key inactive.

\textbf{XtNtextBackground}

This resource reflects the default color used in the scrolling list and textEdit
widgets.

\textbf{XtNtextFontColor}

This resource reflects the default font color used in the scrolling list and textEdit
widgets.

\textbf{XtNthreeD}

This resource determines how the visuals are rendered. The default value,
\texttt{TRUE}, displays the visuals with a three dimensional look. Setting this resource
to \texttt{FALSE} will cause the visuals to have a two dimensional appearance.
Core Resources

These are the resources of the Core class, of which all widget classes are subclasses. They are described here to avoid repeating their descriptions for each widget.

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerators</td>
<td>XtCAccelerators</td>
<td>XtTranslations</td>
<td>G</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcolormap</td>
<td>XtCColormap</td>
<td>Colormap</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>Cardinal</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>SI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmanaged</td>
<td>XtCManaged</td>
<td>Boolean</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNtranslations</td>
<td>XtCTranslations</td>
<td>XtTranslations</td>
<td>G</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>SGI</td>
</tr>
</tbody>
</table>
**XtNancestorSensitive**

Range of Values:

- TRUE
- FALSE

This argument specifies whether the immediate parent of the widget will receive input events. To preserve data integrity, the application should use the `XtSetSensitive` routine if it wants to change the resource (see `XtNsensitive` below).

**XtNaccelerators**

This resource should not be set by an application.

**XtNbackground**

Range of Values:

(any valid current display Pixel value)

This resource specifies the background color for the widget.

Note that the OPEN LOOK workspace manager arranges for a “normal video” (or “reverse video”) effect on monochrome displays by setting the `XtNbackground` resource to white (black) and the `XtNforeground`, `XtNfontColor`, and `XtNborderColor` resources to black (white) where appropriate. The workspace manager also provides end user access to some of these resources on polychrome displays. However, any color set by the application when a widget is created or in a later call to `XtSetValues` will override the colors set by the user. Thus applications should consider this and avoid setting these resources directly, letting the user have control over them.

**XtNbackgroundPixmap**

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

This resource takes precedence over the `XtNbackground` resource.
See the note about the interaction of this resource with other color resources under the description of the `XtNbackground` resource above.

**XtNborderColor**

Range of Values:

\[(\text{any valid current display Pixel value})/(\text{any name from the rgb.txt file})\]

This resource specifies the color of the border.

See the note about the interaction of this resource with other color resources under the description of the `XtNbackground` resource above.

**XtNborderPixmap**

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

This resource takes precedence over the `XtNborderColor` resource.

See the note about the interaction of this resource with other color resources under the description of the `XtNbackground` resource above.

**XtNborderWidth**

Range of Values:

\[0 \leq \text{XtNborderWidth} \leq \min(\text{XtNwidth}, \text{XtNheight}) / 2\]

This resource sets the width of the border for a widget. Typically the border surrounds the widget’s window on all four sides, but in some widgets it is inside the widget, surrounding a view or control area contained in the widget. The width is specified in pixels, and a width of zero means no border will show.
**XtNcolormap**

Range of Values:

- StaticGray
- GrayScale
- StaticColor
- PseudoColor
- TrueColor
- DirectColor

This is the colormap to be used for the widget. While the range of values listed here is the entire list recognized, the actual valid values depend on the X server being used.

By default, a widget inherits its parent widget’s colormap. If the widget is a top level shell, the default is the default colormap for the screen (see the *XtNscreenu* resource).

**XtNdepth**

Range of Values:

- 0 or (any value supported by the current display)

Determines how many bits should be used for each pixel in the widget’s window. The value of this resource is used by the X Toolkit Intrinsics to set the depth of the widget’s window when the widget is created. A value of zero causes a normal widget to inherit the depth of its parent, or a pop-up widget to inherit the default depth of the screen.

**XtNdestroyCallback**

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

**XtNheight**

Range of Values:

- $0 \leq XtNheight$

This resource contains the height of the widget’s window in pixels, not including the border area. Programs may request a value at creation or through later
calls to XtSetValues, but the request may not succeed because of layout requirements of the parent widget.

The visual representations for some widgets have a fixed height for a given scale. For these widgets, the XtNheight resource gives the height of the space that contains the widget’s representation; the representation is centered vertically in this space unless otherwise specified.

**XtNmanaged**

Range of Values:

- **TRUE**
- **FALSE**

If this resource is set to **TRUE**, the widget is included in the geometry calculations of its parent. If **FALSE**, the widget is ignored by its parent and is not included in the geometry calculations.

**XtNmappedWhenManaged**

Range of Values:

- **TRUE**
- **FALSE**

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

**XtNscreen**

This resource should not be set by an application.

**XtNsensitive**

Range of Values:

- **TRUE**
- **FALSE**

This resource determines whether a widget will receive input events. If a widget is sensitive, the X Toolkit Intrinsic’s event manager will dispatch to the
Introduction to General Resources

widget all keyboard, mouse button, motion, window enter/leave, and focus events. Insensitive widgets do not receive these events. Also, insensitive widgets that appear on the screen are stippled with a 50% gray pattern to show that they are inactive, as a visual indication that the user can't interact with the widget. The 50% gray pattern is one that makes every other pixel of the widget the background color, in a checkerboard pattern.

An application should use the XtSetSensitive routine if it wants to change this resource. That way it ensures that if a parent widget has XtNsensitive set to FALSE, the XtNancestorSensitive flag of all its descendants will be appropriately set.

XtNtranslations

This resource should not be set by an application.

XtNwidth

Range of Values:

\[ 0 \leq \text{XtNwidth} \]

This resource contains the width of the widget's window in pixels, not including the border area. Programs may request a value at creation or through later calls to XtSetValues, but the request may not succeed because of layout requirements of the parent widget.

The visual representations for some widgets have a fixed width for a given scale. For these widgets, the XtNwidth resource gives the width of the space that contains the widget's representation; the representation is centered horizontally in this space unless otherwise specified.

XtNx

Range of Values:

\[ 0 \leq \text{XtNx} \]

This argument contains the x-coordinate of the widget's upper left hand corner (excluding the border) relative to its parent widget. Programs may request a value at creation or through later calls to XtSetValues, but the request may not succeed because of layout requirements of the parent widget.
XtNy

Range of Values:

\[ 0 \leq \text{XtNy} \]

This resource contains the y-coordinate of the widget’s upper left hand corner (excluding the border) relative to its parent widget. Programs may request a value at creation or through later calls to \texttt{XtSetValues}, but the request may not succeed because of layout requirements of the parent widget.

Flat Widget Layout

All of the flat containers have the same layout characteristics. The superclass of all flat widgets is a generic row/column manager. Though each column has its own width and each row has its own height, all columns can have the same width and all rows can have the same height, if desired.

\begin{itemize}
\item[NOTE] As a programming note, the efficiency in both processing steps and data requirements increases as the grid becomes more regular in shape. For example, a grid specifying that all rows must have the same height and all columns must have the same width is the most efficient configuration.
\end{itemize}

The flat row/column manager lays out each managed sub-object in row-major order or in column-major order depending on the attributes of the container, starting with the NorthWest corner of the container.

\begin{itemize}
\item[NOTE] Row-major order implies every column in the current row is filled before filling any columns in the next row. Column-major order implies every row in the current column is filled before filling any rows in the next column.
\end{itemize}

Sub-objects of flat containers are placed within the grid. If the sub-object’s width (or height) is less than the column’s width (or row’s height), the sub-object is positioned in accordance to the \texttt{XtNitemGravity} resource. The following table lists the layout resources of all flat containers:
**NOTE** See the resource tables for each flat container for a more accurate accounting of the default/allowable values for each layout resource.

### Common Flat Container Layout Resources

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNgravity</td>
<td>XtCGravity</td>
<td>int</td>
<td>CenterGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhPad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhSpace</td>
<td>XtCHSpace</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemGravity</td>
<td>XtCItemGravity</td>
<td>int</td>
<td>NorthWestGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMaxHeight</td>
<td>XtCItemMaxHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMaxWidth</td>
<td>XtCItemMaxWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinHeight</td>
<td>XtCItemMinHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinWidth</td>
<td>XtCItemMinWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutHeight</td>
<td>XtCLayoutHeight</td>
<td>OlDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OlDefine</td>
<td>OL_FIXEDROWS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutWidth</td>
<td>XtCLayoutWidth</td>
<td>OlDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameHeight</td>
<td>XtCSameHeight</td>
<td>OlDefine</td>
<td>OL_ALL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameWidth</td>
<td>XtCSameWidth</td>
<td>OlDefine</td>
<td>OL_COLUMNS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvSpace</td>
<td>XtCVSpace</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>
XtNgravity

Range of Values:

- EastGravity/"east"
- WestGravity/"west"
- CenterGravity/"center"
- NorthGravity/"north"
- NorthEastGravity/"northEast"
- NorthWestGravity/"northWest"
- SouthGravity/"south"
- SouthEastGravity/"southEast"
- SouthWestGravity/"southWest"

The gravity resource specifies the position of all sub-objects (that is, as a group) whenever a tight-fitting bounding box that surrounds the sub-objects has a width, or height, less than the container’s width or height, respectively. Essentially, this resource specifies how the sub-objects, as a group, float within its container.

XtNhPad/XtNvPad

Range of Values:

\[
0 \leq XtNhPad \\
0 \leq XtNvPad
\]

These resources specify the minimum spacing to leave round the edges of the container, left and right, and top and bottom, respectively.

XtNhSpace/XtNvSpace

Range of Values:

\[
0 \leq XtNhSpace \\
0 \leq XtNvSpace
\]

These resource specify the amount of space to leave between sub-objects horizontally and vertically, respectively. If the sub-objects are of different sizes in a row or column, the spacing applies to the widest or tallest dimension all sub-objects in the row or column.
**XtNItemGravity**

Range of Values:

- `EastGravity/"east"`
- `WestGravity/"west"`
- `CenterGravity/"center"`
- `NorthGravity/"north"
- `NorthEastGravity/"northEast"
- `NorthWestGravity/"northWest"
- `SouthGravity/"south"
- `SouthEastGravity/"southEast"
- `SouthWestGravity/"southWest"

This resource specifies how an item fits into its row or column whenever the item’s width or height is less than the column’s width or the row’s height. The values of `XtNSameWidth` and `XtNSameHeight` govern the column’s width and the row’s height.

**XtNItemMaxHeight/XtNItemMaxWidth**

Range of Values:

- `OL_IGNORE != XtNItemMaxHeight`
- `OL_IGNORE != XtNItemMaxWidth`

These resources specify the maximum allowable width and height (respectively) of all sub-objects. If either of these resources have a value of `OL_IGNORE`, the maximum size constraint is ignored.

**XtNItemMinHeight/XtNItemMinWidth**

Range of Values:

- `OL_IGNORE != XtNItemMinHeight`
- `OL_IGNORE != XtNItemMinWidth`

These resources specify the minimum allowable width and height (respectively) of all sub-objects. If either resource has a value of `OL_IGNORE`, it is ignored.
**XtNlayoutHeight/XtNlayoutWidth**

Range of Values:

- **OL_MINIMIZE**/"minimize"
- **OL_MAXIMIZE**/"maximize"
- **OL_IGNORE**/"ignore"

These resources specify the resize policy of flat containers whenever a sub-object is added, removed or altered. These resources have no affect when an external force applies a size change to the container, for example, if the application resizes a container. The explanation of the values are:

- **OL_MINIMIZE** The container will modify its width (or height) to be just large enough to tightly wrap around its sub-objects regardless of its current width (or height). Thus the container will grow and shrink depending on the size needs of its sub-objects.

- **OL_MAXIMIZE** The container will increase its width (or height) to be just large enough to tightly wrap around its sub-objects regardless of its current width (or height), but will not give up extra space. Thus the container will grow but never shrink depending on the size needs of its sub-objects.

- **OL_IGNORE** The container will honor its own width and height, for example, it will not grow or shrink in response to the addition, deletion or altering of its sub-objects.

**XtNlayoutType**

Range of Values:

- **OL_FIXEDCOLS**/"fixedcols"
- **OL_FIXEDROWS**/"fixedrows"

This resource controls the number of rows and columns used to layout the sub-objects.

- **OL_FIXEDCOLS** The layout should have a maximum number of columns equal to the value specified by the **XtNmeasure** resource, and there will be enough rows to hold all sub-objects. Sub-objects are placed in row-major order, for example, the columns of the current row are filled before filling any columns in the next row.
**OL_FIXEDROWS**  The layout should have a maximum number of rows equal to the value specified by the **XtNmeasure** resource, and there will be enough columns to hold all sub-objects. Sub-objects are placed in column-major order, for example, the rows of the current column are filled before filling any rows in the next column.

**XtNmeasure**

Range of Values:

\[ 0 < \text{XtNmeasure} \]

This resource gives the number of rows or columns that were requested from the **XtNlayoutType** resource.

**XtNsameHeight**

Range of Values:

\[
\begin{align*}
\text{OL_ALL/"all"} \\
\text{OL_ROWS/"rows"} \\
\text{OL.NONE/"none"}
\end{align*}
\]

This resource defines which sub-objects are forced to be the same height within the container:

- **OL.ALL**  All sub-objects are to be the same height.
- **OL.Rows**  All sub-objects appearing in the same row should be the same height.
- **OL.NONE**  The sub-objects are placed in fixed-height rows but the height of each item is left alone. The height of each row is the height of the tallest sub-object.
XtNsameWidth

Range of Values:

- \texttt{OL\_ALL/"all"}
- \texttt{OL\_COLUMNS/"columns"}
- \texttt{OL\_NONE/"none"}

This resource defines that sub-objects are forced to be the same width within the container:

- \texttt{OL\_ALL}
  All sub-objects are to be the same width.

- \texttt{OL\_COLUMNS}
  All sub-objects appearing in the same column should be the same width.

- \texttt{OL\_NONE}
  The sub-objects are placed in fixed-width columns but the width of each item is left alone. The width of each column is the width of the widest sub-object.

Manager Widget Resources

The following resources are available to the widgets that are a subclass of the Manager class.

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SG</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
</tbody>
</table>
**XtNconsumeEvent**

The resource overrides the OPEN LOOK handling of events. Whenever an event is processed by the standard OPEN LOOK translation table, the **XtNconsumeEvent** list is called for the widget in question allowing the application to consume the **XEvent**. To consume an event, the application should turn on (set to **TRUE**) the consumed field in the **call_data** argument when a given event is processed. If the **XEvent** is consumed, the widget doesn’t use it. If it is not consumed the widget uses it.

```c
typedef struct {
    Boolean consumed;
    XEvent xevent;
    Modifiers dont_care;
    OlVirtualName virtual_name;
    KeySym keysym;
    String buffer;
    Cardinal length;
    Cardinal item_index;
} OlVirtualEventRec, *OlVirtualEvent;
```

**XtNreferenceName**

Range of Values:

valid name of a widget

This resource specifies a position for inserting this widget in its managing ancestor’s traversal list. If the named widget exists in the managing ancestor’s traversal list, this widget will be inserted in front of it. Otherwise, this widget will be inserted at the end of the list.

If both the **XtNreferenceName** and **XtNreferenceWidget** resources are set, they must refer to the same widget. If not, a warning is issued and the widget referred to by name is used.
**XtNinputFocusColor**

Range of Values:

(valid Pixel value for the display)/(valid color name)

This resource specifies the color used to show that the widget has input focus. Normally, this color is derived from the value of **XtNinputFocusColor** resource and is dynamically maintained. This dynamic behavior is abandoned if the application explicitly sets this resource either at initialization or through a call to **XtSetValues**.

**XtNreferenceWidget**

This resource specifies a position for inserting this widget in its managing ancestor’s traversal list. If the reference widget is non-null and exists in the managing ancestor’s traversal list, this widget will be inserted in front of it. Otherwise, this widget will be inserted at the end of the list.

If both the **XtNreferenceName** and **XtNreferenceWidget** resources are set, they must refer to the same widget. If not, a warning is issued and the widget referred to by name is used.

**XtNtraversalOn**

This resource specifies whether this widget is accessible through keyboard traversal.

**XtNuserData**

This resource provides storage for application-specific data. It is not used or set by the widget.
 Primitive Widget Resources

The following resources are available to the Widgets that are a subclass of the Primitive class.

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>Dynamic</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNFont</td>
<td>XtCFont</td>
<td>XFontStruct*</td>
<td>(OPEN LOOK font)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black*</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td>OlFontList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNaccelerator**

This resource is used to define a single **KeyPress** event that will select a Primitive widget. The format of this string is identical to the translation manager syntax. Virtual Keys can be used in this translation.

**XtNacceleratorText**

This resource specifies a string that describes the Primitive’s accelerator. For example, a **Help** button may set the resource to the string “F1” to remind the users that function key 1 is the HELP button. This text will be displayed to the right of the Primitive’s label or image if the return from **OlQueryAcceleratorDisplay** is **OL_DISPLAY**.
This resource defaults to the `xtNaccelerator` string with “+”'s inserted between multiple key sequences.

**XtNconsumeEvent**

The resource overrides the OPEN LOOK handling of events. Whenever an event is processed by the standard OPEN LOOK translation table, the `XtNconsumeEvent` list is called for the widget in question allowing the application to consume the `XEvent`. To consume an event, the application should turn on (set to `TRUE`) the consumed field in the `call_data` argument when a given event is processed. If the `XEvent` is consumed, the widget doesn't use it. If it is not consumed the widget uses it.

```c
typedef struct {
    Boolean consumed;
    XEvent xevent;
    Modifiers dont_care;
    OIVirtual Name virtual_name;
    KeySym keysym;
    String buffer;
    Cardinal length;
    Cardinal item_index;
} OIVirtualEventRec, *OIVirtualEvent;
```

**XtNfont**

**XtNfontcolor**

**XtNfontGroup**

**XtNforeground**

**XtNinputFocusColor**

Range of Values:

(valid Pixel value for the display)/(valid color name)

This resource specifies the color used to show that the widget has input focus. Normally, this color is derived from the value of `XtNinputFocusColor` resource and is dynamically maintained. This dynamic behavior is abandoned if the application explicitly sets this resource either at initialization or through a call to `XtSetValues`. 
XtNmnemonic

This resource is a single character that is used as a mnemonic accelerator for keyboard operation. Typing this character modified with the **XtNmnemonic-Prefix** will be equivalent to clicking select on the Primitive widget. Note that in a Menu shell, the mnemonicPrefix modifier is not needed. The Primitive widget may visually display this character.

XtNreferenceName

Range of Values:

(the name of a widget already created in the traversal list)

This resource specifies a position for inserting this widget in its managing ancestor’s traversal list. If the the named widget exists in the managing ancestor’s traversal list, this widget will be inserted in front of it. Otherwise, this widget will be inserted at the end of the list.

If both the **XtNreferenceName** and **XtNreferenceWidget** resources are set, they must refer to the same widget. If not, a warning is issued and the widget referred to by name is used.

XtNreferenceWidget

This resource specifies a position for inserting this widget in its managing ancestor’s traversal list. If the reference widget is non-null and exists in the managing ancestor’s traversal list, this widget will be inserted in front of it. Otherwise, this widget will be inserted at the end of the list.

If both the **XtNreferenceName** and **XtNreferenceWidget** resources are set, they must refer to the same widget. If not, a warning is issued and the widget referred to by name is used.

XtNtraversalOn

This resource specifies whether this widget is accessible through keyboard traversal.
**XtNuserData**

This resource provides storage for application-specific data. It is not used or set by the widget.

**Shell Resources**

These are resources that are common to all widget classes that are subclasses of Shell. They are described here to avoid repeating their descriptions for each shell widget.

The behavior described for many of these resources assume the OPEN LOOK window manager for the X Window System is being used.

**Base Windows and Popup Windows**

All OPEN LOOK base windows are created using one of the routines: `olInitialize` or `xtCreateApplicationShell`. OPEN LOOK pop-up windows (the `PopupWindow`, `Menu`, and `Notice` widgets) are created using `xtCreatePopupShell`. An application may define other pop-up windows that can be created using `xtCreatePopupShell`.

The following table lists generic resources available to the base windows (that is, `TopLevelShell` and `ApplicationShell` widgets):

<table>
<thead>
<tr>
<th>Base WindowShell Resource Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>XtNiconic</td>
</tr>
<tr>
<td>XtNiconMask</td>
</tr>
<tr>
<td>XtNiconName</td>
</tr>
<tr>
<td>XtNiconPixmap</td>
</tr>
<tr>
<td>XtNiconWindow</td>
</tr>
<tr>
<td>XtNiconX</td>
</tr>
<tr>
<td>XtNiconY</td>
</tr>
</tbody>
</table>

The following resources are typical of base windows and generic pop-up windows, but not all are available for the pop-up windows defined in this toolkit.
See the list of resources for the **PopupWindow**, **Menu**, and **Notice** widgets to see which are available. The Access column in this table identifies the access for base windows only.

### Pop-up and Base Window Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Access (Base Window)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNallowShellResize</td>
<td>XtCAllowShellResize</td>
<td>Boolean</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgeometry</td>
<td>XtCGeometry</td>
<td>String</td>
<td>GI</td>
</tr>
<tr>
<td>XtNheightInc</td>
<td>XtCHeightInc</td>
<td>int</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinitialState</td>
<td>XtCInitialState</td>
<td>int</td>
<td>GI</td>
</tr>
<tr>
<td>XtNinput</td>
<td>XtCInput</td>
<td>Boolean</td>
<td>G</td>
</tr>
<tr>
<td>XtNmaxAspectX</td>
<td>XtCMaxAspectX</td>
<td>Position</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxAspectY</td>
<td>XtCMaxAspectY</td>
<td>Position</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxHeight</td>
<td>XtCMaxHeight</td>
<td>Dimension</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxWidth</td>
<td>XtCMaxWidth</td>
<td>Dimension</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminAspectX</td>
<td>XtCMinAspectX</td>
<td>Position</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminAspectY</td>
<td>XtCMinAspectY</td>
<td>Position</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminHeight</td>
<td>XtCMinHeight</td>
<td>Dimension</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminWidth</td>
<td>XtCMinWidth</td>
<td>Dimension</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNoverrideRedirect</td>
<td>XtCOVERRIDE_REDIRECT</td>
<td>Boolean</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNpopupCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsaveUnder</td>
<td>XtCSaveUnder</td>
<td>Boolean</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtitle</td>
<td>XtCTitle</td>
<td>String</td>
<td>SGI</td>
</tr>
<tr>
<td>XtN transient</td>
<td>XtCTransient</td>
<td>Boolean</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwaitForWm</td>
<td>XtCWaitForWm</td>
<td>Boolean</td>
<td>G</td>
</tr>
<tr>
<td>XtNwidthInc</td>
<td>XtCWidthInc</td>
<td>int</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwindowGroup</td>
<td>XtCWindowGroup</td>
<td>Window</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwmTimeout</td>
<td>XtCWmTimeout</td>
<td>int</td>
<td>G</td>
</tr>
</tbody>
</table>

The previous two tables listed generic resources available to most shells. The table below, however, lists resources necessary to support the OPEN LOOK look and feel. These resources are implemented in the **VendorShell** widget class; and therefore, apply only to shells which are subclasses of the **VendorShell** widget class (that is, **TransientShell**, **MenuShell**, **PopupWindowShell**, **NoticeShell**, **TopLevelShell** and **ApplicationShell**). Since some of these resources
do not apply to all shells (for example, \texttt{XtNresizeCorners} on menus), see the individual manual pages for a more accurate description of the applicable resources and their default values.

\textbf{OPEN LOOK look and feel Resource Set} \\
(VendorShell and its subclasses)

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNbusy</td>
<td>XtCNBusy</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfocusWidget</td>
<td>XtCFocusWidget</td>
<td>Widget</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmenuButtonType</td>
<td>XtCMenuButtonType</td>
<td>OlDefine</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNpushpin</td>
<td>XtCPushpin</td>
<td>OlDefine</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNresizeCorners</td>
<td>XtCResizeCorners</td>
<td>Boolean</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwindowHeader</td>
<td>XtCWindowHeader</td>
<td>Boolean</td>
<td>GI</td>
</tr>
<tr>
<td>XtNwmProtocol</td>
<td>XtCWMProtocol</td>
<td>XtRCallbackList</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNwmProtocolInterested</td>
<td>XtCWMProtocolInterested</td>
<td>int</td>
<td>I†</td>
</tr>
</tbody>
</table>

† Access value is conditional. Please see resource description for more information.

\textbf{NOTE} OlAddCallback must be used instead of XtAddCallback when adding callbacks to the XtNconsumeEvent and XtNwmProtocol callback lists. This restriction is due to a limitation in the Release 4 X Toolkit Intrinsic class extension mechanism with respect to callback lists. Likewise, OlRemoveCallback, OlHasCallbacks and OlCallCallbacks should be used instead of XtRemoveCallback, XtHasCallbacks and XtCallCallbacks, respectively.

All four of these OPEN LOOK routines has the same arguments and semantics of their Intrinsic's counterparts.
Resource Description

**XtNallowShellResize**

Range of Values:

TRUE  
FALSE

This resource controls whether the shell widget is allowed to resize itself in response to a geometry request from its child. If set to TRUE, it will attempt to resize itself as requested by the child. The attempt may be refused by the window manager, which will cause the shell widget to refuse the geometry management request of its child. Otherwise, it accepts the request. If the XtNallowShellResize request is set to FALSE, the shell widget will immediately refuse the geometry management request.

**XtNbusy**

Setting this resource to TRUE makes the application window associated with this shell busy. When a window becomes busy, the window manager grays the window header (if there is one). Setting the XtNbusy resource back to FALSE causes the window to return to its normal appearance and event processing.

NOTE

Neither the window manager or the toolkit grabs mouse or keyboard events when an application window becomes busy.

**XtNconsumeEvent**

The resource overrides the OPEN LOOK handling of events. Whenever an event is processed by the standard OPEN LOOK translation table, the XtNconsumeEvent list is called for the widget in question allowing the application to consume the XEvent. To consume an event, the application should turn on (set to TRUE) the consumed field in the call_data argument when a given event is processed.
NOTE

OlAddCallback must be used instead of XtAddCallback when adding call-
backs to the XtNconsumeEvent callback list.

XtNcreatePopupChildProc

This resource defines a pointer to a single function (not a callback list) that is
called during the process of popping up the shell widget. It is called after the
XtNpopupCallback callbacks are issued but before the shell widget is realized
and mapped. The function is passed a single argument, the ID of the shell
widget.

XtNfocusWidget

This resource controls which widget gets the input focus when a pop-up or base
window gains input focus. If this resource is non-NULL, focus is set it. If this
resource is NULL or the widget to which it refers is unwilling to accept input
focus, the pop-up or base window sets focus to either the default widget in the
window or the first widget willing to accept focus. As focus changes within the
shell, this resource is updated to reflect the last widget with focus.

A resource converter will translate widget names specified in a resource file to a
widget id for this resource.

XtNgeometry

Range of Values:

(any syntactically correct argument to the XParseGeometry function)

This resource can be used to specify the size and position of the shell widget
when it pops up.

XtNheightInc/XtNwidthInc

Range of Values:

\[ 0 \leq \text{XtNheightInc} \]
\[ 0 \leq \text{XtNwidthInc} \]
These resources define an arithmetic progression of sizes, from `$\text{XtNminHeight}$` and `$\text{XtNminWidth}$` to `$\text{XtNmaxHeight}$` and `$\text{XtNmaxWidth}$`, into which the shell widget prefers to be resized by the window manager.

**XtNiconic**

Range of Values:

- `TRUE`
- `FALSE`

This resource provides an equivalent method of setting the `$\text{XtNinitialState}$` resource to `IconicState`.

**XtNiconMask**

This resource defines an image that specifies which pixels of the `$\text{XtNiconPixmap}$` resource should be used for the base window’s icon. This image must be a single plane pixmap.

**XtNiconName**

This resource defines a name that the window manager will display in the shell widget’s icon. If the `$\text{XtNtitle}$` resource is not defined or is `NULL`, this resource is used instead. If this resource is `NULL`, the name of the application is used in its place.

**XtNiconPixmap**

This resource defines the image to be used as the base window’s icon. It must be a single plane pixmap.

**XtNiconWindow**

Range of Values:

- (ID of any existing window)

This resource defines the ID of a window that the window manager should use for the base window’s icon, in place of `$\text{XtNiconPixmap}$`. The `$\text{XtNiconWindow}$` takes precedence over the `$\text{XtNiconPixmap}$` resource.
XtNiconX/XtNiconY

Range of Values:

-1 ≤ XtNiconX
-1 ≤ XtNiconY

These resources define the location where the base window’s icon should appear. If the value of one of these resources is -1, the window manager automatically picks a value, according to its icon placement requirements.

XtNinitialState

Range of Values:

NormalState/“1”
IconicState/“3”

This resource defines how the base window (and associated pop-up windows) appears when the application starts up.

NormalState When set to this value, the application starts up with its base window open.

IconicState When set to this value, the application starts up with its base window closed into an icon.

NOTE Other values are defined by the X Window System for this resource, but the OPEN LOOK window manager recognizes only the iconic and normal states.

XtNinput

This resource controls the type of input focus behavior of the application. It should not be set by an application.
XtNmenuButton

This Boolean resource determines if the menu button decoration should be drawn in the upper left corner of the shell window's header. The default, TRUE, indicates that it should be drawn. If the XtNpushpin resource is not OL_NONE, this resource is ignored.

XtNmenuType

Range of Values:

- OL_MENU_FULL/"full"
- OL_MENU_LIMITED/"limited"
- OL_MENU_CANCEL/"cancel"
- OL_NONE/"none"

This resource provides the application access to the type of window menu that the OPEN LOOK window manager creates. The default value is OL_MENU_FULL for a base shell. This full menu contains the following entries: Close, Full Size, Properties, Back, Refresh, and Quit. Setting this resource to OL_MENU_LIMITED results in a window menu with the following buttons: Dismiss (a MenuButton), Back, Refresh, and Owner?. PopupWindow and Help shells set this resource to OL_MENU_LIMITED. The menu type OL_MENU_CANCEL provides the same menu as the OL_MENU_LIMITED with the exception that the Dismiss button is replaced with a Cancel button. When the XtNmenuType resource is OL_NONE, the window manager does not create a menu or a menu mark.

XtNmaxAspectX/XtNmaxAspectY
XtNminAspectX/XtNminAspectY

Range of Values:

-1 = XtNmaxAspectX, 1 ≤ XtNmaxAspectX
-1 = XtNmaxAspectY, 1 ≤ XtNmaxAspectY
-1 = XtNminAspectX, 1 ≤ XtNminAspectX
-1 = XtNminAspectY, 1 ≤ XtNminAspectY

\[
\frac{XtNminAspectX}{XtNminAspectY} \leq \frac{XtNmaxAspectX}{XtNmaxAspectY}
\]
These resources define the range of aspect ratios allowed for the size of the shell widget’s window. Assuming the width and height of the window are given by \textit{width} and \textit{height}, the following relation shows how the window size is constrained:

$$\frac{\text{XtNminAspectX}}{\text{XtNminAspectY}} \leq \frac{\text{width}}{\text{height}} \leq \frac{\text{XtNmaxAspectX}}{\text{XtNmaxAspectY}}$$

If the end user tries to resize the window to a narrower or wider aspect ratio than allowed by these resources, the window manager adjusts the window to the closest allowed aspect ratio. If possible, it will do this by increasing the width or height to compensate. The \textit{XtNmaxHeight} and \textit{XtNmaxWidth} resources may force the window manager to reduce the width or height instead.

If the values of these resources are -1, the window manager does not constrain the size of the window to any aspect ratio.

\textbf{NOTE}\hspace{1em} An application should either set all values to -1 (the default) or should set all to a positive value. An application should never set a value of zero to any of these resources.

\textbf{XtNmaxHeight/XtNmaxWidth/XtNminHeight/XtNminWidth}\hspace{1em}

Range of Values:

$$\text{XtNminHeight} \leq \text{XtNmaxHeight}$$
$$\text{XtNminWidth} \leq \text{XtNmaxWidth}$$
(or \texttt{OL\_IGNORE} for any of these resources)

These resources define the range allowed for the size of the shell widget’s window. If the end user tries to resize the window smaller or larger than these values allow, the window manager adjusts the width and/or height to compensate.

The default value of \texttt{OL\_IGNORE} keeps the window manager from constraining the window’s size.
**XtNoverrideRedirect**

Range of Values:

```
TRUE
FALSE
```

This resource controls whether the shell widget’s window is managed by the window manager. Since this OPEN LOOK toolkit is designed to have a certain pop-up window behavior, this resource should not be set by an application for the shell widgets defined in this toolkit (Menu, Notice, and PopupWindow).

**XtNpopdownCallback**

This resource defines callbacks automatically issued right after the shell widget’s window has been unmapped (that is, popped down.)

**XtNpopupCallback**

This resource defines callbacks automatically issued right before the shell widget is realized and mapped (that is, popped up.)

**XtNpushpin**

Range of Values:

```
OL_NONE/"none"
OL_OUT/"out"
OL_IN/"in"
```

This resource controls whether the pushpin is included in the window’s decorations. The default for the base shell type is OL_NONE, indicating that the pushpin is not included in the window’s decorations. Setting this resource to OL_OUT adds the pushpin to the window’s decorations, and sets its state to be unpinned. OL_OUT is the default value for this resource in PopupWindow shells. Setting this resource to OL_IN adds the pushpin to the window’s decorations, and sets its state to be pinned. OL_IN is the default value for this resource in Help shells. Applications can query the state of the pushpin by getting the value of this resource, since it is updated when the pushpin’s state changes.
If the shell does not have an OPEN LOOK header (\texttt{XtNwindowHeader set to FALSE}), then \texttt{XtNpushpin} is always \texttt{OL\_NONE}, and attempts to change the value are ignored.

### \texttt{XtNresizeCorners}

This Boolean resource determines if the OPEN LOOK resize corners should be part of the window decorations. The default for the base shell is \texttt{TRUE}, that resize corners are present. PopupWindow also defaults to having resize corners, but Notice, Help and Menu shells do not have resize corners.

### \texttt{XtNsavUnder}

Range of Values:

\begin{verbatim}
    TRUE
    FALSE
\end{verbatim}

This resource directs the shell widget to instruct the server to attempt to save the contents of windows obscured by the shell when it is mapped, and to restore the contents when the shell widget is unmapped.

### \texttt{XtNtitle}

This resource gives the title to include in the header of the base or pop-up window. Widgets of other classes besides \texttt{Shell} may have a resource with the same name.

### \texttt{XtNtransient}

Range of Values:

\begin{verbatim}
    TRUE
    FALSE
\end{verbatim}

This resource controls whether the shell widget’s window is “transient” and is to be unmapped when the associated base window is iconified (see \texttt{XtNwindow-Group}). Since this OPEN LOOK toolkit is designed to have a certain pop-up window behavior, this resource should not be set by an application for the shell widgets defined in this toolkit (\texttt{Menu, Notice, and PopupWindow}).
**XtNuserData**

This resource provides storage for application-specific data. It is not used or set by the widget. Its default value is NULL.

**XtNwaitForWm**

This resource should not be set by an application.

**XtNwindowGroup**

Range of Values:

(ID of any existing window)

This resource identifies the base window associated with this shell widget’s window. When the end user closes the base window, all its associated windows are unmapped (pop-up windows or other shell widget windows with XtNtransient set to TRUE) or closed (base windows with XtNtransient set to FALSE).

**XtNwindowHeader**

This Boolean resource determines if the window manager should provide a header for the window. The header is the area of the window that contains the pushpin, title, and window mark. The default for the base shell is TRUE, that the window does have a header. The PopupWindow and Help shells also have default headers, but the Notice shell would default to FALSE. This resource can only be set at initialization.

**XtNwmProtocol**

Range of Values:

- OL_WM_TAKE_FOCUS
- OL_WM_SAVE_YOURSELF
- OL_WM_DELETE_WINDOW

This resource controls the action that is taken whenever a shell widget (which is a subclass of a VendorShell widget class) receives WM_PROTOCOL messages. If no callback list is specified, the shell performs its default action(s). If a callback list is specified, it is invoked and no default action(s) is taken. The application
can, however, simulate the default action(s) at its convenience by calling \texttt{OlWMProtocolAction} with the \texttt{action} parameter set to \texttt{OL_DEFAULTACTION}. (See \texttt{OlWMProtocolAction(3W)} for more information on this routine.)

When the application's callback procedure is invoked, the \texttt{call_data} field is a pointer to the following structure:

```c
typedef struct {
    int msgtype;    /* type of WM msg */
    XEvent * xevent;
} OlWMProtocolVerify;
```

The field \texttt{msgtype} is an integer constant indicating the type of protocol message which invoked the callback and has a range of values of:

- \texttt{OL_WM_TAKE_FOCUS}
- \texttt{OL_WM_SAVE_YOURSELF}
- \texttt{OL_WM_DELETE_WINDOW}

\texttt{OlAddCallback} must be used instead of \texttt{XtAddCallback} when adding callbacks to the \texttt{XtNwmProtocol} callback list.

\section*{XtNwmProtocolInterested}

This specifies the types of protocol messages that interest the application. By default, it is both \texttt{OL_WM_DELETE_WINDOW} and \texttt{OL_WM_TAKE_FOCUS}. Furthermore, these two types are always turned on and cannot be turned off. Thus, setting \texttt{XtNwmProtocolInterested} to \texttt{OL_WM_SAVE_YOURSELF} will get all three types.

\section*{XtNwmTimeout}

This resource should not be set by an application.
Manual Pages: Convenience Routines

Introduction to the Convenience Routines

General Routines
Dynamic Settings Utilities
Text Buffer Utilities
  ■ Regular Expression Utilities
  ■ Database Routines
  ■ Gauge Widget Routines
  ■ Routines to Support Input Method
  ■ TextEdit Routines
  ■ TextField Routines
  ■ Obsolete Routines

Convenience Routines
Introduction to the Convenience Routines

This section describes the format of the man pages for convenience routines. Each routine starts on a new page and has the layout described below.

- The top of the first page for each major feature gives the feature name on both the top left and top right of the page, followed by a "3W" in parentheses. (3W is the numbering convention assigned to all OPEN LOOK programming manual pages.) Many of the routines are grouped by similar function (for example, Dynamic Setting or a Cursor Utility). The group name appears directly under the page header. When the routine has several variations a general or meaningful form was used for the page header.

- Some manual pages have several versions of a routine each with its own syntax and function. See the Synopsis or Description sub-sections for explanation.

- A synopsis is given that shows, briefly, how to invoke the function.

- Most routines have a description.

- The description is sometimes broken up into minor features, each starting with a separate heading.

- Some routine pages have SEE ALSO sections which direct you towards other commands that deal with a similar function.

- As a final item, some of the routines have an associated return value.

General Routines

Error
Flattened Widget Utilities
Input Focus
Mnemonic/AcceleratorDisplay
OlGetApplicationValues
OlInitialize
OlRegisterHelp
OlWMProtocolAction
OlWidgetToClassName
OlWidgetClassToClassName
PackedWidget
Introduction to the Convenience Routines

Pixel Conversion
Widget Activation/Association

Dynamic Settings Utilities

GetOlMoveCursor
GetOlDuplicateCursor
GetOlBusyCursor
GetOlPanCursor
GetOlQuestionCursor
GetOlTargetCursor
GetOlStandardCursor
OlGet50PercentGrey
OlGet75PercentGrey
OlGrabDragPointer
OlUngrabDragPointer
OlDragAndDrop
OlRegisterDynamicCallback
OlUnregisterDynamicCallback
OlCallDynamicCallbacks

Text Buffer Utilities

Buffer Macros
AllocateBuffer
AllocateTextBuffer
BackwardScanTextBuffer
CopyBuffer
CopyTextBufferBlock
EndCurrentTextBufferWord
ForwardScanTextBuffer
FreeBuffer
FreeTextBuffer
GetTextBufferBlock
GetTextBufferBuffer
GetTextBufferChar
GetTextBufferLine
GetTextBufferLocation
GrowBuffer
IncrementTextBufferLocation
InsertIntoBuffer
LastTextBufferLocation
LastTextBufferPosition
LineOfPosition
LocationOfPosition
NextLocation
NextTextBufferWord
PositionOfLine
PositionOfLocation
PreviousLocation
PreviousTextBufferWord
ReadFileIntoTextBuffer
ReadFileIntoBuffer
ReadStringIntoBuffer
ReadStringIntoTextBuffer
ReplaceCharInTextBuffer
ReplaceBlockInTextBuffer
RegisterTextBufferScanFunctions
RegisterTextBufferUpdate
RegisterTextBufferWordDefinition
SaveTextBuffer
StartCurrentTextBufferWord
stropen
strgetc
strclose
TextBuffer Macros
UnregisterTextBufferUpdate

Regular Expression Utilities

strexp
strrexp
streexp
Introduction to the Convenience Routines

Database Routines

OlGetApplication Values
OlClassSearchIEDB
OlClassSearchTextDB
OlLookupInputEvent
OlWidgetSearchIEDB
OlWidgetSearchTextDB
OlUpdateDisplay

Gauge Widget Routines

OlSetGaugeValue

Routines to Support Input Method

OlImValues
OlResetIm
OlCloseIm
OlCreateIm
OlDisplayOfIm
OlDestroyIm
OlGetImValues
OlGetImValues
OlImValues
OlImOfIm
OlLookUpImString
OlLocaleOfIm
OlOpenIm
OlSetImFocus
OlSetImValues
OlUnSetImFocus

TextEdit Routines

OlTextEditClearBuffer
OlTextEditCopyBuffer
OlTextEditCopySelection
OlTextEditGetCursorPosition
OlTextEditGetLastPosition
OlTextEditInsert

OPEN LOOK GUI Programmer’s Guide
Introduction to the Convenience Routines

OlTextEditPaste
OlTextEditReadSubString
OlTextEditRedraw
OlTextEditSetCursorPosition
OlTextEditBuffer
OlTextEditUpdate

TextField Routines

OlTextFieldCopyString
OlTextFieldGetString

Obsolete Routines

Please see the "Manual Pages: Obsolete Routines" chapter in this guide for descriptions of routines that are no longer supported.

Abbrevstack Widget
Buttonstack Widget
LookupOlInputEvent
OlDetermineMouseAction
OlGetApplicationResources
OlReplayBtnEvent
Text Widget
Virtual Key/Button
NAME
AllocateBuffer

SYNOPSIS
#include <buffutil.h>
...
extern Buffer * AllocateBuffer(element_size, initial_size)
int element_size;
int initial_size;

DESCRIPTION
The AllocateBuffer function allocates a Buffer for elements of the given element_size. The used member of the Buffer is set to zero and the size member is set to the value of initial_size. If initial_size is zero the pointer p is set to NULL, otherwise the amount of space required (initial_size * element_size) is allocated and the pointer p is set to point to this space. The function returns the pointer to the allocated Buffer.

SEE ALSO
FreeBuffer(3W)
Text Buffer Utilities

NAME
AllocateTextBuffer

SYNOPSIS
#include <textbuff.h>
...
extern TextBuffer * AllocateTextBuffer(filename, f, d)
char * filename;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION
The AllocateTextBuffer function is used to allocate a new TextBuffer. After it allocates the structure itself, initializes the members of the structure, allocating storage, setting initial values, etc. The routine also registers the update function provided by the caller. This function normally need not be called by an application developer since the ReadFileIntoTextBuffer and ReadStringIntoTextBuffer functions call this routine before starting their operation. The routine returns a pointer to the allocated TextBuffer.

The FreeTextBuffer function should be used to deallocate the storage allocated by this routine.

SEE ALSO
FreeTextBuffer(3W)
ReadFileIntoTextBuffer(3W)
ReadStringIntoTextBuffer(3W)
NAME

BackwardScanTextBuffer

SYNOPSIS

```
#include <textbuff.h>
...
extern ScanResult BackwardScanTextBuffer(text, exp, location)
    TextBuffer * text;
    char * exp;
    TextLocation * location;
```

DESCRIPTION

The BackwardScanTextBuffer function is used to scan, towards the beginning of
the buffer, for a given expression in the TextBuffer starting at location. A Scan-
Result is returned which indicates

- **SCAN_NOTFOUND**: The scan wrapped without finding a match.
- **SCAN_WRAPPED**: A match was found at a location after the start location.
- **SCAN_FOUND**: A match was found at a location before the start location.
- **SCAN_INVALID**: Either the location or the expression was invalid.

SEE ALSO

ForwardScanTextBuffer(3W)
NAME
Buffer Macros:  BufferFilled, BufferLeft, BufferEmpty

SYNOPSIS
#include <buffutil.h>
BufferFilled(buffer)
BufferLeft(buffer)
BufferEmpty(buffer)

DESCRIPTION
These macros are provided for use with the Buffer Utilities.
BufferFilled returns a flag indicating whether buffer is filled.
BufferLeft returns the number of unused elements in buffer.
BufferEmpty returns a flag indicating whether buffer is empty.
CopyBuffer (3W)

Text Buffer Utilities

NAME
CopyBuffer

SYNOPSIS
#include <buffutil.h>
...
extern Buffer * CopyBuffer(buffer)
Buffer * buffer;

DESCRIPTION
The CopyBuffer function is used to allocate a new Buffer with the same attributes as the given buffer and to copy the data associated with the given buffer into the new Buffer. A pointer to the newly allocated and initialized Buffer is returned. It is the responsibility of the caller to free this storage when appropriate.

SEE ALSO
AllocateBuffer(3W)
FreeBuffer(3W)
InsertIntoBuffer(3W)
**CopyTextBufferBlock (3W)**

**Text Buffer Utilities**

**NAME**

CopyTextBufferBlock

**SYNOPSIS**

```c
#include <textbuff.h>

extern int CopyTextBufferBlock(text, buffer, start_position, end_position)

TextBuffer * text;
char * buffer;
TextPosition start_position;
TextPosition end_position;
```

**DESCRIPTION**

The `CopyTextBufferBlock` function is used to retrieve a text block from the `text` TextBuffer. The block is defined as the characters between `start_position` and `end_position` inclusive. It returns the number of bytes copied; if the parameters are invalid the return value is zero (0).

**SEE ALSO**

GetTextBufferLocation(3W)
GetTextBufferChar(3W)
GetTextBufferLine(3W)

**NOTES**

The storage for the copy is allocated by the caller. It is the responsibility of the caller to ensure that enough storage is allocated to copy `end_position - start_position + 1` bytes.
EndCurrentTextBufferWord (3W)

Text Buffer Utilities

NAME
EndCurrentTextBufferWord

SYNOPSIS
#include <textbuff.h>
...
extern TextLocation EndCurrentTextBufferWord(textBuffer, current)
TextBuffer * textBuffer;
TextLocation current;

DESCRIPTION
The EndCurrentTextBufferWord function is used to locate the end of a word in
the TextBuffer relative to a given current location. The function returns the loca-
tion of the end of the current word. Note: this return value will equal the given
current value if the current location is already at the end of a word.

SEE ALSO
PreviousTextBufferWord(3W)
NextTextBufferWord(3W)
Error(3W)

NAME
Error: OIError, OIWarning, OIVaDisplayErrorMsg, OIVaDisplayWarningMsg, OISetErrorHandler, OISetWarningHandler, OISetVaDisplayErrorMsgHandler, OISetVaDisplayWarningMsgHandler - error and warning message handling

SYNOPSIS
#include <Intrinsic.h>
#include <OpenLook.h>

void OIError(msg)
String msg;     /* error message string */

void OIWarning(msg)
String msg;     /* warning message string */

void OIVaDisplayErrorMsg(dpy, name, type, class, default_msg, ...)
Display * dpy;   /* Display pointer or NULL */
String name;    /* message name */
String type;    /* message type */
String class;   /* message class */
String default_msg; /* message format string */
...            /* variable arguments for */
...            /* the message format string */

void OIVaDisplayWarningMsg(dpy, name, type, class, default_msg, ...)
Display * dpy;   /* Display pointer or NULL */
String name;    /* message name */
String type;    /* message type */
String class;   /* message class */
String default_msg; /* message format string */
...            /* variable arguments for */
...            /* the message format string */

OIErrorHandler
OIErrorHandler (handler)
    OIErrorHandler handler; /* handler or NULL */

OIWarningHandler
OIWarningHandler (handler)
    OIWarningHandler handler; /* handler or NULL */

OIVaDisplayErrorMsgHandler
OIVaDisplayErrorMsgHandler (handler)
    OIVaDisplayErrorMsgHandler handler; /* handler or NULL */

OIVaDisplayWarningMsgHandler
OIVaDisplayWarningMsgHandler (handler)
    OIVaDisplayWarningMsgHandler handler; /* handler or NULL */
typedef void (*OlErrorHandler)(
    String msg /* error message string */
);

typedef void (*OlWarningHandler)(
    String msg /* warning message string */
);

typedef void (*OlVaDisplayErrorMsgHandler)(
    Display * dpy, /* Display pointer or NULL */
    String name, /* message name */
    String type, /* message type */
    String class, /* message class */
    String default_msg, /* message format string */
    va_list ap /* variable arguments for */
                     /* the message format string */
);

typedef void (*OlVaDisplayWarningMsgHandler)(
    Display * dpy, /* Display pointer or NULL */
    String name, /* message name */
    String type, /* message type */
    String class, /* message class */
    String default_msg, /* message format string */
    va_list ap /* variable arguments for */
                     /* the message format string */
);

DESCRIPTION
OlError
OlError writes a simple string to stderr and then exits.

OlWarning
OlWarning writes a simple string to stderr and then returns.

OlVaDisplayErrorMsg
OlVaDisplayErrorMsg writes an error to stderr and exits. The error message is
looked up in the error database by calling XtAppGetErrorDatabaseText using
the name, type, and class arguments. If no message is found in the error database,
the default_msg string is used. The application context is determined by calling
XtDisplayToApplicationContext with the supplied Display pointer. If the
display pointer is NULL, the display created at application startup is used to
determine the application context.

An application programmer can choose to customize the text in an error message
prefix. The following shows what the prefix is in an example message and which
is the standard substituted string in the prefix:
"application class" OPEN LOOK Toolkit Error: m e s s a g e
To customize the prefix, put the following line in the client’s ***_msgs file, where *** is the name of the client:

*message.prefix: text %s text %s text

where the first %s will be the application name and the second will be the content of the message.

If the prefix is not customized, the current prefix will be supplied as the default.

To silence any toolkit error, put a corresponding "NULL" message in the client app-defaults/***_msgs file:

For example, to silence:

*invalidResource.setValues: SetValues: widget

Put the line

*invalidResource.setValues:

in the ***_msgs file.

Note that OlVaDisplayErrorMsg will still cause the client application to exit.

OlVaDisplayWarningMsg

OlVaDisplayWarningMsg has the same semantics as OlVaDisplayErrorMsg except it returns instead of exiting.

An application programmer can choose to customize the text in a warning message prefix. The following shows what the prefix is in an example message and the standard substituted strings in the prefix:

"application class" OPEN LOOK Toolkit Warning: m e s s a g e
To customize the prefix, put the following line in the client’s ***_msgs file:

*message.prefix: text %s text %s text

where the first %s will be the application name and the second will be the type of message.

If the prefix is not customized, the current prefix will be supplied as the default.

To silence any toolkit warning, put a corresponding "NULL" message in the client app-defaults/***_msgs file:

For example, to silence:

*invalidResource.setValues: SetValues: widget

Put the line

*invalidResource.setValues:

in the ***_msgs file.
Others

OlSetErrorHandler, OlSetWarningHandler, OlSetVaDisplayErrorMsgHandler, and OlSetVaDisplayWarningMsgHandler allow an application to override the various warning and error handlers. These routines return a pointer to the previous handler. If NULL is supplied to any of these routines, the default handler will be used.

Since the error routines normally exit the program, application-supplied error handlers should do the same since continuation of an application will result in undefined behavior.

NOTES

Most programs should not use OlError and OlWarning since they don’t allow for customization and internationalization.

Since OpenLook.h does not include stdarg.h (or varargs.h), an application using OlSetVaDisplayErrorMsgHandler or OlSetVaDisplayWarningMsgHandler should include one of these two headers before including OpenLook.h to insure the correct function prototype will be used for the application’s error/warning handler.

SEE ALSO

OlGetMessage(3W)
NAME
Flattened Widget Utilities: OlFlatCallAcceptFocus, OlFlatGetFocusItem, OlFlatGetItemIndex, OlFlatGetItemGeometry, OlFlatGetValues, OlFlatSetValues, OlVaFlatGetValues OlVaFlatSetValues – queries and manipulates flattened widget attributes

SYNOPSIS
There are several convenience routines for querying or manipulating flattened widget attributes. All of these routines issue a warning if the widget id is not a subclass of a flattened widget.

Boolean OlFlatCallAcceptFocus(widget, index, time)
  Widget widget; /* Flattened widget id */
  Cardinal index; /* Item’s index */
  Time time; /* time of request */

Cardinal OlFlatGetFocusItem(widget)
  Widget widget; /* Flattened widget id */

Cardinal OlFlatGetItemIndex(widget, x, y)
  Widget widget; /* Flattened widget id */
  Position x; /* x location within widget */
  Position y; /* y location within widget */

void Ol.FlatGetItemGeometry(widget, index, x_ret, y_ret, w_ret, h_ret)
  Widget widget; /* Flattened widget id */
  Cardinal index; /* Item’s index */
  Position * x_ret; /* returned x coordinate */
  Position * y_ret; /* returned y coordinate */
  Dimension * w Ret; /* returned width */
  Dimension * h Ret; /* returned height */

Getting and Setting Flattened Widget Item State
void OlFlatGetValues(widget, index, args, num_args)
  Widget widget; /* Flattened widget id */
  Cardinal index; /* Item’s index */
  ArgList args; /* attributes to query */
  Cardinal num_args; /* number of args */

void OlFlatSetValues(widget, index, args, num_args)
  Widget widget; /* Flattened widget id */
  Cardinal index; /* Item’s index */
  ArgList args; /* attributes to set */
  Cardinal num_args; /* number of args */

void OlVaFlatGetValues(widget, index, ...)
  Widget widget; /* Flattened widget id */
DESCRIPTION

OlFlatCallAcceptFocus
If specified item is capable of accepting input focus, focus is assigned to the item
and function returns TRUE; otherwise, FALSE is returned.

OlFlatGetFocusItem
Returns the item within the flattened widget which has focus. OL_NO_ITEM is
returned if no item within the widget has focus.

OlFlatGetItemIndex
Returns the item that contains the given x and y coordinates. OL_NO_ITEM is
returned otherwise if no item contains the coordinate pair.

OlFlatGetItemGeometry
Returns the location and width/height of an item with respect to its flattened
widget container. If the supplied item index is invalid, a warning is issued and
the return values are set to zero.

OlFlatGetValues
Queries attributes of an item. This routine is very similar to XtGetValues.
Applications can query any attribute of an item even if the attribute was not
specified in the XtNitemFields resource of the flat widget container. Also see
OlVaFlatGetValues.

OlFlatSetValues
Sets attributes of an item. This routine is very similar to XtSetValues. Applica­
tions can set values of item attributes only even if the attribute name was
specified in the XtNitemFields resource of the flat widget container or if the
item’s attribute is always maintained (i.e., implicitly) by the flat widget container
regardless of the XtNitemFields entries. For example, since the FlatEx­
clusives Widget always maintains the value of an item’s XtNset attribute even
if XtNset was not in the XtNitemFields resource (see FlatExclusives(3W)).
Therefore, an application can set the value of XtNset even though XtNset was
not specified explicitly in the XtNitemFields resource for the widget. XtNfont,
on the otherhand, is not implicitly maintained by the FlatExclusives Widget, so an
application must specify XtNfont in the XtNitemFields resource if that applica­
tion wants to change the font value via OlFlatSetValues. Also see
OlVaFlatSetValues.

OlVaFlatGetValues
Variable argument interface to OlFlatGetValues. The variable length list of
resource name/value pairs is terminated by a NULL resource name.
OLVaFlatSetValues
Variable argument interface to OlFlatSetValues. The variable length list of
resource name/value pairs is terminated by a NULL resource name.
Text Buffer Utilities

NAME
ForwardScanTextBuffer

SYNOPSIS
#include <textbuff.h>
...
extern ScanResult ForwardScanTextBuffer(text, exp, location)
TextBuffer * text;
TextLocation * location;
char * exp;

DESCRIPTION
The ForwardScanTextBuffer function is used to scan, towards the end of the
buffer, for a given expression in the TextBuffer starting at location. A ScanResult
is returned which indicates

SCAN_NOTFOUND The scan wrapped without finding a match.
SCAN_WRAPPED A match was found at a location before the start location.
SCAN_FOUND A match was found at a location after the start location.
SCAN_INVALID Either the location or the expression was invalid.

SEE ALSO
BackwardScanTextBuffer(3W)
Text Buffer Utilities

NAME
FreeBuffer

SYNOPSIS
#include <buffutil.h>
...  
extern void FreeBuffer(buffer)
Buffer * buffer;

DESCRIPTION
The FreeBuffer procedure is used to deallocate (free) storage associated with the given buffer pointer.

SEE ALSO
AllocateBuffer(3W)
FreeTextBuffer (3W)

Text Buffer Utilities

NAME
FreeTextBuffer

SYNOPSIS
#include <textbuff.h>
...
extern void FreeTextBuffer(text, f, d)
TextBuffer * text;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION
The FreeTextBuffer procedure is used to deallocate storage associated with a
given TextBuffer. Note: the storage is not actually freed if the TextBuffer is still
associated with other update function/data pairs.

SEE ALSO
AllocateTextBuffer(3W)
RegisterTextBufferUpdate(3W)
NAME
GetOLBusyCursor

SYNOPSIS
#include <OlCursors.h>
...
extern Cursor GetOLBusyCursor(screen)
Screen * screen;

DESCRIPTION
The GetOLBusyCursor function is used to retrieve the Cursor id for screen that complies with the OPEN LOOK specification of the Busy cursor.

SEE ALSO
GetOLMoveCursor(3W)
GetOLDuplicateCursor(3W)
GetOLPanCursor(3W)
GetOLQuestionCursor(3W)
GetOLTargetCursor(3W)
GetOLStandardCursor(3W)

RETURN VALUE
The Cursor id is returned.
GetOLDuplicateCursor

NAME
GetOLDuplicateCursor

SYNOPSIS
#include <OlCursors.h>

extern Cursor GetOLDuplicateCursor(screen)
Screen * screen;

DESCRIPTION
The GetOLDuplicateCursor function is used to retrieve the Cursor id for screen that complies with the OPEN LOOK specification of the Duplicate cursor.

SEE ALSO
GetOLMoveCursor(3W)
GetOLBusyCursor(3W)
GetOLPanCursor(3W)
GetOLQuestionCursor(3W)
GetOLTTargetCursor(3W)
GetOLStandardCursor(3W)

RETURN VALUE
The Cursor id is returned.
NAME
GetOlMoveCursor

SYNOPSIS
#include <OlCursors.h>
...
extern Cursor GetOlMoveCursor(screen)
Screen * screen;

DESCRIPTION
The GetOlMoveCursor function is used to retrieve the Cursor id for screen that complies with the OPEN LOOK specification of the Move cursor.

SEE ALSO
GetOlDuplicateCursor(3W)
GetOlBusyCursor(3W)
GetOlPanCursor(3W)
GetOlQuestionCursor(3W)
GetOlTargetCursor(3W)
GetOlStandardCursor(3W)

RETURN VALUE
The Cursor id is returned.
NAME
GetOlPanCursor

SYNOPSIS
#include <olCursors.h>
...
extern Cursor GetOlPanCursor(screen)
Screen * screen;

DESCRIPTION
The GetOlPanCursor function is used to retrieve the Cursor id for screen that com-
plies with the OPEN LOOK specification of the Pan cursor.

SEE ALSO
GetOlMoveCursor(3W)
GetOlDuplicateCursor(3W)
GetOlBusyCursor(3W)
GetOlQuestionCursor(3W)
GetOlTargetCursor(3W)
GetOlStandardCursor(3W)

RETURN VALUE
The Cursor id is returned.
NAME
GetOlQuestionCursor

SYNOPSIS
#include <OlCursors.h>

... extern Cursor GetOlQuestionCursor(screen)
Screen * screen;

DESCRIPTION
The GetOlQuestionCursor function is used to retrieve the Cursor id for screen that complies with the OPEN LOOK specification of the Question cursor.

SEE ALSO
GetOlMoveCursor(3W)
GetOlDuplicateCursor(3W)
GetOlBusyCursor(3W)
GetOlPanCursor(3W)
GetOlTargetCursor(3W)
GetOlStandardCursor(3W)

RETURN VALUE
The Cursor id is returned.
GetOlStandardCursor(3W)

NAME
    GetOlStandardCursor

SYNOPSIS
    #include <OlCursors.h>
    ...
    extern Cursor GetOlStandardCursor(screen)
    Screen * screen;

DESCRIPTION
    The GetOlStandardCursor function is used to retrieve the Cursor id for screen that complies with the OPEN LOOK specification of the Standard cursor.

SEE ALSO
    GetOlMoveCursor(3W)
    GetOlDuplicateCursor(3W)
    GetOlBusyCursor(3W)
    GetOlPanCursor(3W)
    GetOlQuestionCursor(3W)
    GetOlTargetCursor(3W)

RETURN VALUE
    The Cursor id is returned.
NAME
GetOlTargetCursor

SYNOPSIS
#include <OlCursors.h>
...
extern Cursor GetOlTargetCursor(screen)
Screen * screen;

DESCRIPTION
The GetOlTargetCursor function is used to retrieve the Cursor id for screen that complies with the OPEN LOOK specification of the Target cursor.

SEE ALSO
GetOlMoveCursor(3W)
GetOlDuplicateCursor(3W)
GetOlBusyCursor(3W)
GetOlPanCursor(3W)
GetOlQuestionCursor(3W)
GetOlStandardCursor(3W)

RETURN VALUE
The Cursor id is returned.
GetTextBufferBlock (3W)

Text Buffer Utilities

NAME
GetTextBufferBlock

SYNOPSIS
#include <textbuff.h>
...
extern char * GetTextBufferBlock(text, start_location, end_location)
TextBuffer * text;
TextLocation start_location;
TextLocation end_location;

DESCRIPTION
The GetTextBufferBlock function is used to retrieve a text block from the text
TextBuffer. The block is defined as the characters between start_location and
end_location inclusive. It returns a pointer to a string containing the copy. If
the parameters are invalid NULL is returned.

SEE ALSO
GetTextBufferLocation(3W)
GetTextBufferChar(3W)
GetTextBufferLine(3W)

NOTES
The storage for the copy is allocated by this routine. It is the responsibility of the
caller to free this storage when it becomes dispensable.
NAME
GetTextBufferBuffer

SYNOPSIS
#include <textbuff.h>
...
extern Buffer * GetTextBufferBuffer(text, line)
TextBuffer * text;
TextLine line;

DESCRIPTION
The GetTextBufferBuffer function is used to retrieve a pointer to the Buffer stored in TextBuffer text for line. This pointer is volatile; subsequent calls to any TextBuffer routine may make it invalid. If a more permanent copy of this Buffer is required the Buffer Utility CopyBuffer can be used to create a private copy of it.

SEE ALSO
GetTextBufferBlock(3W)
GetTextBufferLocation(3W)
NAME
    GetTextBufferChar

SYNOPSIS
    #include <textbuff.h>
    ...
    int GetTextBufferChar(text, location)
    TextBuffer * text;
    TextLocation location;

DESCRIPTION
    The GetTextBufferChar function is used to retrieve a character stored in the text
    TextBuffer at location. It returns either the character itself or EOF if location is out-
    side the range of valid locations within the TextBuffer.

SEE ALSO
    GetTextBufferLocation(3W)
    GetTextBufferBlock(3W)
    GetTextBufferLine(3W)
GetTextBufferLine (3W)

Text Buffer Utilities

NAME
GetTextBufferLine

SYNOPSIS
#include <textbuff.h>
...
extern char * GetTextBufferLine(text, lineindex)
TextBuffer * text;
TextLine lineindex;

DESCRIPTION
The GetTextBufferLine function is used to retrieve the contents of line from the text TextBuffer. It returns a pointer to a string containing the copy of the contents of the line or NULL if the line is outside the range of valid lines in text.

SEE ALSO
GetTextBufferLocation(3W)
GetTextBufferChar(3W)
GetTextBufferBlock(3W)

NOTES
The storage for the copy is allocated by this routine. It is the responsibility of the caller to free this storage when it becomes dispensable.
NAME
   GetTextBufferLocation

SYNOPSIS
   #include <textbuff.h>

   ... extern char * GetTextBufferLocation(text, line_number, location)
   TextBuffer * text;
   TextLine line_number;
   TextLocation * location;

DESCRIPTION
   The GetTextBufferLocation function is used to retrieve the contents of the
given line within the TextBuffer. It returns a pointer to the character string. If
the line number is invalid a NULL pointer is returned. If a non-NULL TextLoca-
tion pointer is supplied in the argument list the contents of this structure are
modified to reflect the values corresponding to the given line.

SEE ALSO
   GetTextBufferBlock(3W)
NAME
GrowBuffer

SYNOPSIS
#include <buffutil.h>
... 
extern void GrowBuffer(buffer, increment)
Buffer * buffer;
int increment;

DESCRIPTION
The GrowBuffer procedure is used to expand (or compress) a given buffer size by increment elements. If the increment is negative the operation results in a reduction in the size of the Buffer.

SEE ALSO
AllocateBuffer(3W)
Text Buffer Utilities

NAME
IncrementTextBufferLocation

SYNOPSIS
#include <textbuff.h>

extern TextLocation IncrementTextBufferLocation(text, location, line, offset)

TextBuffer * text;
TextLocation location;
TextLine line;
TextPosition offset;

DESCRIPTION
The IncrementTextBufferLocation function is used to increment a location by either line lines and/or offset characters. It returns the new location. Note: if line or offset are negative the function performs a decrement operation. If the starting location or the resulting location is invalid the starting location is returned without modification; otherwise the new location is returned.

SEE ALSO
NextLocation(3W)
PreviousLocation(3W)
NAME
Input Focus: OlCallAcceptFocus, OlCanAcceptFocus, OlGetCurrentFocusWidget, OlHasFocus, OlMoveFocus, OlSetInputFocus—manipulates input focus for widgets and gadgets.

SYNOPSIS
Six convenience routines are available for manipulating input focus. Each of these routines work with widgets or gadgets.

Boolean OlCallAcceptFocus(w, time)
    Widget w;
    Time time;

Boolean OlCanAcceptFocus(w, time)
    Widget w;
    Time time;

void OlSetInputFocus(w, revert_to, time)
    Widget w;
    int revert_to;
    Time time;

Widget OlGetCurrentFocusWidget(w)
    Widget w;

Boolean OlHasFocus(w)
    Widget w;

Widget OlMoveFocus(w, direction, time)
    Widget w;
    OlVirtualName direction;
    Time time;

DESCRIPTION
OlCallAcceptFocus
If widget w currently is capable of accepting input focus, focus is assigned to w and the function returns TRUE; otherwise, FALSE is returned. Time specifies the X server time of the event that initiated this accept focus request.

OlCanAcceptFocus
If the widget can accept focus, TRUE is returned; otherwise, FALSE is returned. Acceptance of focus is determined by the following:

The widget is not being destroyed.
The widget is managed
The widget is mapped when managed (if it’s not a gadget)
The widget is realized, or for a gadget, the gadget’s parent must be realized.

The widget and its ancestors are sensitive.

A query window attributes is successful and the widget’s window is viewable (i.e., the window and all its ancestor windows are mapped).

Otherwise, FALSE is returned.

**OILSetInputFocus**

OILSetInputFocus sets focus to a widget. Applications should use this routine instead of OXSetInputFocus since this routine checks the current focus model and before setting focus.

**OILGetCurrentFocusWidget**

OILGetCurrentFocusWidget returns the id of the widget which currently has focus in the window group of the specified widget. If no widget in the window group has focus, NULL is returned.

**OILHasFocus**

This function is simply a convenience function that calls OILGetCurrentFocusWidget and compares the result of that call to the supplied widget id.

**OILMoveFocus**

This function moves the input focus from one widget to another and returns the id of the new focus widget. The routine OICallAcceptFocus is used to move the input focus. If the function is unable to move input focus, NULL is returned.

When moving input focus between widgets contained within an exclusives or non-exclusives widget, valid values for direction are:

Note: For the OL_MULTI directions below, the value of m is the value of the application resource XtNmultiObjectCount.

**OL_IMMEDIATE**

Set focus to the next widget that will accept it, starting with w.

**OL_MOVERIGHT**

Set focus to the widget in the next column (and same row) that will accept it, starting with the first column after w’s column. If w is located in the extreme right column, focus is set to the widget in the extreme left column of the same row.

**OL_MOVELEFT**

Set focus to the widget in the previous column (and same row) that will accept it, starting with the first column before w’s column. If w is located on the extreme left column, focus is set to the widget in the extreme right column of the same row. (The widget columns are searched in reverse order.)

**OL_MOVEUP**

Set focus to the widget in the next row (and same column) that will accept it, starting with the first row after w’s row. If w is located in the top row, focus is set to the widget in the bottom row of the same column. (The widget rows are searched in reverse order.)
OL_MOVEDOWN
Set focus to the widget in the previous row (and same column) that will accept it, starting with the first row before \( w \)'s row. If \( w \) is located in the bottom row, focus is set to the widget in the top row of the same column.

OL_MULTIRIGHT
Set focus to the widget in the next column (and same row) that will accept it, starting with the first column \( m \) columns after \( w \)'s column. If \( m \) is greater than the number of objects between \( w \) and the extreme right column, focus is set to the widget in the extreme left column of the same row.

OL_MULTILEFT
Set focus to the widget in the next column (and same row) that will accept it, starting with the first column \( m \) columns before \( w \)'s column. If \( m \) is greater than the number of objects between \( w \) and the extreme left column, focus is set to the widget in the extreme right column of the same row. (The widget columns are searched in reverse order.)

OL_MULTIUP
Set focus to the widget in the next row (and same column) that will accept it, starting with the first row \( m \) rows after \( w \)'s row. If \( m \) is greater than the number of objects between \( w \) and the extreme bottom row, focus is set to the widget in the extreme top row of the same column. (The widget rows are searched in reverse order.)

OL_MULTIDOWN
Set focus to the widget in the next row (and same column) that will accept it, starting with the first row \( m \) rows before \( w \)'s row. If \( m \) is greater than the number of objects between \( w \) and the extreme bottom row, focus is set to the widget in the extreme bottom row of the same column. When moving between widgets in a base window or popup window, valid values for direction are:

OL_IMMEDIATE
Set focus to the next object that will accept it, starting with \( w \).

OL_NEXTFIELD
Set focus to the next object that will accept it, starting with the first object after \( w \).

OL_PREVFIELD
Set focus to the next object that will accept it, starting with the first object before \( w \). (The list is searched in reverse order.)

OL_MOVERIGHT
Behaves like OL_NEXTFIELD direction.

OL_MOVELEFT
Behaves like OL_PREVFIELD direction.

OL_MOVEUP
Behaves like OL_PREVFIELD direction.
OL_MOVEDOWN
Behaves like OL_NEXTFIELD direction.

OL_MULTIRIGHT
Set focus to the next object that will accept it, starting with the first object $m$ objects after $w$.

OL_MULTILEFT
Set focus to the next object that will accept it, starting with the first object $m$ objects before $w$. (The list is searched in reverse order.)

OL_MULTIUP
Behaves like OL_MULTILEFT direction.

OL_MULTIDOWN
Behaves like OL_MULTIRIGHT direction.
NAME
InsertIntoBuffer

SYNOPSIS
#include <buffutil.h>

extern int InsertIntoBuffer(target, source, offset)
Buffer * target;
Buffer * source;
int offset;

DESCRIPTION
The InsertIntoBuffer function is used to insert the elements stored in the source buffer into the target buffer before the element stored at offset. If the offset is invalid or if the source buffer is empty the function returns zero; otherwise it returns one after completing the insertion.

SEE ALSO
ReadStringIntoBuffer(3W)
ReadFileIntoBuffer(3W)
BufferMacros(3W)
Text Buffer Utilities

NAME
LastTextBufferLocation

SYNOPSIS
#include <textbuff.h>
...
extern TextLocation LastTextBufferLocation(text)
TextBuffer * text;

DESCRIPTION
The LastTextBufferLocation function returns the last valid TextLocation in the
TextBuffer associated with text.

SEE ALSO
LastTextBufferPosition(3W)
FirstTextBufferLocation(3W)
Text Buffer Utilities

NAME

LastTextBufferPosition

SYNOPSIS

#include <textbuff.h>

...extern TextPosition LastTextBufferPosition(text)
TextBuffer * text;

DESCRIPTION

The LastTextBufferPosition function returns the last valid TextPosition in the
TextBuffer associated with text.

SEE ALSO

LastTextBufferLocation(3W)
FirstTextBufferLocation(3W)
NAME
LineOfPosition

SYNOPSIS
#include <textbuff.h>
...
extern int LineOfPosition(text, position)
TextBuffer * text;
TextPosition position;

DESCRIPTION
The LineOfPosition function is used to translate a position in the text TextBuffer to a line index. It returns the translated line index or EOF if the position is invalid.

SEE ALSO
LineOfPosition(3W)
PositionOfLocation(3W)
LocationOfPosition(3W)
NAME
LocationOfPosition

SYNOPSIS

```
#include <textbuff.h>
...
extern TextLocation LocationOfPosition(text, position)
TextBuffer * text;
TextPosition position;
```

DESCRIPTION

The LocationOfPosition function is used to translate a position in the text
TextBuffer to a TextLocation. It returns the translated TextLocation. If the position is invalid the
Buffer pointer buffer is set to NULL and the line and offset members are set the last valid
location in the TextBuffer; otherwise buffer is set to a non-NULL (though useless) value.

SEE ALSO
LineOfPosition(3W)
PositionOfLocation(3W)
LocationOfPosition(3W)
NAME

Mnemonic/AcceleratorDisplay: OlQueryMnemonicDisplay
OlQueryAcceleratorDisplay – Controls mnemonic and accelerator displays

DESCRIPTION

Query Mnemonic Display for Screen
OlDefine OlQueryMnemonicDisplay(w)
Widget w;

Range of Values:

OL_UNDERLINE
OL_HIGHLIGHT
OL_NONE
OL_INACTIVE

This function queries how the keyboard mnemonics on the controls should be displayed. The default value is OL_UNDERLINE; the mnemonics should be displayed in the Primitive controls by drawing a line under the character in the font color. OL_HIGHLIGHT will display the mnemonic character with the background and foreground colors reversed. When highlighting a character that is displayed on a pixmap background, the mnemonic character will be drawn in a solid color. The mnemonic accelerator will not be displayed and will not activate the control if this function returns OL_INACTIVE.

Query Accelerator Display for Screen
OlDefine OlQueryAcceleratorDisplay(w)
Widget w;

Range of Values for return:

OL_DISPLAY
OL_NONE
OL_INACTIVE

When this function returns OL_DISPLAY, the keyboard accelerators should be displayed. When this function returns OL_NONE, the keyboard accelerators should not be displayed. The return value OL_INACTIVE means the keyboard accelerators should not be displayed and the controls should ignore the accelerator action.
Text Buffer Utilities

NAME
   NextLocation

SYNOPSIS
   #include <textbuff.h>
   ...
   extern TextLocation NextLocation(textBuffer, current)
   TextBuffer * textBuffer;
   TextLocation current;

DESCRIPTION
   The NextLocation function returns the TextLocation which follows the given current location in a TextBuffer. If the current location points to the end of the TextBuffer this function wraps.

SEE ALSO
   PreviousLocation(3W)
NextTextBufferWord (3W)

Text Buffer Utilities

NAME
NextTextBufferWord

SYNOPSIS
#include <textbuff.h>
...
extern TextLocation NextTextBufferWord(textBuffer, current)
TextBuffer * textBuffer;
TextLocation current;

DESCRIPTION
The NextTextBufferWord function is used to locate the beginning of the next word from a given current location in a TextBuffer. If the current location is within the last word in the TextBuffer the function wraps to the beginning of the TextBuffer.

SEE ALSO
PreviousTextBufferWord(3W)
StartCurrentTextBufferWord(3W)
Dynamic Settings

NAME
OlCallDynamicCallbacks

SYNOPSIS
#include <OpenLook.h>
...
extern void OlCallDynamicCallbacks();

DESCRIPTION
The OlCallDynamicCallbacks procedure is used to trigger the calling of the functions registered on the dynamic callback list. This procedure is called automatically whenever the RESOURCE_MANAGER property of the RootWindow is updated. It may also be called to force a synchronization of the dynamic settings, though applications rarely need to do this explicitly.

SEE ALSO
OlRegisterDynamicCallback(3W)
OlUnregisterDynamicCallback(3W)
NAME
OlClassSearchIEDB – register a given database on a specific widget class

SYNOPSIS
#include <Xol/OpenLook.h>

... 
void
OlClassSearchIEDB(wc, db)

WidgetClass wc;
OlVirtualEventTable db;

DESCRIPTION
The OlClassSearchIEDB procedure is used to register a given database on a specific widget class. The db value was returned from a call to OlCreateInputEventDB(3W).

Once a database is registered with a given widget class, the OlLookupInputEvent(3W) procedure (if db flag is OL_DEFAULT_IE or db) will include this database in the search stack if the given widget id is a subclass of this widget class.

NOTES
The registering order determines the searching order when doing a lookup.

SEE ALSO
OlClassSearchTextDB(3W)
OlCreateInputEventDB(3W)
OlLookupInputEvent(3W)
OlWidgetSearchIEDB(3W)
OlWidgetSearchTextDB(3W)

EXAMPLE
/* To create a client application database */
#include <Xol/OpenLook.h>
#include <Xol/Stub.h>
/* start with a big value to avoid */
/* the "virtual_name" collision */
#define OL_MY_BASE 1000
#define OL_MY_DRAWLINEBTN OL_MY_BASE+0
#define OL_MY_DRAWARCBTN OL_MY_BASE+1
#define OL_MY_REDISPLAYKEY OL_MY_BASE+2
#define OL_MY_SAVEPARTKEY OL_MY_BASE+3

#define XtNmyDrawLineBtn "myDrawLineBtn"
#define XtNmyDrawArcBtn "myDrawArcBtn"
#define XtNmyRedisplayKey "myRedisplayKey"
#define XtNmySavePartKey "mySavePartKey"

static OlKeyOrBtnRec OlMyBtnlnfo[] = {
   /*name default_value virtual_name */
   { XtNmyDrawLineBtn, "c<Button1>", OL_MY_DRAWLINEBTN },
   { XtNmyDrawArcBtn, "s<myDrawLineBtn>", OL_MY_DRAWARCBTN },

3/91


```c
static OIKeyOrBtnRec OIMyKeyInfo[] = {
    /*name*/ default_value virtual_name */
    { XtNmyRedisplayKey, "c<FS>" }, OIMyDB, OL_MY_REDISPLAYKEY },
    { XtNmySavePartKey, "c<FS>" }, OIMyDB, OL_MY_SAVEPARTKEY },
};

static OIVirtualEventTable OIMyDB;

OIMyDB = OICreateInputEventDB(
    w,
    OIMyKeyInfo, XtNumber(OIMyKeyInfo),
    OIMyBtnInfo, XtNumber(OIMyBtnInfo)
);
/* assume: all stub widgets are interested in OIMyDB */
OIClassSearchIEDB(stubWidgetClass, OIMyDB);
/* once this step is done, all stub widget instances */
/* will receive the OIMyDB commands after a call to */
/* OILookupInputEvent(), or in the XtNconsumeEvent */
/* callback’s OIVirtualEvent structure supplied with */
/* the call_data field. */
```
OlClassSearchTextDB (3W) (Database Utilities) OlClassSearchTextDB (3W)

NAME
OlClassSearchTextDB – register the OPEN LOOK TEXT database on a specific widget class

SYNOPSIS
#include <Xol/OpenLook.h>
...
void OlClassSearchTextDB (wc)
    WidgetClass wc;

DESCRIPTION
The OlClassSearchTextDB procedure is used to register the OPEN LOOK TEXT database on a specific widget class.

Once the OPEN LOOK TEXT database is registered with a given widget class, the OlLookupInputEvent(3W) procedure (if db_flag is OL_DEFAULT_IE or OL_TEXT_IE) will include this database in the search stack if the given widget id is a subclass of this widget class.

NOTES
The registering order determines the searching order when doing a lookup.

SEE ALSO
OlClassSearchIEDB(3W)
OlCreateInputEventDB(3W)
OlLookupInputEvent(3W)
OlWidgetSearchIEDB(3W)
OlWidgetSearchTextDB(3W)

EXAMPLE
...
#include <Xol/OpenLook.h>
#include <Xol/Stub.h>
...

    /* assume: all stub widgets are interested in the */
    /* OPEN LOOK TEXT database */
    OlClassSearchTextDB(stubWidgetClass);
    /* once this step is done, all stub widget instances */
    /* will receive OPEN LOOK TEXT commands after a */
    /* call to OlLookupInputEvent(), or in the */
    /* XtNconsumeEvent callback’s OlVirtualEvent */
    /* structure supplied with the call_data field. */
    ...

...
NAME
OlCloseDatabase – responsible for closing a database opened with OlOpenDatabase

SYNOPSIS
void OlCloseDatabase(name, database)
    String name;
    XrmDatabase database;

DESCRIPTION
This function is responsible for closing a database opened by OlOpenDatabase. Both functions expect the database to be in XrmDatabase format.

The function will resolve the pathname of the database according to the current locale.

SEE ALSO
OlOpenDatabase(3W)
OlGetMessage(3W)
NAME
OlCloseIm – responsible for closing the Input Method

SYNOPSIS
void OlCloseIm(Im)
    OlIm *im;

DESCRIPTION
This function is responsible for closing the Input Method (IM). Its functionality is implementation dependent. Some common steps that are performed by all implementations include:

• Destroying all ICs (Input Contexts) associated with the IM and de-allocating storage used by these ICs.

• De-allocating storage used by the OlIm structure.

SEE ALSO
OlOpenIm(3W)
OlGetImValues(3W)
OlDisplayOfIm(3W)
OlLocaleOfIm(3W)
OlImOfIc(3W)
NAME
OlCreateIC - Create an Input Context to register a client's text insertion window with an IM

SYNOPSIS
typedef struct _OlICWindowAttr {
    Pixel background;
    Pixel foreground;
    Colormap colormap;
    Colormap std_colormap;
    Pixmap backPixmap;
    OLFontList fontlist;
    int spacing;
    Cursor cursor;
    OLIMCallback callback [NUM_IM_CALLBACKS];
} OlICWindowAttr;

typedef struct _OlIC {
    Window cl_win;
    XRectangle cl_area;
    Window focus_win;
    XRectangle s_area;
    Window s_attr;
    XRectangle pre_area;
    Window pre_attr;
    OLIMStyle style;
    XPoint spot;
    struct OlIC *nextic;
    void *ictype;
} OlIC;

OlIC * OlCreateIC(im, icvalues)
    OLIM    *im;
    OlICValues icvalues;

DESCRIPTION
This function creates an Input Context for the client's text insertion window with Input Method (IM). im is a pointer to the OLIM structure returned by the function OlOpenIM. icvalues points to a variable list of attribute name value pairs to be associated with the IC (Input Context). (See OlICValues for the description of valid OlICValues attributes.) One attribute, XtNclientWindow must always be specified at creation time.

The function creates and initializes the OlIC structure, containing the context information about a particular text area. This includes information about client and focus windows, pre-edit and status areas, pre-edit and status attributes, a pointer to the OLIM structure associated with the IC, and a pointer to the next IC associated with the same IM. The function returns a pointer to this structure.
If for any reason the function fails to create an Input Context, it should return a null value. The ictype field is a hook for attaching implementation dependent data structures.

SEE ALSO
OlDestroyIc(3W)
OlGetIcValues(3W)
OlIcValues(3W)
OlImOfIc(3W)
OlSetIcFocus(3W)
OlSetIcValues(3W)
OlUnSetIcFocus(3W)
OlResetIc(3W)
Database Utilities

NAME

OlCreateInputEventDB – create a client specific Key and/or Button database

SYNOPSIS

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

OlVirtualEventTable
OlCreateInputEventDB(w, key_info, num_key_info, btn_info, num_btn_info)

Widget w;
OlKeyOrBtnInfo key_info;
int num_key_info;
OlKeyBtnInfo btn_info;
int num_btn_info;
```

DESCRIPTION

The OlCreateInputEventDB function is used to create a client specific Key and/or Button database. This function returns a database pointer if the call to this function is successful otherwise a NULL pointer is returned. Mapping for a new virtual command can be composed from the mappings of a previously defined virtual command.

The returned value from this function is an opaque pointer (OlVirtualEventTable). A client application should use this pointer when registering and/or looking up this database.

```c
typedef struct _OlVirtualEventInfo *OlVirtualEventTable;
```

The `key_info` and `btn_info` parameters are a pointer to an OlKeyOrBtnRec structure.

```c
typedef struct {
    String name;
    String default_value; /* "," separate string */
    OlVirtualName virtual_name;
} OlKeyOrBtnRec, *OlKeyOrBtnInfo;
```

Caveat

A client application can create a Key only database by having the NULL `btn_info`. The same applies to a Button only database.

Each virtual command can have two different bindings because the OPEN LOOK toolkit allows the alternate key or button sequence.

The OPEN LOOK toolkit already has a set of predefined OPEN LOOK virtual names. It is important that the `virtual_name` value of a client application database starts with a big value to avoid the `virtual_name` collision.

SEE ALSO

OlClassSearchIEDB(3W)
OlClassSearchTextDB(3W)
OlLookupInputEvent(3W)
OlWidgetSearchIEDB(3W)
OlWidgetSearchTextDB(3W)
EXAMPLE

/* To create a client application database */
...
#include <Xol/OpenLook.h>
...
/* start with a big value to avoid */
/* the "virtual_name" collision */
#define OL_MY_BASE 1000
#define OL_MY_DrawLineBTN OL_MY_BASE+0
#define OL_MY_DrawArcBTN OL_MY_BASE+1
#define OL_MY_RedisplayKey OL_MY_BASE+2
#define OL_MY_SavePartKey OL_MY_BASE+3

#define XtNmyDrawLineBtn "myDrawLineBtn"
#define XtNmyDrawArcBtn "myDrawArcBtn"
#define XtNmyRedisplayKey "myRedisplayKey"
#define XtNmySavePartKey "mySavePartKey"

static OlKeyOrBtnRec OlMyBtnInfo[] = {
    /* name */ default_value virtual_name */
    { XtNmyDrawLineBtn, "c<Button1>", OL_MY_DRAWLINEBTN },
    { XtNmyDrawArcBtn, "s<myDrawLineBtn>", OL_MY_DRAWARCBTN },
};

static OlKeyOrBtnRec OlMyKeyInfo[] = {
    /* name */ default_value virtual_name */
    { XtNmyRedisplayKey, "c<F5>", OL_MY_REDISPLAYKEY },
    { XtNmySavePartKey, "c<F5>", OL_MY_SAVEPARTKEY },
};

static OlVirtualEventTable OlMyDB;
...

OlMyDB = OlCreateInputEventDB(
    w,
    OlMyKeyInfo, XtNumber(OlMyKeyInfo),
    OlMyBtnInfo, XtNumber(OlMyBtnInfo)
);
...

NAME

OICtToEuc – returns a null-terminated string

SYNOPSIS

```c
int OICtToEuc(ctstr, eucstr, euc_len, font1)
    XctString    ctstr;
    XctString    eucstr;
    int          euc_len;
    OlFontList   *font1;
```

DESCRIPTION

Given a null-terminated compound text encoded string in the ctstr argument, this function returns a semantically equivalent null-terminated string in the eucstr argument that conforms to the EUC syntax. Equivalence implies that the characters in both strings are the same and are stored in the same logical order. The code set scheme syntax is the only difference in the two strings. The fontl argument provides this function with information about supplemental EUC code set support in the current locale. If during the conversion OICtToEuc(3W) encounters a character from an unsupported code set, it generates an error by returning -1. Optional directional information provided in the compound text encoded string is ignored as EUC does not support directional rendering.

The memory for storing the EUC encoded string must be pre-allocated prior to the function call. Doing this, rather than making the function allocate the memory every time it is called, results in better performance, because pre-allocated memory could be easily re-used in case of repeated calls to OICtToEuc(3W). The euc_len argument specifies the length of the eucstr and is required to prevent buffer overflow. If the number of bytes indicated by euc_len is too small to store the converted string, -1 is returned.

This function returns the length, in bytes (not counting the terminating null character), of the EUC encoded string on success, or -1 on failure.

SEE ALSO

O1EucToCt(3W)
OlCvtFontGroupToFontStructList (3W) OlCvtFontGroupToFontStructList (3W)

NAME

OlCvtFontGroupToFontStructList – converts a fontGroup string into a
font_list

SYNOPSIS

#include <OpenLook.h>
#include <OlStrings.h>
#include <Converters.h>

OlFontList * font_list;

static XtResource resources[] =
{
  { XtNfontGroup, XtCFontGroup, OlROlFontList, sizeof(OlFontList *),
    &font_list, XtrString, (XtPointer)NULL
  },
}

static Boolean
CvtFontGroupToFontStructList (display, args, num_args, from, to, XtCacheRef)
Display display;
XrmValuePtr args = NULL;
Cardinal num_args = 0;
char * fontGroup;
Cardinal num_args=0;
XrmValue from;
XrmValue to;
XtCacheRef *cache_ref_return = NULL;

from.addr = (XtPointer) fontGroup;
from.size = strlen(fontGroup);
to.addr = (XtPointer) &font_list;
to.size = sizeof(OlFontList *);
XtCallConverter(dpy,
  OlCvtFontGroupToFontStruct,
  args,
  num_args,
  &from,
  &to,
  cache_ref_return);

DESCRIPTION

The OlCvtFontGroupToFontStruct OPEN LOOK converter converts a font­
Group string into corresponding OlFontList for use with internationalized text
drawing.
The converter can be invoked in two ways:

1. By specifying the source type of a `fontGroup` as a string and destination type as `OlFontList` (`OlO1FontList`) in the `XtResource` array. The converter, in this case, is automatically invoked and, if successful, a pointer to the `OlFontList` structure is returned in `font_list`.

2. The converter can be directly invoked by the application to obtain a pointer to `OlFontList` by calling the `Xt Intrinsics` function `XtCallConverter` as shown above.

The `font_list` returned in either case can then be passed to text drawing utilities `OlDrawString`, `OlDrawImageString`, `OlTextWidth`, `OlgetNextStrSegment`.

**SEE ALSO**
- `XtCallConverter`
- `OlDrawString(3W)`
- `OlDrawImageString(3W)`
- `OlTextWidth(3W)`
- `OlgetNextStrSegment(3W)`

**NOTES**

The storage for the `OlFontList` is allocated by the converter in both cases. Invocation by specifying different source and destination types in the `XtResource` array will cause Intrinsics to keep track of reference count and manage the cache. Hence, the user does not need to free storage explicitly. However, in case of direct invocation via `XtCallConverter()`, it is the caller's responsibility to free the storage for `OlFontList`. 
OIDragAndDrop(3W)

Dynamic Settings

NAME
OIDragAndDrop

SYNOPSIS
#include <QpenLook.h>
...
extern void OIDragAndDrop(w, window, xPosition, yPosition)
Widget w;
Window * window;
Position * xPosition;
Position * yPosition;

DESCRIPTION
The OIDragAndDrop function is used to monitor a direct manipulation operation; returning, when the operation is completed, the drop_window and the x and y coordinates corresponding to the location of the drop. These return values will reflect the highest (in the stacking order) window located under the pointer at the time of the button release.

SEE ALSO
OIDragAndDrop(3W)

EXAMPLE
The following code provides a sample of the use of the facilities:

static void ButtonConsumeCB (w, client_data, call_data)
{
    Widget w;
    XtPointer client_data;
    XtPointer call_data;

    Window drop_window;
    Position x;
    Position y;
    OIVirtualEvent ve;

    ve = (OIVirtualEvent) call_data

    switch (ve->virtual_name) {
    case OL_SELECT:
        OIDragAndDrop(&drop_window, &x, &y);
        DropOn(drop_window, x, y, ....);
        OIUngrabDragPointer();
        break;
    }
    ....
}
OIDestroylc (3W)

NAME
OIDestroylc – destroys a specified IC

SYNOPSIS

```c
void OIDestroylc(ic)
    OLIC *ic;
```

DESCRIPTION
This function will destroy the specified Input Context (IC). It will remove it from the ic_list maintained by the Input Method (IM), and then de-allocate memory used by the OLIC structure.

SEE ALSO
OIDCreateIc(3W)
OIDicValues(3W)
OLImOfIc(3W)
OLSetIcFocus(3W)
OLSetIcValues(3W)
OLUnSetIcFocus(3W)
NAME
  *DisplayOfIm – queries a display associated with an IM

SYNOPSIS
  Display * OILDisplayOfIm(im)
  OILm *im;

DESCRIPTION
  This function returns a pointer to the Display corresponding to the given Input
  Method.

SEE ALSO
  OILOpenIm
  OIXOpenDisplay
OlDrawlmageString(3W)

NAME
OlDrawlmageString – a Toolkit drawing function that draws EUC encoded text

SYNOPSIS
OlDrawlmageString(display, drawable, fontlist, gc, x, y, string, len)
   Display *display;
   Drawable drawable;
   OlFontList *fontlist;
   GC gc;
   int x,y;
   unsigned char *string;
   int len;

DESCRIPTION
OlDrawlmageString is a general purpose text drawing utility to draw an internationalized string which may be composed of characters from different code sets. It replaces the Xlib function XDrawlmageString. The argument list is the same as XDrawlmageString with the exception of one additional argument called "fontlist." The fontlist specifies a list of XFontStructs. This utility considers the string to be composed of several segments in which characters belong to the same code set. Each string segment is drawn using the appropriate XFontStruct from the fontlist. Although, the font information in the gc is not used by the function, we recommend that the gc contain font information for default fonts or ASCII fonts to preserve backward compatibility and to optimize drawing for LATIN-1 characters.

NOTES
The len argument specifies the length of string in bytes (not characters) as returned by w_charstrlenstring*.

This function assumes that the string passed to it is a multibyte string.

This function will update the font stored in the GC argument and will not restore the font before returning. If a client relies on a particular font being present in a GC, it must restore the font in the GC after calling of OlDrawlmageString.

SEE ALSO
XDrawlmageString
XDrawlmageString16
OlGetNextStrSegment(3W)
OlDrawString(3W)
NAME
OldDrawString – a Toolkit drawing function that draws EUC encoded text

SYNOPSIS
OldDrawString(display, drawable, fontlist, gc, x, y, string, len)
Display *display;
Drawable drawable;
OIFontList *fontlist;
GC gc;
int x,y;
unsigned char *string;
int len;

DESCRIPTION
OldDrawString is a general purpose text drawing utility to draw an internationalized string which may be composed of characters from different code sets. It replaces the Xlib function XDrawString. The argument list is the same as XDrawString with the exception of one additional argument called "fontlist." The fontlist specifies a list of XFontStructs. This utility considers the string to be composed of several segments in which characters belong to the same code set. Each string segment is drawn using the appropriate XFontStruct from the fontlist. Although, the font information in the gc is not used by the function, we recommend that the gc contain font information for default fonts or ASCII fonts to preserve backward compatibility and to optimize drawing for LATIN-1 characters.

NOTES
The len argument specifies the length of string in bytes (not characters) as returned by w_chartstrlenstring*.

This function assumes that the string passed to it is a multibyte string and not a wide character.

This function will update the font stored in the gc argument and will not restore the font before returning. If a client relies on a particular font being present in a gc, it must restore the font in the gc after calling of OldDrawString.

SEE ALSO
XDrawString
OldDrawString(3W)
OIGetNextStrSegment(3W)
NAME
OlEucToCt – returns a null-terminated string

SYNOPSIS
int OlEucToCt(eucstr, ctstr, ct_len, fontl)
    char *eucstr;
    char *ctstr;
    int ct_len;
    OlFontList *fontl;

DESCRIPTION
Given a null-terminated EUC encoded string in the argument eucstr, this function returns a semantically equivalent null-terminated string in the argument ctstr that conforms to the compound text syntax. Equivalence implies that the characters in both strings are the same and are stored in the same logical order. The code set scheme syntax is the only difference in the two strings. OlEucToCt(3W) will retrieve necessary EUC code set mapping information from the fontl structure.

The memory for storing the ctstr string must be pre-allocated prior to the function call. Doing this, rather than making the function allocate the memory every time it is called, will result in a better performance, because pre-allocated memory could be easily re-used in case of repeated calls to OlEucToCt(3W). The ct_len argument specifies the length of the ctstr and is required to prevent buffer overflow. If the number of bytes indicated by ct_len is too small to store the converted string, -1 is returned.

This function returns the length, in bytes (not counting the terminating null character), of the Compound Text encoded string on success, -1 on failure.

SEE ALSO
OlCtToEuc(3W)
NAME
OlFindHelpFile – used to find the locale-specific help file

SYNOPSIS
String OlFindHelpFile(widget, filename)
  Widget   widget;  /* for getting display info */
  String   filename;  /* file to retrieve */

DESCRIPTION
The OlFindHelpFile procedure is used to retrieve the full name of the passed in
base filename in the current locale. It accesses the application’s help directory
resource to create the full name. The help directory resource is created using the
XFILESEARCHPATH environment variable.
Cursor/Bitmap Utilities

NAME
OlGet50PercentGrey

SYNOPSIS
#include <OlCursors.h>
...
extern Pixmap OlGet50PercentGrey(screen)
Screen * screen;

DESCRIPTION
The OlGet50PercentGrey function is used to retrieve the id of a 50 percent grey Pixmap for screen.

RETURN VALUE
The Pixmap id is returned.
 OlGet75PercentGrey (3W)

Cursor/Bitmap Utilities

NAME
OlGet75PercentGrey

SYNOPSIS
#include <OlCursors.h>

... extern Pixmap OlGet75PercentGrey(screen)
Screen * screen;

DESCRIPTION
The OlGet75PercentGrey function is used to retrieve the id of a 75 percent grey Pixmap for screen.

RETURN VALUE
The Pixmap id is returned.
OlGetApplicationValues (3W)

NAME
OlGetApplicationValues – used to retrieve application resources

SYNOPSIS

void OlGetApplicationValues (widget, args, num_args)
    Widget widget; /* for getting display info */
    ArgList args; /* args to query */
    Cardinal num_args; /* number of args */

DESCRIPTION

The OlGetApplicationValues procedure is used to retrieve the application resources that are accessible from the OPEN LOOK toolkit. The semantics is similar to the XtGetValues call.

Application resources are resources that all OPEN LOOK applications have in common. Their values are updated dynamically by changing preferences in the WorkSpace Manager’s property sheets. Therefore, it’s recommended that an application query the values each time it needs them.

SEE ALSO

NAME
olGetIcValues – used for reading Input Context (IC) attributes

SYNOPSIS
char* olGetIcValues(ic, icvalues)
OlIc *ic;
OlIcValues icvalues;

DESCRIPTION
This function is used for reading IC attributes. icvalues must point to a location where the values will be stored. The function returns NULL if no error occurs; otherwise it returns the name of the first argument that could not be obtained. The end of the icvalues list must be indicated by a NULL value for the attribute name. This function will allocate memory to store the values and it is the caller’s responsibility to free the storage.

SEE ALSO
olCreateIc(3W)
olDestroyIc(3W)
olIcValues(3W)
olImOfIc(3W)
olSetIcFocus(3W)
olSetIcValues(3W)
olUnSetIcFocus(3W)
NAME
OlGetImValues — returns a list of properties and features supported by the Input Method

SYNOPSIS
#define OlImNeedNothing 000
#define OlImPreEditArea 001
#define OlImPreEditCallbacks 002
#define OlImPreEditPosition 004
#define OlImStatusArea 010
#define OlImStatusCallbacks 020
#define OlImFocusTracks 040
typedef unsigned short OlImStyle;
typedef struct _OlImStyles {
    unsigned short styles_count;
    OlImStyle *supported_styles;
} OlImStyles;

void OlGetImValues(im, imvalues)
    OlIm *im;
    OlImValues *imvalues;

DESCRIPTION
This function returns a list of properties and features supported by the IM. The end of the list is indicated by a NULL value for the attribute name in the imvalues list. Only one attribute, XtNQueryInputStyle, is defined at this time. It is used to query input styles supported by the IM.

A client should always query the IM to find out what styles are supported, to determine if they match the styles that the clients intend to support. If there are no matches, the client should either choose another IM or terminate the execution.

The imvalues attr_value field must be a pointer to the OlImStyles structure, which contains a count and an array of supported styles. The imvalues attr_name field should contain the string XtNQueryInputStyle. Each element in that array is a bitmask in which IM indicates its requirements should a particular style to be selected.

Clients are responsible for freeing OlImStyles using XFree.

NOTES
Not all style combinations are allowed.

SEE ALSO
OlOpenIm(3W)
OlImValues(3W)
**NAME**

OlGetMessage – responsible for providing a localized message or a default message.

**SYNOPSIS**

```c
char *OlGetMessage(display, buf, bufsiz, name, type, class, default_msg, database)
Display *display;
char *buf;
int bufsiz;
String name;
String type;
String class;
String default_msg;
XrmDatabase database;
```

**DESCRIPTION**

OlGetMessage will get a unique localized message where the class of the message corresponds to the filename and the name and type of message maps to a unique message in that XrmDatabase format file.

If a message is found and a buffer is supplied, the message is copied into the buffer and the pointer to this buffer returned.

If a message is found and a buffer is not supplied, the pointer returned is the Xt pointer to that message and should not be freed.

If a message is not found and a buffer is supplied, the pointer to that buffer is returned, with the default message copied into it.

If a message is not found and a buffer is not supplied, the pointer returned is a toolkit pointer to the default message and should not be freed.

**SEE ALSO**

OlOpenDatabase(3W)
OlCloseDatabase(3W)
OlVaDisplayWarningMessage(3W)
OlVaDisplayErrorMessage(3W)
OlGetNextStrSegment (3W)

NAME

OlGetNextStrSegment — a string parsing utility to obtain string segments in which all characters belong to the same code set

SYNOPSIS

```c
typedef struct _OlStrSegment {
    unsigned short code_set; /* EUC code set number */
    int len; /* length of the string segment */
    unsigned char *segment; /* string segment */
} OlStrSegment;

int OlGetNextStrSegment(fontlist, segment, str, len)
    FontList *fontlist;
    OlStrSegment *segment;
    unsigned char **str;
    int *len;
```

DESCRIPTION

This function identifies the code set to which the first character (not byte) of the string belongs and copies all subsequent contiguous characters in this string that belong to the same code set into the segment field of the OlStrSegment structure, until a character belonging to another code set is identified. The auxiliary characters such as single shift two and single shift three (552 and 5S3) that identifies the code set of a character in the EUC scheme will not be copied. A large enough space to store a string segment into the segment field must be allocated by the caller. The fontlist contains information about the code sets being used. str points to the original string that must be parsed and moves this pointer to the first character belonging to a different code set. len points to an integer for length (in bytes) of the str. It is also modified to reflect the length of the remaining part of the string.

The function returns -1 if an error occurs, 0 otherwise.

The following example shows how one could obtain string segments from a string using this function.

```c
char *str; /*a string is parsed into segment */
int len; /*holds limited length of string */
OlFontList *fontlist; /*see CvtOlGetFontGroupToFontStringList */
OlStrSegment *parse; /* Large enough must have been pre-allocated */
while (len > 0)
{
    OlGetNextStrSegment (fontlist, parse, &str, &len)
    ...
    do something useful with parse -> str - it points to string segment
}
```
Dynamic Settings

NAME
OlGrabDragPointer

SYNOPSIS
#include <OpenLook.h>
...
extern void OlGrabDragPointer(w, cursor, window)
Widget w;
Cursor cursor;
Window window;

DESCRIPTION
The OlGrabDragPointer procedure is used to effect an active grab of the mouse
pointer. This function is normally called after a mouse drag operation is experi­
enced and prior to calling the OlDragAndDrop procedure which is used to monitor
a drag operation.

SEE ALSO
OlDragAndDrop(3W)
OlUngrabDragPointer(3W)
NAME
OlIcValues – contains a list of IC attribute names and value pairs

SYNOPSIS
typedef struct _OlIcValues {
    char *attr_name;
    void *attr_value;
} OlIcValues;

typedef OlIcValues * OlIcValuesList;

typedef void (*OlImProc)();
typedef void * OlImValue;

typedef struct _OlImCallback {
    OlIcValues client_data;
    OlImProc callback;
} OlImCallback;

DESCRIPTION
OlIcValuesList contains a list of Input Context (IC) attribute names and value pairs. It is used for getting and setting various IC attributes. The supported IC attributes are shown in the table below. The end of the list is indicated by a NULL value in the attribute name.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlNclientWindow</td>
<td>Window*</td>
</tr>
<tr>
<td>OlNClientArea</td>
<td>XRectangle*</td>
</tr>
<tr>
<td>OlNinputStyle</td>
<td>OlImStyle*</td>
</tr>
<tr>
<td>OlNfocusWindow</td>
<td>Window*</td>
</tr>
<tr>
<td>OlNpreeditArea</td>
<td>XRectangle*</td>
</tr>
<tr>
<td>OlNstatusArea</td>
<td>XRectangle*</td>
</tr>
<tr>
<td>OlNspotLocation</td>
<td>XPoint*</td>
</tr>
<tr>
<td>OlNresourceDatabase</td>
<td>XrmDatabase*</td>
</tr>
<tr>
<td>OlNpreeditAttributes</td>
<td>OlIcWindowAttr*</td>
</tr>
<tr>
<td>OlNstatusAttributes</td>
<td>OlIcWindowAttr*</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Attribute Value Type</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>OINcolormap</td>
<td>colormap</td>
</tr>
<tr>
<td>OINstdColormap</td>
<td>colormap</td>
</tr>
<tr>
<td>OINbackground</td>
<td>pixel</td>
</tr>
<tr>
<td>OINforeground</td>
<td>pixel</td>
</tr>
<tr>
<td>OINbackgroundPixmap</td>
<td>pixmap</td>
</tr>
<tr>
<td>OINfontset</td>
<td>OListFont</td>
</tr>
<tr>
<td>OINlineSpacing</td>
<td>int</td>
</tr>
<tr>
<td>OINcursor</td>
<td>cursor</td>
</tr>
<tr>
<td>Callbacks</td>
<td>OListCallback</td>
</tr>
</tbody>
</table>

**OINclientWindow**

specifies the client window in which the IM may display data or create subwindows. Dynamic changes of client window are not supported; this argument must be set at the IC creation time and cannot be changed later. It is a static attribute that is required by OICreateIC. The value is a pointer to a window.

**OINClientArea**

specifies the client area in which IM may display data or create subwindows. IM will establish its own pre-edit and status geometry accordingly. When this attribute is left unspecified, IM will default usable client area to actual client window geometry. It is a dynamic attribute that can be modified via calls to OICValues. The value is a pointer to an XRectangle.

**OINinputStyle**

specifies the input style to be used. The value of this argument must be one of the supported styles returned by the OIGetICValues function, otherwise OICreateIC will fail. If you do not specify this attribute, IM will use an implementation defined default style. OPEN LOOK does not support Dynamic changes of IM style. This argument must be set at the IC creation time and cannot be changed later. The value is a pointer to OICICStyle.

**OINfocusWindow**

specifies to IM the window XID of the focus window. The input method may possibly affect that window: select events on it, send events to it, modify its properties, and grab the keyboard within that window.

When this attribute is left unspecified, IM will default from the focus window to the client window. Setting this attribute explicitly to NULL has a specific meaning: when the focus window is set explicitly from a non NULL value to NULL, the Input Method is required to clear any displayed data in the status area corresponding to the focus window. It is a dynamic attribute that can be modified via calls to OICValues. The value is a pointer to a window.
OllcValues (3W)

**OlnpreeditArea**
the area where pre-edit data should be displayed. The value of this argument is a pointer to XRectangle, relative to the client window. IM may or not create a preedit window in this area, using the specified geometry, as a child of a client window.

When you leave **OlnpreeditArea** unspecified, IM will default from the preedit area to an implementation defined area. This area shall be contained within the client area.

If you specify this attribute for root or XimPreEditCallbacks Input Method, it is ignored.

If you specify this attribute for an XimPreEditArea Input Method, the width and height determine the size of the area within the “over-the-spot” window that is now available for pre-edit.

**OlnstatusArea**
specifies to the IM the usable area to display IC state information. The value of this argument is a pointer to XRectangle, relative to the client window.

The IM may or not create a status window in this area, using the specified geometry, as a child of the client window.

When **OlnstatusArea** is left unspecified, The IM defaults to the status area defined by the IM implementation. This area is contained within the client area. This is a dynamic attribute that can be modified via calls to OllsetICValues.

**Important Note:** if a client leaves all areas unspecified, the IM may not be able to run properly. Some implementations will generate errors if none of the focus window, focus area, client area, preedit area, and status area are defined. At best, it may behave randomly using any area in the client window, possibly clearing the whole window or erasing any region.

**OlnspotLocation**
specifies to IM the coordinates of the “spot” (the current cursor position in the text insertion window), to be used by the “over-the-spot” or “on-the-spot” IMs. The type is a pointer to Xpoint. The x coordinate specifies the position where the next character would be inserted. The y coordinate is the position of the baseline used by current text line in the focus window.

**SEE ALSO**
OllCreateIC(3W), OllDestroyIC(3W), OllGetICValues(3W), OllImOfIC(3W), OllSetICFocus(3W), OllSetICValues(3W), OllUnSetICFocus(3W), OllReSetIC(3W)
NAME
OImOfIc – returns a pointer to OIm

SYNOPSIS
OIm * OImOfIc(ic)
   OIC *ic;

DESCRIPTION
This function returns a pointer to the OIm structure associated with the specified Input Context. A Null value is returned if an invalid ic is specified.

SEE ALSO
OICreateIc(3W)
NAME
OlImValues - contains a list of IM attributes

SYNOPSIS
typedef struct {
    unsigned short count_styles;
    OlIMStyle * supported_styles;
} OlIMStyles

DESCRIPTION
OlImValues contains a list of Input Method values or attributes returned by OlIMStyles structure. The OlIMStyles structure contains in its field count_styles the number of input styles supported. This is also the size of the array in the field supported_styles. Each element in the array represents a different input style supported by this Input Method. It is a bitmask in which the Input Method indicates its requirements, should this style be selected. These requirements fall into the following categories:

OlImPreEditArea If chosen the Input Method requires the client to provide some area values for preediting. Refer to the Input Context Attribute OlNpreeditArea.

OlIMPreeditPosition If chosen, the Input Method requires the client to provide positional values. Refer to IC attributes OlNspotLocation and OlNfocusWindow.

OlImPreEditCallbacks If chosen, the Input Method requires the client to define the set of preedit callbacks. Refer to IC values OlNPreEditStartCallback, OlNPreEditDoneCallback, OlNPreEditDrawCallback, OlNPreEditCaretCallback.

OlImNeedNothing If chosen, the Input Method can function without any PreEdit values.

OlIMStatusArea The input method requires the client to provide some area values for it to do its Status feedback. Refer to OlNArea and OlNAreaNeeded.

OlIMStatusCallbacks The Input Method requires the client to define the set of status callbacks.

OlImStatusArea The Input Method requires the client to provide some area values for it to do its Status feedback. Refer to OlNArea and OlNAreaNeeded.
**NAME**

**OlInitialize** – a mandatory routine that must be called by every application

**SYNOPSIS**

```c
#include <Xlib.h>
#include <OpenLook.h>

Widget OlInitialize(shell_name, classname, urlist, num_urs, argc, argv)

    char *shell_name;
    char *classname;
    XrmOptionDescRec *urlist;
    Cardinal num_urs;
    Cardinal *argc;
    char *argv[];
```

**DESCRIPTION**

This initialization routine must be called by each application before any OPEN LOOK widgets are created or other OPEN LOOK routines are used.

The arguments to this routine are similar to the arguments to the standard **XtInitialize** routine.

**OlInitialize** also does the following:

- A call to the **OlLocaleInitialize** routine to set the locale categories in accordance with the Workspace Manager locale announcement;
- **OlInitialize** sets up communication with an input method library if the locale set by **OlLocaleInitialize** requires and input method.

The following resources are set by the Workspace Manager as part of its locale announcement scheme.

- *xnlLanguage
- *inputLang
- *displayLang
- *numeric
- *timeFormat

**OlInitialize** maps the values of these resources to locale categories, through the `setlocale` function using the following locale category map.

<table>
<thead>
<tr>
<th>X Toolkit</th>
<th>Locale Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>*xnlLanguage</td>
<td>LC_ALL</td>
</tr>
<tr>
<td>*displayLang</td>
<td>LC_MESSAGES</td>
</tr>
<tr>
<td>*numeric</td>
<td>LC_NUMERIC</td>
</tr>
<tr>
<td>*time</td>
<td>LC_TIME</td>
</tr>
<tr>
<td>*inputLang</td>
<td>LC_CTYPE</td>
</tr>
</tbody>
</table>
Locale names (the right hand side values of these resources) announced by the Workspace Manager must match the underlying system locale names.
NAME
 OlLocaleOfIm – queries locale for the input method

SYNOPSIS
 char * OlLocaleOfIm(im)
     OlIm *im;

DESCRIPTION
 This function returns a locale name string under which the specified input method runs.
NAME

OILookuplmString – localized version of Xlib XLookupString function but maps a keypress event to a language string, a keysym, or OIComposeStatus

SYNOPSIS

typedef enum _OIComposeStatus {
    XBufferOverflow,
    XLookupNone,
    XLookupChars,
    XLookupKeysym,
    XLookupBoth
} OIComposeStatus;

int OILookuplmString(event, ic, buffer_return, buffer_len, keysym_return, status_ret,
    KeyEvent *event;
    {IC *ic;
    char *buffer_return;
    int buf_len;
    KeySym *keysym_return;
    OIComposeStatus *status_return;

DESCRIPTION

This function is similar to Xlib function XLookupString except that it takes an extra ic argument and may or may not return any string for a key press event. Instead, it will return a composed character when one is available. If an existing program that uses XLookupString is modified to use OILookuplmString, the program should check for any string that has been returned by the function. The value returned in the status_return value indicates what has been returned in the other arguments. The return value is the length, in bytes, of the string returned in buffer_return if a string has been returned.

- XBufferOverflow means that the input string to be returned is too large for the supplied buffer_return. The required size is returned as the value of the function, and the contents of buffer_return and keysym_return are not modified. The client should call the function again with the same event and a buffer of adequate size in order to obtain the string.

- XLookupNone means that no consistent input has been composed so far. The contents of buffer_return and keysym_return are not modified, and the function returns zero as a value.

- XStringReturned means some input string has been composed. It is placed in buffer_return and the string length is returned as the value of the function. The string is encoded in the locale bound to the Input Context. The contents of keysym_return is not modified.

- XLookupKeySym means a KeySym has been returned instead of a string. The KeySym is returned in keysym_return. The contents of buffer_return is not modified, and the function returns zero.
• **XLookupBoth** means that both a KeySym and a string are returned, in `buffer_return` and `keysym_return` respectively.

It is not necessary for the Input Context passed as an argument to `olLookupImString` to have focus. Input can be composed within this Input Context before it loses focus and that input is returned by the function even though it no longer has keyboard focus. This result is dependent on the IM implementation and may not be true in all cases.

**SEE ALSO**

- `olDrawImString(3W)`
- `XLookupImString(3W)`
- `XLookupKeysym(3W)`
- `XKeysymToString(3W)`
- `XstringToKeySym(3W)`
NAME
OlLookupInputEvent – translates an X event to an OPEN LOOK virtual event

SYNOPSIS
#include <Xol/OpenLook.h>
... 
void OlLookupInputEvent(w, xevent, virtual_event_ret, db_flag)
    Widget w;
    XEvent * xevent;
    OlVirtualEvent virtual_event_ret;
    XtPointer db_flag;

DESCRIPTION
The OlLookupInputEvent procedure is used to translate a X event to an OPEN LOOK virtual event. The X event (xevent) could be a KeyPress, ButtonPress, ButtonRelease, EnterNotify, LeaveNotify, or MotionNotify event. The procedure attempts to translate this event based on the setting of the OPEN LOOK defined dynamic databases.

The virtual_event_ret parameter is a pointer to an OlVirtualEventRec structure:

typedef struct {
    Boolean consumed;
    XEvent * xevent;
    Modifiers dont_care;
    OlVirtualName virtual_name;
    Keysym keysym;
    String buffer;
    Cardinal length;
    Cardinal item_index;
} OlVirtualEventRec, *OlVirtualEvent;

If the X event is a KeyPress, the keysym, buffer, and length, information will be included in virtual_event_ret. The information was returned from a call to XLookupString(3X).

The db_flag parameter is an XtPointer type. The valid values are OL_DEFAULT_IE, OL_CORE_IE, OL_TEXT_IE, or the return value from a OlCreateInputEventDB(3W) call.

The (w, db_flag) pair determines the searching database(s). If the db_flag value is not OL_DEFAULT_IE then only the given database (for example, OL_TEXT_IE means: search the OPEN LOOK TEXT database) will be searched, otherwise a search stack will be built. This stack is based on the widget information (w) and the registering order [see OlSearchClassIEDB(3W), OlSearchClassTextDB(3W), OlSearchWidgetIEDB(3W), and OlSearchWidgetTextDB(3W) for details] to determine the searching database(s). Once this stack is built, the procedure uses the LIFO (Last In First Out) fashion to perform the search.

All widgets have an XtNconsumeEvent callback. When this callback is called, the call_data field is a pointer to an OlVirtualEventRec structure which is filled in with the results of calling OlLookupInputEvent with the db_flag set to OL_DEFAULT_IE.
OPEN LOOK Defined Databases

NOTE: For readability, we have abbreviated the following keys in the Default Bindings column:

<table>
<thead>
<tr>
<th>Key</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift key</td>
<td>s</td>
</tr>
<tr>
<td>Alt key</td>
<td>a</td>
</tr>
<tr>
<td>Ctrl key</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core Database (OL_CORE_IE)</th>
<th>Command Name</th>
<th>Virtual Expression</th>
<th>Virtual Event</th>
<th>Default Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUST</td>
<td>adjustBtn</td>
<td>OL_ADJUST</td>
<td></td>
<td>&lt;Button2&gt;</td>
</tr>
<tr>
<td>ADJUSTKEY</td>
<td>adjustKey</td>
<td>OL_ADJUSTKEY</td>
<td></td>
<td>c &amp;</td>
</tr>
<tr>
<td>CANCEL</td>
<td>cancelKey</td>
<td>OL_CANCEL</td>
<td></td>
<td>&lt;Escape&gt;</td>
</tr>
<tr>
<td>CONSTRAIN</td>
<td>constrainBtn</td>
<td>OL_CONSTRAIN</td>
<td></td>
<td>c &lt;Button1&gt;</td>
</tr>
<tr>
<td>COPY</td>
<td>copyKey</td>
<td>OL_COPY</td>
<td></td>
<td>&lt;Insert&gt;</td>
</tr>
<tr>
<td>CUT</td>
<td>cutKey</td>
<td>OL_CUT</td>
<td></td>
<td>s &lt;Delete&gt;</td>
</tr>
<tr>
<td>DEFAULTACTION</td>
<td>defaultActionKey</td>
<td>OL_DEFAULTACTION</td>
<td></td>
<td>&lt;Return&gt;, &lt;Return&gt;</td>
</tr>
<tr>
<td>DRAG</td>
<td>dragKey</td>
<td>OL_DRAG</td>
<td></td>
<td>&lt;F5&gt;</td>
</tr>
<tr>
<td>DROP</td>
<td>dropKey</td>
<td>OL_DROP</td>
<td></td>
<td>&lt;F2&gt;</td>
</tr>
<tr>
<td>DUPLICATEKEY</td>
<td>duplicateKey</td>
<td>OL_DUPLICATEKEY</td>
<td></td>
<td>s &lt;space&gt;</td>
</tr>
<tr>
<td>HELP</td>
<td>helpKey</td>
<td>OL_HELP</td>
<td></td>
<td>&lt;F1&gt;</td>
</tr>
<tr>
<td>HSBMENU</td>
<td>horizSBMenuKey</td>
<td>OL_HSBMENU</td>
<td>a &lt;bracketleft&gt;</td>
<td></td>
</tr>
<tr>
<td>MENU</td>
<td>menuBtn</td>
<td>OL_MENU</td>
<td></td>
<td>&lt;Button3&gt;</td>
</tr>
<tr>
<td>MENUEDEFAULT</td>
<td>menuDefaultBtn</td>
<td>OL_MENUEDEFAULT</td>
<td>s &lt;Button3&gt;</td>
<td></td>
</tr>
<tr>
<td>MENUEDEFAULTKEY</td>
<td>menuDefaultKey</td>
<td>OL_MENUEDEFAULTKEY</td>
<td>c &lt;M&gt;, &lt;F4&gt;</td>
<td></td>
</tr>
<tr>
<td>MENUKEY</td>
<td>menuKey</td>
<td>OL_MENUEKEY</td>
<td></td>
<td>c &lt;m&gt;, &lt;F4&gt;</td>
</tr>
<tr>
<td>MOVEDOWN</td>
<td>downKey</td>
<td>OL_MOVEDOWN</td>
<td></td>
<td>&lt;Down&gt;</td>
</tr>
<tr>
<td>MOVELEFT</td>
<td>leftKey</td>
<td>OL_MOVELEFT</td>
<td></td>
<td>&lt;Left&gt;</td>
</tr>
<tr>
<td>MOVERIGHT</td>
<td>rightKey</td>
<td>OL_MOVERIGHT</td>
<td></td>
<td>&lt;Right&gt;</td>
</tr>
<tr>
<td>MOVEUP</td>
<td>upKey</td>
<td>OL_MOVEUP</td>
<td></td>
<td>&lt;Up&gt;</td>
</tr>
<tr>
<td>MULTIDOWN</td>
<td>multiDownKey</td>
<td>OL_MULTIDOWN</td>
<td></td>
<td>c &lt;Down&gt;</td>
</tr>
<tr>
<td>MULTILEFT</td>
<td>multiLeftKey</td>
<td>OL_MULTILEFT</td>
<td></td>
<td>c &lt;Left&gt;</td>
</tr>
<tr>
<td>MULTIRIGHT</td>
<td>multiRightKey</td>
<td>OL_MULTIRIGHT</td>
<td></td>
<td>c &lt;Right&gt;</td>
</tr>
<tr>
<td>MULTIUP</td>
<td>multiUpKey</td>
<td>OL_MULTIUP</td>
<td></td>
<td>c &lt;Up&gt;</td>
</tr>
<tr>
<td>NEXTAPP</td>
<td>nextAppKey</td>
<td>OL_NEXTAPP</td>
<td>a &lt;Escape&gt;</td>
<td></td>
</tr>
<tr>
<td>NEXT_FIELD</td>
<td>nextFieldKey</td>
<td>OL_NEXT_FIELD</td>
<td>&lt;Tab&gt;, &lt;Tab&gt;</td>
<td></td>
</tr>
<tr>
<td>NEXTWINDOW</td>
<td>nextWinKey</td>
<td>OL_NEXTWINDOW</td>
<td>a &lt;F6&gt;</td>
<td></td>
</tr>
<tr>
<td>PAGEDOWN</td>
<td>pageDownKey</td>
<td>OL_PAGEDOWN</td>
<td>c &lt;Next&gt;</td>
<td></td>
</tr>
<tr>
<td>PAGELEFT</td>
<td>pageLeftKey</td>
<td>OL_PAGELEFT</td>
<td>c &lt;bracketleft&gt;</td>
<td></td>
</tr>
<tr>
<td>PAGERIGHT</td>
<td>pageRightKey</td>
<td>OL_PAGERIGHT</td>
<td>c &lt;bracketright&gt;</td>
<td></td>
</tr>
<tr>
<td>PAGEUP</td>
<td>pageUpKey</td>
<td>OL_PAGEUP</td>
<td>c &lt;Prior&gt;</td>
<td></td>
</tr>
<tr>
<td>PAN</td>
<td>panBtn</td>
<td>OL_PAN</td>
<td>a &lt;Button1&gt;</td>
<td></td>
</tr>
<tr>
<td>PASTE</td>
<td>pasteKey</td>
<td>OL_PASTE</td>
<td>s &lt;Insert&gt;</td>
<td></td>
</tr>
</tbody>
</table>
## Core Database (OL_CORE_IE)

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Virtual Expression</th>
<th>Virtual Event</th>
<th>Default Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREVAPP</td>
<td>prevAppKey</td>
<td>OL_PREVAPP</td>
<td>a &lt;Escape&gt;</td>
</tr>
<tr>
<td>PREV_FIELD</td>
<td>prevFieldKey</td>
<td>OL_PREV_FIELD</td>
<td>s&lt;Tab&gt;,c s&lt;Tab&gt;</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>propertiesKey</td>
<td>OL_PROPERTY</td>
<td>c&lt;p&gt;</td>
</tr>
<tr>
<td>SCROLLBOTTOM</td>
<td>scrollBottomKey</td>
<td>OL_SCROLLBOTTOM</td>
<td>a&lt;Next&gt;</td>
</tr>
<tr>
<td>SCROOLDDOWN</td>
<td>scrollDownKey</td>
<td>OL_SCROLLDOWN</td>
<td>&lt;Next&gt;</td>
</tr>
<tr>
<td>SCROLLLEFT</td>
<td>scrollLeftKey</td>
<td>OL_SCROLLLEFT</td>
<td>a&lt;Bracketleft&gt;</td>
</tr>
<tr>
<td>SCROLLLEFTEDGE</td>
<td>scrollLeftEdgeKey</td>
<td>OL_SCROLLLEFTEDGE</td>
<td>a s&lt;Braceleft&gt;</td>
</tr>
<tr>
<td>SCROLLRIGHT</td>
<td>scrollRightKey</td>
<td>OL_SCROLLRIGHT</td>
<td>a&lt;Bracketright&gt;</td>
</tr>
<tr>
<td>SCROLLRIGHTEDGE</td>
<td>scrollRightEdgeKey</td>
<td>OL_SCROLLRIGHTEDGE</td>
<td>a s&lt;Braceright&gt;</td>
</tr>
<tr>
<td>SCROLLTOP</td>
<td>scrollTopKey</td>
<td>OL_SCROLLTOP</td>
<td>a&lt;Prior&gt;</td>
</tr>
<tr>
<td>SCROLLUP</td>
<td>scrollUpKey</td>
<td>OL_SCROLLUP</td>
<td>&lt;Prior&gt;</td>
</tr>
<tr>
<td>SELCHARBAK</td>
<td>selCharBakKey</td>
<td>OL_SELCHARBAK</td>
<td>s&lt;Left&gt;</td>
</tr>
<tr>
<td>SELCHARFWD</td>
<td>selCharFwdKey</td>
<td>OL_SELCHARFWD</td>
<td>s&lt;Right&gt;</td>
</tr>
<tr>
<td>SELECT</td>
<td>selectBtn</td>
<td>OL_SELECT</td>
<td>&lt;Button1&gt;</td>
</tr>
<tr>
<td>SELECTKEY</td>
<td>selectKey</td>
<td>OL_SELECTKEY</td>
<td>&lt;Space&gt;,c&lt;Space&gt;</td>
</tr>
<tr>
<td>SELFLIPENDS</td>
<td>selFlipEndsKey</td>
<td>OL_SELFLIPENDS</td>
<td>a&lt;Insert&gt;</td>
</tr>
<tr>
<td>SELLINE</td>
<td>selLineKey</td>
<td>OL_SELLINE</td>
<td>c&lt;Left&gt;</td>
</tr>
<tr>
<td>SELLINEBAK</td>
<td>selLineBakKey</td>
<td>OL_SELLINEBAK</td>
<td>s&lt;Home&gt;</td>
</tr>
<tr>
<td>SELLINEFWD</td>
<td>selLineFwdKey</td>
<td>OL_SELLINEFWD</td>
<td>s&lt;End&gt;</td>
</tr>
<tr>
<td>SELECTWORDBAK</td>
<td>selectWordBakKey</td>
<td>OL_SELECTWORDBAK</td>
<td>c&lt;Left&gt;</td>
</tr>
<tr>
<td>SELECTWORDFWD</td>
<td>selectWordFwdKey</td>
<td>OL_SELECTWORDFWD</td>
<td>c s&lt;Right&gt;</td>
</tr>
<tr>
<td>STOP</td>
<td>stopKey</td>
<td>OL_STOP</td>
<td>c&lt;space&gt;</td>
</tr>
<tr>
<td>TOGGLEPUSHPIN</td>
<td>togglePushpinKey</td>
<td>OL_TOGGLEPUSHPIN</td>
<td>c&lt;t&gt;</td>
</tr>
<tr>
<td>UNDO</td>
<td>undoKey</td>
<td>OL_UNDO</td>
<td>a&lt;BackSpace&gt;</td>
</tr>
<tr>
<td>VSBMENU</td>
<td>vertSBMenuKey</td>
<td>OL_VSBMENU</td>
<td>c&lt;r&gt;</td>
</tr>
<tr>
<td>WINDOWMENU</td>
<td>windowMenuKey</td>
<td>OL_WINDOWMENU</td>
<td>s&lt;Escape&gt;</td>
</tr>
<tr>
<td>WORKSPACEMENU</td>
<td>workspaceMenuKey</td>
<td>OL_WORKSPACEMENU</td>
<td>c&lt;w&gt;</td>
</tr>
</tbody>
</table>
Text Database (OL TEXT IE)

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Virtual Expression</th>
<th>Virtual Event</th>
<th>Default Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARBAK</td>
<td>charBakKey</td>
<td>OL_CHARBAK</td>
<td>&lt;Left&gt;</td>
</tr>
<tr>
<td>CHARFWD</td>
<td>charFwdKey</td>
<td>OL_CHARFWD</td>
<td>&lt;Right&gt;</td>
</tr>
<tr>
<td>DELCHARBAK</td>
<td>delCharBakKey</td>
<td>OL_DELCHARBAK</td>
<td>&lt;BackSpace&gt;</td>
</tr>
<tr>
<td>DELCHARFWD</td>
<td>delCharFwdKey</td>
<td>OL_DELCHARFWD</td>
<td>&lt;Delete&gt;</td>
</tr>
<tr>
<td>DELLINE</td>
<td>delLineKey</td>
<td>OL_DELLINE</td>
<td>a s&lt;Delete&gt;</td>
</tr>
<tr>
<td>DELLINEBAK</td>
<td>delLineBakKey</td>
<td>OL_DELLINEBAK</td>
<td>c&lt;BackSpace&gt;</td>
</tr>
<tr>
<td>DELLINEFWD</td>
<td>delLineFwdKey</td>
<td>OL_DELLINEFWD</td>
<td>c&lt;Delete&gt;</td>
</tr>
<tr>
<td>DELWORDBAK</td>
<td>delWordBakKey</td>
<td>OL_DELWORDBAK</td>
<td>c s&lt;BackSpace&gt;</td>
</tr>
<tr>
<td>DELWORDFWD</td>
<td>delWordFwdKey</td>
<td>OL_DELWORDFWD</td>
<td>c s&lt;Delete&gt;</td>
</tr>
<tr>
<td>DOCEND</td>
<td>docEndKey</td>
<td>OL_DOCEND</td>
<td>&lt;End&gt;</td>
</tr>
<tr>
<td>DOCSTART</td>
<td>docStartKey</td>
<td>OL_DOCSTART</td>
<td>c&lt;Home&gt;</td>
</tr>
<tr>
<td>LINEEND</td>
<td>lineEndKey</td>
<td>OL_LINEEND</td>
<td>&lt;End&gt;</td>
</tr>
<tr>
<td>LINESSTART</td>
<td>lineStartKey</td>
<td>OL_LINESSART</td>
<td>&lt;Home&gt;</td>
</tr>
<tr>
<td>PANEEEND</td>
<td>paneEndKey</td>
<td>OL_PANEEND</td>
<td>c s&lt;End&gt;</td>
</tr>
<tr>
<td>PANESTART</td>
<td>paneStartKey</td>
<td>OL_PANEGET</td>
<td>c s&lt;Home&gt;</td>
</tr>
<tr>
<td>RETURN</td>
<td>returnKey</td>
<td>OL_RETURN</td>
<td>&lt;Return&gt;</td>
</tr>
<tr>
<td>ROWDOWN</td>
<td>rowDownKey</td>
<td>OL_ROWDOWN</td>
<td>&lt;Down&gt;</td>
</tr>
<tr>
<td>ROWUP</td>
<td>rowUpKey</td>
<td>OL_ROWUP</td>
<td>&lt;Up&gt;</td>
</tr>
<tr>
<td>WORDBAK</td>
<td>wordBakKey</td>
<td>OL_WORDBAK</td>
<td>c&lt;Left&gt;</td>
</tr>
<tr>
<td>WORDFWD</td>
<td>wordFwdKey</td>
<td>OL_WORDFWD</td>
<td>c&lt;Right&gt;</td>
</tr>
</tbody>
</table>

SEE ALSO
 OlCreateInputEventDB(3W)
 OlClassSearchIEDB(3W)
 OlClassSearchTextDB(3W)
 OlWidgetSearchIEDB(3W)
 OlWidgetSearchTextDB(3W)
 XtNconsumeEvent discussion in “Primitive Widget Resources” section

EXAMPLE

```c
#include <Xol/OpenLook.h>
...
OlVirtualEventRec ve;

/* To look up the OPEN LOOK CORE database */
OlLookupInputEvent(w, xevent, &ve, OL_CORE_IE);
switch (ve.virtual_name)
{
    case OL_UNKNOWN_INPUT:
        ...
        break;
    case OL_UNKNOWN_BTN_INPUT:
        ...
        break;
```
OLookupInputEvent (3W)    (Database Utilities)    OILookupInputEvent (3W)

    case OL_UNKNOWN_KEY_INPUT:
        ...
        break;
    case OL_ADJUST:
        printf ("pressed the adjustBtn\n");
        ...
        break;
    case OL_ADJUSTKEY:
        printf ("pressed the adjustKey\n");
        ...
        break;
...

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

OIVirtualEventRec ve;
    /* To look up the OPEN LOOK TEXT database */
OLookupInputEvent(w, xevent, &ve, OL_TEXT_IE);
switch (ve.virtual_name)
{
    ...
    case OL_DOCEND:
        printf ("pressed the docEndKey\n");
        ...
        break;
    case OL_LINEEND:
        printf ("pressed the lineEndKey\n");
        ...
        break;
...

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

OIVirtualEventRec ve;
    /* To look up all possible databases */
    /* assume: "w" is a textfield widget */
OLookupInputEvent(w, xevent, &ve, OL_DEFAULT_IE);
switch (ve.virtual_name)
{
    ...
    case OL_ADJUST:
        printf ("pressed the adjustBtn\n");
        ...
        break;
    case OL_ADJUSTKEY:
        printf ("pressed the adjustKey\n");
... break;
...
    case OL_DOCEND:
        printf ("pressed the docEndKey\n");
        ... break;
    case OL_LINEEND:
        printf ("pressed the lineEndKey\n");
        ...
        break;
    ...}
NAME
OlMaxFontInfo – queries the maximum values of font related attributes from all fonts in fontlist

SYNOPSIS
typedef struct _OlFontInfo {
   int ascent;
   int descent;
   int height;
   int width;
} OlFontInfo;

OlFontInfo *
OlMaxFontInfo (fontlist)
OlFontList *fontlist;

DESCRIPTION
This routine determines the maximum values for the four font related attributes from all fonts in the variable fontlist. It then allocates an OlFontInfo structure and fills in the structure with those values. This routine is useful to keep track of character positions in a string composed of characters from different code sets.

NOTES
The caller must free memory of the returned structure by calling XFree.

SEE ALSO
CvtOlFontGroupToFontStructLists(3W)
OlFreeFontList(3W)
OlGetFontList(3W)
OlMaxFontInfo(3W)
NAME
OlOpenDatabase – responsible for opening a localized XrmDatabase format database.

SYNOPSIS
XrmDatabase OlOpenDatabase(display, filename)
    Display *display;
    String filename;

DESCRIPTION
This function is responsible for opening a localized version of a database in XrmDatabase format. The function will resolve the pathname according to the current locale.

If the function call is successful, the XrmDatabase pointer will be returned; if unsuccessful, (XrmDatabase) NULL is returned.

If the database has already been opened by OlOpenDatabase, the cached XrmDatabase pointer is returned.

SEE ALSO
OlCloseDatabase(3W)
OlGetMessage(3W)
OlOpenDatabase(3W)
NAME
OlOpenIm – opens a connection to the Input Method

SYNOPSIS
typedef struct _Ol1mValues {
    char * attr_name;
    void * attr_value;
} Ol1mValues;

typedef Ol1mValues * Ol1mValuesList;

typedef struct _Ol1m {
    Ol1c *iclist; /*input context list */
    Ol1mStyles *im_styles; /*supported re-edit types */
    Ol1mValues* *imvalues; /*current IM attributes */
    char *appl_name; /*application name */
    char *appl_class; /*application class */
    long version /*OPEN LOOK version */
    void *imtype; /*hook for IM specific data*/
} Ol1m;

Ol1m *OlOpenIm(dpy, rdb, res_name, res_class)
    Display dpy;
    XrmDatabase rdb;
    String res_name, res_class;

DESCRIPTION
This is an IM dependent routine responsible for opening a connection to the Input Method. Depending on a particular implementation it may have to start an Input Method server, establish a connection with an already running server, set up a STREAMS connection, or simply create an Ol1m structure.

The routine returns a pointer to the Ol1m structure, a pointer to a list of Ol1c structures, information about supported IM styles, and the Toolkit’s version number (to be used by some IMs).

The imtype field is a hook for attaching implementation dependent data structures.

SEE ALSO
OlCloseIm(3W)
OlDisplayOfIm(3W)
OlGetImValues(3W)
OlImOflc(3W)
OlLocaleOfIm(3W)
Dynamic Settings

NAME
OlRegisterDynamicCallback

SYNOPSIS
#include <OpenLook.h>
...
extern void OlRegisterDynamicCallback(CB, data)
OlDynamicCallbackProc CB;
XtPointer data;

DESCRIPTION
The OlRegisterDynamicCallback procedure is used to add a function to the list of registered callbacks to be called whenever the procedure OlCallDynamicCallbacks is invoked. The OlCallDynamicCallback procedure is invoked whenever the RESOURCE_MANAGER property of the Root Window is updated. The OlCallDynamicCallbacks procedure may also be called directly by either the application or other routines in the widget libraries. The callbacks registered are guaranteed to be called in FIFO order of registration and will be called as

(*CB)(data);

SEE ALSO
OlUnregisterDynamicCallback(3W)
OlCallDynamicCallbacks(3W)
OlRegisterHelp (3W)

NAME

OlRegisterHelp – associates help information with either a widget instance or

class

SYNOPSIS

#include <Intrinsic.h>
#include <OpenLook.h>

void OlRegisterHelp(id_type, id, tag, source_type, source);
OlDefine id_type;
XtPointer id;
String tag;
OlDefine source_type;
XtPointer source;

DESCRIPTION

These resources define the look of the Help window.

<table>
<thead>
<tr>
<th>Default Window Decorations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
</tr>
<tr>
<td>XtNmenuButton</td>
</tr>
<tr>
<td>XtNpushpin</td>
</tr>
<tr>
<td>XtNresizeCorners</td>
</tr>
<tr>
<td>XtNwindowHeader</td>
</tr>
</tbody>
</table>

Associating Help with Widgets or Gadgets

The OlRegisterHelp routine associates help information with either a widget
instance or a widget class. The widget ID or widget class pointer is given in id,
and id_type identifies whether it is a widget or a widget class using one of the
values OL_WIDGET_HELP, OL_CLASS_HELP, or OL_FLAT_HELP, respectively. Use
OL_WIDGET_HELP to register help on gadgets.

The tag value is shown in the title of the help window, as suggested below:

app-name: tag Help

where app-name is the name of the application. More than one help message can
be registered with the same tag. tag can be null, in which case only app-name:
Help is printed.

Help for Flat Widgets

To set the same help message for all items in a flat widget container, use the
OlRegisterHelp routine with id_type set to OL_WIDGET_HELP.

To register help for individual items in a flat widget container, use
OlRegisterHelp with id_type set to OL_FLAT_HELP. Use the following structure to specify object that gets the help message:

typedef struct {
    Widget widget;
    Cardinal item_index;
} OlFlatHelpId;
Format of the Help

The help message is identified in source; source_type identifies the form of the help message:

**OL_STRING_SOURCE**

The source is of type String and contains simple text with embedded newlines. The OlRegisterHelp routine does not copy this source; the application is expected to maintain the original as long as it is registered with a tag.

**OL_DISK_SOURCE**

The source is also of type String, but contains the name of a file that contains the text. The OlRegisterHelp routine does not copy this filename; the application is expected to maintain the original as long as it is registered. The file content is considered to be simple text with embedded newlines.

**OL_INDIRECT_SOURCE**

The source is of type void(*) and is a pointer to an application defined routine. The routine is called after HELP has been clicked. The application is expected to define the type of source in the routine. After the routine has returned, the help information is displayed.

The routine is called as follows:

```
(*source)(id_type, id, src_x, src_y, &source_type, &source);
```

The routine arguments are:

- **id**
  - are the values for the widget class or widget instance that was under the pointer when HELP was pressed. These are the same values registered with the tag.

- **src_x**
  - **src_y**
  - are the coordinates of the pointer when HELP was pressed. These are relative to the upper-left corner of the window.

- **source_type**
  - **source**
  - are pointers to values the application’s routine should set for the help source it wants to display. The only source_type values accepted are OL_STRING_SOURCE and OL_DISK_SOURCE.

**OL_TRANSPARENT_SOURCE**

The source is of type void(*) and is a pointer to an application defined routine. The routine is called after HELP has been invoked. The application is expected to handle the HELP event completely. This might be used by an application that does not want the standard help window (for example, xterm simply generates an escape sequence).

The routine is called as follows:

```
(*source)(id_type, id, src_x, src_y);
```

The routine arguments are:

- **id_type**
id are the values for the widget class or widget instance that was under the pointer when HELP was pressed. These are the same values registered with the tag.

src_x  
src_y are the coordinates of the pointer when HELP was pressed. These are relative to the upper-left corner of the window.

The help window is automatically popped up for the OL_STRING_SOURCE, OL_DISK_SOURCE, and OL_INDIRECT_SOURCE help sources. (It is popped up after the indirect routine returns for the OL_INDIRECT_SOURCE help source.) The application is responsible for popping up a help window (if needed) for the OL_TRANSPARENT_SOURCE help source.

Handling the Help Key Event
When the end user clicks HELP, if the event occurs within a widget or window registered with the OlRegisterHelp routine, the corresponding help message is automatically displayed (for source types OL_STRING_SOURCE and OL_DISK_SOURCE) or the application routine is called (for source types OL_INDIRECT_SOURCE and OL_TRANSPARENT_SOURCE). If the event occurs elsewhere, a default help message is displayed.

If the help key is pressed on a widget, the help routine attempts to look up help on that widget of type OL_WIDGET_HELP. If no help is found, the help routine searches up the widget tree (i.e., goes to the widget's parent) looking for the first widget that has help of type OL_WIDGET_HELP registered. If it finds help registered on one of the original widget's ancestors, the help message for the ancestor will be used. If help is not found, the help routine looks for help of type OL_CLASS_HELP on the original widget. If no help is found, the default message is used.

Separate Help per Application
An application will have, at most, one help message displayed. However, several applications can display their separate help messages simultaneously, in different help windows.

Displaying the Help Message
A help source of type OL_STRING_SOURCE and OL_DISK_SOURCE is displayed in a help window that is 50 ens wide and 10 lines tall. (An en is S/2 points, where S is the current point size.) Lines longer than 50 ens are wrapped at the space(s) between words, or at the nearest character boundary if there is no space at which to wrap. Lines are also wrapped at embedded newlines regardless of their length.

Only spaces and newlines are recognized for format control; all other non-printable characters are silently ignored.

Up to ten lines of the message are visible at once. Messages longer than ten lines have a scrollbar control that allows scrolling non-visible lines into view.

Static Variables
The tag and source values should be statically defined (or allocated and not freed). Using automatic variables here will almost always fail.
NAME
OlResetIC – resets the state of an input context to the initial state

SYNOPSIS

char *OlResetIC(ic)
OlIC *ic  */specifies the IC to be reset*/

DESCRIPTION
OlResetIC resets the input context to its initial state. Any input pending on that context is deleted. Input method is required to clear the pre-edit area, if any, and update the status accordingly. Calling OlResetIC does not change the focus.

SEE ALSO
OlCreateIC(3W)
OlDstryIC(3W)
OlGetICValues(3W)
OlImOfIC(3W)
OlICValues(3W)
OlSetICFocus(3W)
OlSetICValues(3W)
OlUnSetICFocus(3W)
NAME
OlSetGaugeValue

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

DESCRIPTION
This function is an alternate and faster way of setting the slider value of a gauge widget.

SEE ALSO
Gauge(3W)
OlSetApplicationValues (3W)

NAME

OlSetApplicationValues – used to set application resources

SYNOPSIS

void OlSetApplicationValues(widget, args, num_args)

    Widget    widget;  /* for getting display info */
    ArgList   args;     /* args to query */
    Cardinal  num_args; /* number of args */

DESCRIPTION

The OlSetApplicationValues procedure is used to set the application resources that are accessible from the OPEN LOOK toolkit. The semantics is similar to the XtSetValues call.

Application resources are resources that all OPEN LOOK applications have in common. Their values are updated dynamically by changing preferences in the WorkSpace Manager’s property sheets.

SEE ALSO

See the section “Application Resources” for the types and descriptions of available application resources in the Appendix, “Manual Pages: Introduction” in this Programmer’s Guide.
NAME

OlSetICFocus - notifies the Input Method that the focus window attached to the "ic" argument has received keyboard focus

SYNOPSIS

void OlSetICFocus(ic)
OlIC *ic;

DESCRIPTION

This function informs the IM that the text area associated with a particular IC now has the input focus and should receive all the keyboard events. Its implementation depends on the Input Method. Depending on the implementation, this function may update the status information and provide appropriate feedback. For an input method style that includes the OlImFocusTracks mask, the client must call OlSetICFocus(ic) in response to a FocusOut event on the widget associated with the Input Context.

SEE ALSO

OlUnsetFocus(3W)
NAME
OlSetICValues - sets Input Context attributes

SYNOPSIS
char * OlSetICValues(ic, icvalues)
    OlIC *ic;
    OlICValues* icvalues;

DESCRIPTION
This function sets IC attributes. ic specifies the input context to be changed. icvalues is a pointer to a list of attribute names and new value pairs. All values must be appropriate data type, matching the data type imposed by the semantics of the argument. The function returns NULL if all arguments can be set, otherwise it returns the name of the first argument that can not be set. The end of the icvalues list will be indicated by a NULL value for the attribute name.

SEE ALSO
OlCreateIC(3W)
OlDestroyIC(3W)
OlGetICValues(3W)
OlICValues(3W)
NAME
OlTextWidth — get the width in pixels of a localized string

SYNOPSIS
int OlTextWidth(fontlist, string, len)
  FontList *fontlist;
  unsigned char *string;
  int len;

DESCRIPTION
This function returns the width of an EUC string in pixels and as such replaces
the Xlib function XtextWidth. The fontlist is a pointer to a list from
XFontStruct; string, specifies a string whose width in pixels is to be computed,
and the len argument specifies the length of string in bytes (not characters). Upon
successful completion, the function returns the width of the string in pixels, or
otherwise 0.

SEE ALSO
  XTextWidth
  XTextWidth16
NAME
OlTextEditClearBuffer

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>
...
extern Boolean OlTextEditClearBuffer(ctx)
TextEditWidget ctx;

DESCRIPTION
The OlTextEditClearBuffer function is used to delete all of the text associated with the TextEdit widget ctx.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget or if the clear operation fails; otherwise TRUE is returned.

SEE ALSO
OlTextEditUpdate(3W)
TextEdit(3W)
NAME
OlTextEditCopyBuffer

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>

extern Boolean OlTextEditCopyBuffer(ctx, buffer)
TextEditWidget ctx;
char ** buffer;

DESCRIPTION
The OlTextEditCopyBuffer function is used to retrieve a copy of the TextBuffer
associated with the TextEdit Widget ctx. The storage required for the copy is
allocated by this routine; it is the responsibility of the caller to free this storage
when appropriate.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget or if the buffer
cannot be read; otherwise TRUE is returned.

SEE ALSO
OlTextEditReadSubString(3W)
TextEdit(3W)
NAME
OlTextEditCopySelection

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>

extern Boolean OlTextEditCopySelection(ctx, delete)
TextEditWidget ctx;
int delete;

DESCRIPTION
The OlTextEditCopySelection function is used to Copy or Cut the current selection in the TextEdit ctx. If no selection exists or if the TextEdit cannot acquire the CLIPBOARD, FALSE is returned. Otherwise the selection is copied to the CLIPBOARD then, if the delete flag is non-zero, the text is then deleted from the TextBuffer associated with the TextEdit widget (i.e., a CUT operation is performed). Finally, TRUE is returned.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget or if the operation fails; otherwise TRUE is returned.

SEE ALSO
OlTextEditUpdate(3W)
OlTextEditGetCursorPosition(3W)
OlTextEditSetCursorPosition(3W)
OlTextEditReadSubString(3W)
OlTextEditCopyBuffer(3W)
TextEdit(3W)
NAME
OITextEditGetCursorPosition

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>

extern Boolean OITextEditGetCursorPosition
    (ctx, start, end, cursorPosition)
    TextEditWidget ctx;
    TextPosition * start;
    TextPosition * end;
    TextPosition * cursorPosition;

DESCRIPTION
The OITextEditGetCursorPosition function is used to retrieve the current
selection start and end and cursorPosition. If there is no current selection start and
end will both be equal to cursorPosition.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget; otherwise
TRUE is returned.

SEE ALSO
OITextEditSetCursorPosition(3W)
TextEdit
NAME
OlTextEditGetLastPosition

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>

extern Boolean OlTextEditGetLastPosition(ctx, position)
TextEditWidget ctx;
TextPosition * position;

DESCRIPTION
The OlTextEditGetLastPosition function is used to retrieve the position of the last character in the TextBuffer associated with the TextEdit widget ctx.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget; otherwise TRUE is returned.

SEE ALSO
OlTextEditGetCursorPosition(3W)
TextEdit(3W)
NAME
OlTextEditInsert

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>
...
extern Boolean OlTextEditInsert(ctx, buffer, length)
TextEditWidget ctx;
String buffer;
int length;

DESCRIPTION
The OlTextEditInsert function is used to insert a NULL-terminated buffer containing length bytes in the TextBuffer associated with the TextEdit widget ctx. The inserted text replaces the current (if any) selection.

Note: The value of length is not used internally, but is passed on as the length field in the XtNmodifyVerification callback.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget or if the insert operation fails; otherwise TRUE is returned.

SEE ALSO
OlTextEditGetCursorPosition(3W)
TextEdit(3W)
NAME
OlTextEditPaste

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>
...
extern Boolean OlTextEditPaste(ctx)
TextEditWidget ctx;

DESCRIPTION
The OlTextEditPaste function is used to paste the contents of the CLIPBOARD into the TextEdit widget ctx. The current (if any) selection is replaced by the contents of the CLIPBOARD.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget; otherwise TRUE is returned.

SEE ALSO
OlTextEditCopySelection(3W)
TextEdit(3W)
NAME
OlTextEditReadSubString

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>

... 
extern Boolean OlTextEditReadSubString(ctx, buffer, start, end)
TextEditWidget ctx;
char ** buffer;
TextPosition start;
TextPosition end;

DESCRIPTION
The OlTextEditReadSubString function is used to retrieve a copy of a substring from the TextBuffer associated with the TextEdit Widget ctx between positions start through end inclusive. The storage required for the copy is allocated by this routine; it is the responsibility of the caller to free this storage when appropriate.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget or if the operation fails; otherwise TRUE is returned.

SEE ALSO
OlTextEditCopyBuffer(3W)
TextEdit(3W)
NAME
O1TextEditRedraw

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>
...
extern Boolean O1TextEditRedraw(ctx)
TextEditWidget ctx;

DESCRIPTION
The O1TextEditRedraw function is used to force a complete refresh of the TextEdit widget display. This routine does nothing if the TextEdit widget is not realized or if the update state is set to FALSE.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget or if the widget is not realized or if the update state is FALSE; otherwise TRUE is returned.

SEE ALSO
O1TextEditUpdate(3W)
TextEdit(3W)
NAME
OlTextEditSetCursorPosition

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>

extern Boolean OlTextEditSetCursorPosition
    (ctx, start, end, cursorPosition)
TextEditWidget ctx;
TextPosition start;
TextPosition end;
TextPosition cursorPosition;

DESCRIPTION
The OlTextEditSetCursorPosition function is used to change the current selection start and end and cursorPosition. The function does NOT check (for efficiency) the validity of the positions. If invalid values are given results are unpredictable. The function attempts to ensure that the cursorPosition is visible by scrolling the display.

RETURN VALUE
FALSE is returned if the widget supplied is not a TextEdit Widget; otherwise TRUE is returned.

SEE ALSO
OlTextEditGetCursorPosition(3W)
TextEdit(3W)
NAME
OlTextEditTextBuffer

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <TextEdit.h>

...  
extern TextBuffer * OlTextEditTextBuffer(ctx)  
TextEditWidget ctx;

DESCRIPTION
The OlTextEditTextBuffer function is used to retrieve the TextBuffer pointer associated with the TextEdit widget ctx. This pointer can be used to access the facilities provided by the Text Buffer Utilities module.

SEE ALSO
TextBufferUtilities(3W)
TextEdit(3W)
NAME
   OlTextEditUpdate

SYNOPSIS
   #include <buffutil.h>
   #include <textbuff.h>
   #include <TextEdit.h>
   
   extern Boolean OlTextEditUpdate(ctx, state)
   TextEditWidget ctx;
   Boolean state;

DESCRIPTION
   The OlTextEditUpdate function is used to set the updateState of a
   TextEdit Widget. Setting the state to FALSE turns screen update off; setting
   the state to TRUE turns screen updates on and refreshes the display.

RETURN VALUE
   FALSE is returned if the widget supplied is not a TextEdit Widget; otherwise
   TRUE is returned.

SEE ALSO
   OlTextEditRedraw(3W)
   TextEdit(3W)
OITextFieldCopyString(3W)

NAME

OITextFieldCopyString

SYNOPSIS

#include <buffutil.h>
#include <textbuff.h>
#include <TextField.h>
...
extern int OITextFieldCopyString(tfw, string)
TextFieldWidget tfw;
char * string;

DESCRIPTION

The OITextFieldCopyString function is used to copy the string associated with
the TextField widget tfw into the user supplied area pointed to by string. The
function returns the length of this string.

SEE ALSO

OITextFieldGetString(3W)
TextField Widget(3W)
NAME
OlTextFieldGetString

SYNOPSIS
#include <buffutil.h>
#include <textbuff.h>
#include <textField.h>
...
extern char *OlTextFieldGetString(tfw, size)
TextFieldWidget tfw;
int * size;

DESCRIPTION
The OlTextFieldGetString function is used to retrieve a new copy of the string associated with the TextField widget tfw. The function returns a pointer to the newly allocated string copy. Optionally, if size is not NULL, the function returns in size the length of the string.

SEE ALSO
TextField Widget(3W)
OlTextFieldCopyString(3W)
NAME
OlUngrabDragPointer

SYNOPSIS
#include <OpenLook.h>
...
extern void OlUngrabDragPointer(w)
Widget w;

DESCRIPTION
The OlUngrabDragPointer procedure is used to relinquish the active pointer grab
which was initiated by the OlGrabDragPointer procedure. This function simply
ungrabs the pointer.

SEE ALSO
OlDetermineMouseAction(3W)
OlGrabDragPointer(3W)
OlDragAndDrop(3W)
Dynamic Settings

NAME
OlUnregisterDynamicCallback

SYNOPSIS
#include <OpenLook.h>

... extern int OlUnregisterDynamicCallback(CB, data) 
OlDynamicCallbackProc CB; 
XtPointer data;

DESCRIPTION
The OlUnregisterDynamicCallback procedure is used to remove a function from the list of registered callbacks to be called whenever the procedure OlCallDynamicCallbacks is invoked.

SEE ALSO
OlRegisterDynamicCallback(3W) 
OlCallDynamicCallbacks(3W)

RETURN VALUE
Zero (0) is returned if the dynamic callback cannot be removed; otherwise one (1) is returned.
NAME
OlUnSetIcFocus – notifies the Input Method that the focus area attached to the
ic no longer has input focus

SYNOPSIS
void OlUnsetIcFocus(ic)
O1Ic *ic;

DESCRIPTION
This function informs the Input Method that the text insertion window associated
with a particular IC no longer has the input focus and should not receive the key­
board events. Its implementation depends on the Input Method. The input
method may choose to give some visual feedback to indicate loss of focus (for
example, change color of cursor in pre-edit text).

SEE ALSO
OlSetIcFocus(3W)
NAME
OIWidgetToClassName - finds the class name for a widget

SYNOPSIS
String OIWidgetToClassName(w)
Widget w;

DESCRIPTION
Given a widget, the function returns the classname of the widget.

SEE ALSO
OIWidgetClassToClassName(3W)
OlWidgetClassToClassName(3W)

NAME

OlWidgetClassToClassName - finds the classname for a widget class

SYNOPSIS

String OlWidgetClassToClassName(wc)
    WidgetClass wc;

DESCRIPTION

Given a widget class, the function returns the classname of the widget.

SEE ALSO

OlWidgetToClassName(3W)
NAME
OIWidgetSearchTextDB — register the OPEN LOOK TEXT database on a specific widget instance

SYNOPSIS
#include <Xol/OpenLook.h>
...
void
OIWidgetSearchTextDB(w)

DESCRIPTION
The OIWidgetSearchTextDB procedure is used to register the OPEN LOOK TEXT database on a specific widget instance.

Once the OPEN LOOK TEXT database is registered with a given widget instance, the OILookupInputEvent(3W) procedure (if db flag is OL_DEFAULT_IE or OL_TEXT_IE) will include this database in the search stack if the given widget id is this widget instance.

NOTES
The registering order determines the searching order when doing a lookup.

SEE ALSO
OIClassSearchIEDB(3W),
OIClassSearchTextDB(3W),
OICreateInputEventDB(3W),
OILookupInputEvent(3W),
OIWidgetSearchIEDB(3W)

EXAMPLE
...
#include <Xol/OpenLook.h>
...

/* assume: "w" is a stub widget that is interested in */
/* the OPEN LOOK TEXT database */
OIWidgetSearchTextDB(w);
/* once this step is done, this widget instance will */
/* receive OPEN LOOK TEXT commands after a call */
/* to OILookupInputEvent(), or in the XtNconsumeEvent */
/* callback's OIVirtualEvent structure supplied with */
/* the call_data field. */

...
**NAME**
OlWidgetSearchIEDB — register a given database on a specific widget instance

**SYNOPSIS**
```
#include <Xol/OpenLook.h>
...
void
OlWidgetSearchIEDB(w, db)
    Widget w;
    OlVirtualEventTable db;
```

**DESCRIPTION**
The `OlWidgetSearchIEDB` procedure is used to register a given database on a specific widget instance. The `db` value was returned from a call to `OlCreateInputEventDB(3W)`.

Once a database is registered with a given widget instance, the `OlLookupInputEvent(3W)` procedure (if `db_flag` is `OL_DEFAULT_IE` or `db`) will include this database in the search stack if the given widget id is this widget instance.

**NOTES**
The registering order determines the searching order when doing a lookup.

**SEE ALSO**
OlClassSearchIEDB(3W)
OlClassSearchTextDB(3W)
OlCreateInputEventDB(3W)
OlLookupInputEvent(3W),
OlWidgetSearchTextDB(3W)

**EXAMPLE**
```
/* To create a client application database */
...
#include <Xol/OpenLook.h>
...
/* start with a big value to avoid */
/* the "virtual_name" collision */
#define OL_MY_BASE 1000
#define OL_MY_DRAWLINEBTN OL_MY_BASE+0
#define OL_MY_DRAWARCBTN OL_MY_BASE+1
#define OL_MY_REDISPLAYKEY OL_MY_BASE+2
#define OL_MY_SAVEPARTKEY OL_MY_BASE+3
#define XtNmyDrawLineBtn "myDrawLineBtn"
#define XtNmyDrawArcBtn "myDrawArcBtn"
#define XtNmyRedisplayKey "myRedisplayKey"
#define XtNmySavePartKey "mySavePartKey"

static OlKeyOrBtnRec OlMyBtnInfo[] = {
    /*name default_value virtual_name */
    { XtNmyDrawLineBtn, "c<Button1>", OL_MY_DRAWLINEBTN },
};
```
{ XtNmyDrawArcBtn, "s<myDrawLineBtn>", OL_MY_DRAWARCBTN },
};

static OIKeyOrBtnRec OIMyKeyInfo[] = {
    /*name default_value virtual_name */

    { XtNmyRedisplayKey, "c<F5>", OL_MY_REDISPLAYKEY },
    { XtNmySavePartKey, "c<F5>", OL_MY_SAVEPARTKEY },
};

static OIVirtualEventTable OIMyDB;

... OIMyDB = OICreateInputEventDB(
    w,
    OIMyKeyInfo, XtNumber(OIMyKeyInfo),
    OIMyBtnInfo, XtNumber(OIMyBtnInfo)
);
...

    /* assume: "w" is a stub widget that is interested in */
    /*
    / * OIMyDB */
    OIWidgetSearchIEDB(w, OIMyDB);
    /* once this step is done, this widget instance will */
    /* receive OIMyDB commands after a call to */
    /* OILookupInputEvent(), or in the XtNconsumeEvent */
    /* callback’s OIVirtualEvent structure supplied with */
    /* the call_data field. */
    ...

Page 2  3/91
NAME

OlWMProtocolAction – simulates a response to any window manager’s protocol messages

SYNOPSIS

    void OlWMProtocolAction(w, st, action)
    
    Where:

    typedef struct {
        int msgtype;       /* type of WM msg */
        XEvent * xevent;
    } OlWMProtocolVerify;

DESCRIPTION

This routine can be used to simulate a response to any window manager’s protocol messages. The OlWMProtocolVerify field msgtype is an integer constant indicating the type of protocol message which invoked the callback and has a range of values of:

    OL_WM_TAKE_FOCUS
    OL_WM_SAVE_YOURSELF
    OL_WM_DELETE_WINDOW

The w parameter must be a widget that is a subclass of the VendorShell. Otherwise, no action will be taken.

The action parameter can be:

    OL_QUIT: quit the application immediately.
    OL_DEFAULTACTION: perform the action that is appropriate for each subclass of VendorShell.
    OL_DESTROY: destroy the shell widget.
    OL_DISMISS: dismiss or unmap the shell widget.

SEE ALSO

See the section on “Shell Resources” in the Appendix “Manual Pages: Introduction” in this Programmer’s Guide.
NAME
O1UpdateDisplay – process all pending exposure events immediately

SYNOPSIS
#include <Xol/OpenLook.h>
...

void
O1UpdateDisplay(w)
   Widget w;

DESCRIPTION
The O1UpdateDisplay procedure is used to process all pending exposure events
so that the appearance of a given widget can be updated right away.

Normally, an operation is accomplished by a set of callback functions. If one of
the callback functions performs a time-consuming action, it is possible that the
portion of an application window will not be redrawn right away after a XtSet-
Values call. This is because the normal exposure processing does not occur until
all callback functions have been invoked. This situation can be resolved by calling
this function before starting a time-consuming action.

EXAMPLE
...
#include <Xol/OpenLook.h>
...
extern Widget status_area; /* a staticText widget */
...
void
fooCB(w, client_data, call_data)
   Widget w;
   XtPointer client_data;
   XtPointer call_data;
{
   ...
   Arg args[5];

   /* display the status in the footer area */ /* before the actual operation */
   XtSetArg(args[0], XtNstring,
            "Start the operation, please wait ...");
   XtSetValues(status_area, args, 1);
   /* show the status in the footer area right away*/
   O1UpdateDisplay(status_area);

   /* now we can start the actual operation */
   ...
   return;
}
NAME

Packed_Widget: OlCreatePackedWidgetList, OlPackedWidget – a convenience routine that allows an application to create a widget tree or subtree in one call

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>

typedef struct {
    Widget widget_returned;
    String name;
    WidgetClass *class_ptr;
    Widget *parent_ptr;
    String descendant;
    ArgList resources;
    Cardinal num_resources;
    Boolean managed;
} OlPackedWidget;

Widget = OlCreatePackedWidgetList(pw_list, num_pw)
OlPackedWidget *pw_list;
Cardinal num_pw;
```

DESCRIPTION

Create Widget (Sub)Trees in One Call

The OlCreatePackedWidgetList routine and its associated OlPackedWidget structure allow an application to create a widget tree or subtree in one call.

The tree is pointing to `pw_list`. Each element in this array is of the type `OlPackedWidget`. This structure gives all the information needed to create a new widget:

- `widget_returned` will contain the ID of the newly created widget.
- `name` is the name of the widget that will be created.
- `class_ptr` is a pointer to the `WidgetClass` pointer for the new widget. This gives the class of widget to create. It is a pointer to the pointer because typically the pointer itself is an external value that is not suitable for using in an array initialization; the pointer to the pointer is.
- `parent_ptr` is a pointer to the widget ID of the intended parent of the new widget or the ID of an indirect widget that "knows who the parent is" (see below). This value may point to a .widget member in another PackedWidget item; if the parent is an indirect widget, it must appear earlier in the list.
- `descendant` is the name of a resource available in the widget identified by `parent_ptr`. The value of this resource is the ID of the real parent for the new widget.
If the .descendant value is not zero, .parent is expected to identify an indirect parent that is interrogated for the ID of the real parent. If this value is zero, .parent is expected to identify the real parent.

resources
is the resource array to use when creating the new widget.

num_resources
is the number of resources in the array.

managed
is TRUE if the new widget should be managed when created, FALSE otherwise.

The OlCreatePackedWidgetList is passed a pointer to an OlPackedWidget array and the number of elements in the array. It creates widgets starting from the first element in the array, and returns the ID of the topmost widget.
Pixel Conversion (3W)

NAME

Pixel_Conversion – a group of routines which examine the data structures that describe the physical dimensions and the pixel resolution of a screen

SYNOPSIS

#include <Xlib.h>
#include <OpenLook.h>

Screen *OlDefaultScreen;
Display *OlDefaultDisplay;
Axis axis;
Screen screen;
OlMMToPixel(axis, millimeters);
Ol_MMToPixel(axis, millimeters);
OlPointToPixel(axis, points);
Ol_PointToPixel(axis, points);
OlScreenMMToPixel(axis, millimeters, screen);
Ol_ScreenMMToPixel(axis, millimeters, screen);
OlScreenPointToPixel(axis, points, screen);
Ol_ScreenPointToPixel(axis, points, screen);
OlPixelToMM(axis, pixels);
Ol_PixelToMM(axis, pixels);
OlPixelToPoint(axis, pixels);
Ol_PixelToPoint(axis, pixels);
OlScreenPixelToPoint(axis, pixels, screen);
Ol_ScreenPixelToPoint(axis, pixels, screen);
OlScreenPixelToMM(axis, pixels, screen);
Ol_ScreenPixelToMM(axis, pixels, screen);

DESCRIPTION

All the X-based OPEN LOOK widgets refer to pixels in coordinates and dimensions for compatibility with other X Window System widgets. However, this puts the burden on the application programmer to convert between externally useful measures, such as points or millimeters, and pixels as applied to the screen at hand. These routines examine the data structures that describe the physical dimensions and the pixel resolution of a screen and convert among millimeters, points, and pixels for that screen.

Which Screen?

The shorter forms of these routines (the ones without the word Screen in their names) work for the default screen. This is the screen that is active when the X Toolkit Intrinsics are started. The longer forms of these routines take a Screen * type argument that refers to a particular screen.

The macros OlDefaultScreen and OlDefaultDisplay identify the current screen and display being used by the Intrinsics. Although the SYNOPSIS above implies these are variables of type Screen * and Display *, respectively, they are really macros that produce values of these types.
Note: Use After Toolkit Initialization

These routines make use of data structures that are initialized when the Toolkit is initialized (see OllInitialize later in this document). Therefore, using them before Toolkit initialization (for example, as an initial value to a statically defined variable) will result in a run time error.

Axis Argument

The first argument of all the routines is the direction in which the measurement is made. This is necessary because not all screens have equivalent resolution in the horizontal and vertical axes. The axis argument can take one of the two values: OL_HORIZONTAL or OL_VERTICAL. These routines are not directly usable in computing a diagonal measure. (Find the diagonal with the Pythagorean Theorem: $a^2 + b^2 = c^2$)

Implemented as Macros

All these routines are implemented as macros, so they can take any reasonable type value for the millimeters, points, and pixels. The macros cast the values into the proper type needed for the conversion. However, only a single type value can be "returned". The routines without an underscore in their names produce values of type int (the values are rounded to the nearest integer). The routines with an underscore in their names produce values of type double (these values have not been rounded, leaving it up to the application to round up, round down, or truncate as needed). Given the small size of the units involved, the integer returning routines should be sufficient for many applications.

Because these routines are implemented as macros, there are no function addresses available.
Text Buffer Utilities

NAME
PositionOfLine

SYNOPSIS
#include <textbuff.h>
...
extern TextPosition PositionOfLine(text, lineindex)
TextBuffer * text;
TextLine lineindex;

DESCRIPTION
The PositionOfLine function is used to translate a lineindex in the text TextBuffer to a TextPosition. It returns the translated TextPosition or EOF if the lineindex is invalid.

SEE ALSO
LineOfPosition(3W)
PositionOfLocation(3W)
LocationOfPosition(3W)
Text Buffer Utilities

NAME
PositionOfLocation

SYNOPSIS
#include <textbuff.h>
...
extern TextPosition PositionOfLocation(text, location)
TextBuffer * text;
TextLocation location;

DESCRIPTION
The PositionOfLocation function is used to translate a location in the text
TextBuffer to a TextPosition. The function returns the translated TextPosition or
EOF if the location is invalid.

SEE ALSO
PositionOfLine(3W)
LocationOfPosition(3W)
NAME

PreviousLocation

SYNOPSIS

```c
#include <textbuff.h>

... extern TextLocation PreviousLocation(textBuffer, current)
TextBuffer * textBuffer;
TextLocation current;
```

DESCRIPTION

The `PreviousLocation` function returns the Location which precedes the given `current` location in a TextBuffer. If the current location points to the beginning of the TextBuffer this function wraps.

SEE ALSO

`NextLocation(3W)`
NAME
PreviousTextBufferWord

SYNOPSIS
#include <textbuff.h>
...
extern TextLocation PreviousTextBufferWord(textBuffer, current)
TextBuffer * textBuffer;
TextLocation current;

DESCRIPTION
The PreviousTextBufferWord function is used to locate the beginning of a word in a TextBuffer relative to a given current location. It returns the location of the beginning of the word which precedes the given current location. If the current location is within a word this function will skip over the current word. If the current word is the first word in the TextBuffer the function wraps to the end of the buffer.

SEE ALSO
PreviousTextBufferWord(3W)
Text Buffer Utilities

NAME
ReadFileIntoBuffer

SYNOPSIS
#include <buffutil.h>

... extern int ReadFileIntoBuffer(fp, buffer)
FILE * fp;
Buffer * buffer;

DESCRIPTION
The ReadFileIntoBuffer function reads the file associated with fp and inserts the characters read into the buffer. The read operation terminates when either EOF is returned when reading the file or when a NEWLINE is encountered. The function returns the last character read to the caller (either EOF or NEWLINE).

SEE ALSO
ReadStringIntoBuffer(3W)
NAME
ReadFileIntoTextBuffer

SYNOPSIS
#include <textbuff.h>

extern TextBuffer * ReadFileIntoTextBuffer(filename, f, d)
char * filename;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION
The ReadFileIntoTextBuffer function is used to read the given file into a newly allocated TextBuffer. The supplied TextUpdateFunction and data pointer are associated with this TextBuffer.

SEE ALSO
ReadStringIntoTextBuffer(3W)
Text Buffer Utilities

**NAME**

ReadStringIntoBuffer

**SYNOPSIS**

```
#include <buffutil.h>
...
extern int ReadStringIntoBuffer(sp, buffer)
Buffer * sp;
Buffer * buffer;
```

**DESCRIPTION**

The `ReadStringIntoBuffer` function reads the buffer associated with `sp` and inserts the characters read into `buffer`. The read operation terminates when either EOF is returned when reading the buffer or when a NEWLINE is encountered. The function returns the last character read to the caller (either EOF or NEWLINE).

**SEE ALSO**

ReadFileIntoBuffer(3W)
Text Buffer Utilities

NAME
ReadStringIntoTextBuffer

SYNOPSIS
#include <textbuff.h>

extern TextBuffer *
ReadStringIntoTextBuffer(string, f, d)
char * string;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION
The ReadStringIntoTextBuffer function is used to copy the given string into a newly allocated TextBuffer. The supplied TextUpdateFunction and data pointer are associated with this TextBuffer.

SEE ALSO
ReadFileIntoTextBuffer(3W)
REGISTERTEXTBUFFERSCANFUNCTIONS (3W)

Text Buffer Utilities

NAME

REGISTERTEXTBUFFERSCANFUNCTIONS

SYNOPSIS

```
#include <textbuff.h>
...
extern void REGISTERTEXTBUFFERSCANFUNCTIONS(forward, backward)
char * (*forward)();
char * (*backward)();
```

DESCRIPTION

The REGISTERTEXTBUFFERSCANFUNCTIONS procedure provides the capability to replace the scan functions used by the ForwardScanTextBuffer and BackwardScanTextBuffer functions. These functions are called as:

```
(*forward)(string, curp, expression);
(*backward)(string, curp, expression);
```

and are responsible for returning either a pointer to the beginning of a match for the expression or NULL.

Calling this procedure with NULL function pointers reinstates the default regular expression facility.
RegisterTextBufferUpdate

Text Buffer Utilities

NAME

RegisterTextBufferUpdate

SYNOPSIS

#include<textbuff.h>
...
extern void RegisterTextBufferUpdate(text, f, d)
TextBuffer * text;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION

The RegisterTextBufferUpdate procedure associates the TextUpdateFunction f and data pointer d with the given TextBuffer text. This update function will be called whenever an update operation is performed on the TextBuffer. See ReplaceBlockInTextBuffer for more details.

SEE ALSO

UnregisterTextBufferUpdate(3W)
ReadStringIntoTextBuffer(3W)
ReadFileIntoTextBuffer(3W)

NOTES

Calling this function increments a reference count mechanism used to determine when to actually free the TextBuffer. Calling the function with a NULL value for the function circumvents this mechanism.
RegisterTextBufferWordDefinition

NAME
RegisterTextBufferWordDefinition

SYNOPSIS
#include <textbuff.h>
...
extern void RegisterTextBufferWordDefinition(word_definition)
int (*word_definition)();

DESCRIPTION
The RegisterTextBufferWordDefinition procedure provides the capability to replace the word definition function used by the TextBuffer Utilities. This function is called as:

(*word_definition)(c);

The function is responsible for returning non-zero if the character c is considered a character that can occur in a word and zero otherwise.

Calling this function with NULL reinstates the default word definition which allows the following set of characters: a-zA-Z0-9_
Text Buffer Utilities

NAME
ReplaceBlockInTextBuffer

SYNOPSIS
#include <textbuff.h>

extern EditResult ReplaceBlockInTextBuffer(text, startloc, endloc, string, f, d)

TextBuffer * text;
TextLocation * startloc;
TextLocation * endloc;
char * string;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION
The ReplaceBlockInTextBuffer function is used to update the contents of the
TextBuffer associated with text. The characters stored between startloc and
endloc are deleted and the string is inserted after startloc. If the edit succeeds
the TextUpdateFunction f is called with the parameters: d, text, and 1; then any
other different update functions associated with the TextBuffer are called with
their associated data pointer, text, and 0.

This function records the operation performed in TextUndoItem structures. The
contents of these structures can be used to implement an undo function. The
contents can also be used to determine the type of operation performed. A struc-
ture is allocated for both the delete and insert information.

The hints provided in these structures is the inclusive or of:

TEXT BUFFER_NOP
TEXT BUFFER_INSERT_LINE
TEXT BUFFER_INSERT_SPLIT_LINE
TEXT BUFFER_INSERT_CHARS
TEXT BUFFER_DELETE_START_LINE
TEXT BUFFER_DELETE_END_LINE
TEXT BUFFER_DELETE_START_CHARS
TEXT BUFFER_DELETE_END_CHARS
TEXT_BUFFER_DELETE_JOIN_LINE
TEXT_BUFFER_DELETE_SIMPLE

SEE ALSO
ReplaceCharInTextBuffer(3W)
ReplaceCharInTextBuffer (3W)

Text Buffer Utilities

NAME
ReplaceCharInTextBuffer

SYNOPSIS
#include <textbuff.h>

extern EditResult ReplaceCharInTextBuffer(text, location, c, f, d)
TextBuffer * text;
TextLocation * location;
int c;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION
The ReplaceCharInTextBuffer function is used to replace the character in the
TextBuffer text at location with the character c.

SEE ALSO
ReplaceBlockInTextBuffer(3W)
SaveTextBuffer (3W)

Text Buffer Utilities

NAME
  SaveTextBuffer

SYNOPSIS
  #include <textbuff.h>
  ...
  extern SaveResult SaveTextBuffer(text, filename)
  TextBuffer * text;
  char * filename;

DESCRIPTION
  The SaveTextBuffer function is used to write the contents of the text TextBuffer to the file filename. It returns a SaveResult which can be:

  SAVE_FAILURE
  SAVE_SUCCESS
Text Buffer Utilities

NAME
StartCurrentTextBufferWord

SYNOPSIS
#include <textbuff.h>
...
extern TextLocation StartCurrentTextBufferWord(textBuffer, current)
TextBuffer * textBuffer;
TextLocation current;

DESCRIPTION
The StartCurrentTextBufferWord function is used to locate the beginning of a word in the TextBuffer relative to a given current location. The function returns the location of the beginning of the current word.

NOTES
The return value will equal the given current value if the current location is the beginning of a word.

SEE ALSO
PreviousTextBufferWord(3W)
NextTextBufferWord(3W)
TextBuffer Macros (3W)

NAME
TextBuffer Macros: TextBufferUserData, TextBufferName, TextBufferModified, TextBufferEmpty, TextBufferNamed, LinesInTextBuffer, LastTextBufferLine, LastCharacterInTextBufferLine, LengthOfTextBufferLine, SameTextLocation

SYNOPSIS
#include <textbuff.h>
TextBufferUserData(text, line)
TextBufferName(text)
TextBufferModified(text)
TextBufferEmpty(text)
TextBufferNamed(text)
LinesInTextBuffer(text)
LastTextBufferLine(text)
LastCharacterInTextBufferLine(text, line)
LengthOfTextBufferLine(text, line)
SameTextLocation(x, y)

DESCRIPTION
These macros are provided for use with the Text Buffer Utilities.

TextBufferUserData used to access the per-line user data.
TextBufferName returns the filename associated with text.
TextBufferModified returns a flag indicating whether text has been modified since last saved.
TextBufferEmpty returns a flag indicating whether text is empty.
TextBufferNamed returns a flag indicating whether text is associated with a filename.
LinesInTextBuffer returns the number of lines in text.
LastTextBufferLine returns the line number of the last line in text.
LastCharacterInTextBufferLine returns the offset of the last character in text on line.
LengthOfTextBufferLine returns the length of line in text.
SameTextLocation returns a flag indicating whether location x and y are the same.
UnregisterTextBufferUpdate (3W)

Text Buffer Utilities

NAME

UnregisterTextBufferUpdate

SYNOPSIS

#include<textbuff.h>
...
extern int UnregisterTextBufferUpdate(text, f, d)
TextBuffer * text;
TextUpdateFunction f;
caddr_t d;

DESCRIPTION

The UnregisterTextBufferUpdate function disassociates the TextUpdateFunction f and data pointer d with the given TextBuffer text. If the function/data pointer pair is not associated with the given TextBuffer zero is returned otherwise the association is dissolved and one is returned.

SEE ALSO

RegisterTextBufferUpdate(3W)
FreeTextBuffer(3W)
NAME
strclose

SYNOPSIS
#include <buffutil.h>
...
extern void strclose(sp)
Buffer * sp;

DESCRIPTION
The strclose procedure is used to close a string Buffer which was opened using
the stropen function.

SEE ALSO
stropen(3W)
strgetc(3W)
NAME
  streexp
SYNOPSIS
  #include <regexp.h>
  ...
  extern char * streexp()
DESCRIPTION
  The streexp function is used to retrieve the pointer of the last character in a match
  found following a strexp/strrexp function call.
SEE ALSO
  strexp(3W)
  strrexp(3W)
Regular Expression Utilities

NAME
  strexp

SYNOPSIS
  #include <expcmp.h>
  ...
  extern char * strexp(string, curp, expression)
  char * string;
  char * curp;
  char * expression;

DESCRIPTION
  The strexp function is used to perform a regular expression forward scan of string
  for expression starting at curp.

THE REGULAR EXPRESSION LANGUAGE USED IS
  c - match c
  [<set>] - match any character in <set>
  [!]<set>] - match any character not in <set>
  * - match any character(s)
  ^ - match must start at curp

RETURN VALUE
  NULL is returned if expression cannot be found in string; otherwise a pointer to
  the first character in the substring which matches expression is returned. The func­
  tion streexp(3W) can be used to get the pointer to the last character in the match.

SEE ALSO
  strrexp(3W)
  streexp(3W)
NAME
strgetc

SYNOPSIS
#include <buffutil.h>
...
extern int strgetc(sp)
Buffer * sp;

DESCRIPTION
The strgetc function is used to read the next character stored in the string buffer. The function returns the next character in the Buffer. When no characters remain the routine returns EOF.

SEE ALSO
stropen(3W)
strclose(3W)
NAME
  stropen

SYNOPSIS
  #include <buffutil.h>
  ...
  extern Buffer * stropen(string)
  char * string;

DESCRIPTION
  The stropen function copies the string into a newly allocated Buffer. This string
  buffer can be read using the strgetc function and closed using the strclose pro-
  cedure. The strclose function frees the buffer allocated by stropen.

SEE ALSO
  strclose(3W)
  strgetc(3W)
Regular Expression Utilities

NAME

strrexp

SYNOPSIS

```c
#include <expcmp.h>
...
extern char * strrexp(string, curp, expression)
char * string;
char * curp;
char * expression;
```

DESCRIPTION

The `strrexp` function is used to perform a regular expression backward scan of `string` for `expression` starting at `curp`.

THE REGULAR EXPRESSION LANGUAGE USED IS

- `c` - match `c`
- `[<set>]` - match any character in `<set>`
- `![<set>]` - match any character not in `<set>`
- `*` - match any character(s)
- `^` - match must start at `curp`

RETURN VALUE

NULL is returned if expression cannot be found in string; otherwise a pointer to the first character in the substring which matches expression is returned. The function `strexp` can be used to get the pointer to the last character in the match.

SEE ALSO

`strexp(3W)`
`strexp(3W)`
NAME
OlActivateWidget, OlAssociateWidget, OlUnassociateWidget — three routines for activating widget based on widget type

SYNOPSIS
#include <QopenLook.h>

Boolean OlActivateWidget(widget, activation_type, data)
    Widget widget;
    OlVirtualName activation_type;
    XtPointer data;

Boolean OlAssociateWidget(leader, follower, disable_traversal)
    Widget leader;
    Widget follower;
    Boolean disable_traversal;

void OlUnassociateWidget(follower)
    Widget follower;

DESCRIPTION
OlActivateWidget programmatically activates a widget using the specified type. (See the widget manual pages for valid activation types.) The function returns TRUE if the activation type was accepted by the supplied widget or one of its associated followers; otherwise, FALSE is returned. When activating a widget, if the initially-supplied widget does not accept the activation request, OlActivateWidget recursively attempts to activate all associated follower widgets until one of them accepts the activation type.

OlAssociateWidget associates a widget (the follower) with another widget (the leader). Associating a widget with a lead widget effectively expands the number of ways the lead widget can be activated since OlActivateWidget automatically activates any follower widgets if the lead widget does not accept the supplied activation type. This routine returns TRUE if the association was successful; otherwise FALSE is returned. Attempts to create an association-cycle is illegal and produces a warning. It’s typically desirable to prevent keyboard traversal into widgets which are associated with other widgets. The disable_traversal parameter is a convenient interface to setting the follower widget’s XtNtraversalOn resource to FALSE.

OlUnassociateWidget removes a follower widget from a previous association with another lead widget. No warning is generated if the supplied widget was not previously associated with another widget.

NOTES
The above routines accept gadget arguments also.
# Manual Pages: Widgets

## Introduction to the Widgets
- The List of Widget Manual Pages: C-1
- Sample Widget Manual Page: C-2

## Widgets
- Widget Naming Conventions: C-6

---

Table of Contents
Introduction to the Widgets

The List of Widget Manual Pages

The following is a list of all widgets available to the application programmer.

Action Widgets
- OblongButton
- RectButton
- CheckBox
- MenuButton (was: ButtonStack)
- AbbrevMenuButton (was: AbbrevStack)
- Slider
- Gauge
- Scrollbar
- Stub

Text Control Widgets
- StaticText
- EditText
- TextField

Manager Widgets
- BulletinBoard
- ControlArea
- Form
- FooterPanel

Container Widgets
- Caption
- Exclusives
- Nonexclusives
- FlatCheckbox
- FlatExclusives
- FlatNonExclusives
- ScrolledWindow
- ScrollingList

Popup Choices
- Notice
- PopupWindow
- Menu
Sample Widget Manual Page

This section contains the widget manual pages in alphabetical order. The format of the page is described below.
Introduction to the Widgets

Figure C-1: Sample Manual Page Format

 BulletinBoard (3W)                       BulletinBoard (3W)
(Name)

NAME
BulletinBoard

SYNOPSIS
#include <Intrinsics.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <BulletinBo.h>

widget = XtCreateWidget (name, bulletinBoardWidgetClass...);

DESCRIPTION
Simple Composite Widget
The BulletinBoard widget is a composite widget that enforces no ordering on its children. It is up to the application to specify the x- and y-coordinates of each child inserted; otherwise, it will be placed in the upper left corner of the BulletinBoard widget.

SUBSTRUCTURE
Name:
Class:

RESOURCES

<table>
<thead>
<tr>
<th>BulletinBoard Resource Set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Class</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XtNlayout

Range of Values:
OL_MINIMIZE/"minimize"
OL_MAXIMIZE/"maximize"
OL_IGNORE/"ignore"

This resource identifies the layout policy of the BulletinBoard...
Introduction to the Widgets

The numbers in circles refer to the following notes:

1. The top of the first page for each widget page gives the widget name on both the top left and top right of the page, followed by a "3W" in parentheses. (3W is the numbering convention assigned to all OPEN LOOK programming manual pages.)

2. The NAME for the widget is given here. The various ways of referring to the widget, including the name of the C header file, are based on this root name.

3. A synopsis is given that shows, briefly, how to create an instance of the widget.

4. All widgets have a description.

5. The description is typically broken up into minor features, each starting with a separate heading.

6. If the widget is automatically built up of additional widgets, a description of the application and end-user interface to this substructure is given.

7. A complete list of resources available for the widget is given in alphabetical order. The resources are not broken up into their class groupings, to avoid suggesting a particular implementation of the class hierarchy. (Of course, some of the hierarchy is built into the "X Toolkit Intrinsics.")

Many of the resources listed in these tables are part of the superclasses of which all widgets are subclasses; the general resources are described in Section 2 General Resources.

**NOTE**

Do not use a General resource for a widget if the resource is not listed for that particular widget.

The table lists five attributes for each resource:

1. the resource name;
2. the class to which the resource belongs;
3. the type of values handled by the resource;
4. the default value given to the resource by the widget if the application does not set a value or sets an illegal value;

5. the access to the base window allowed:
   I the value can be set at initialization time;
   S the value can be set with a call to XtSetValues;
   G the value can be read with a call to XtGetValues;
   * the value is set in other ways - see the description of the resource for information;
   † the access is conditional - see the description of the resource for information.

As defined in the "X Toolkit Intrinsics," each resource is referred to in resource files by stripping off the XtN prefix from the name for direct reference or the XtC prefix from the class name for reference by class.

8. The resources that are unique to a widget are described after the list of resources.

9. The range of values that each resource can take are listed immediately after the resource name, if the resource can be set or initialized by an application. Exceptions are resources that take pointer values: the pointer cannot be validated, so no range is given.

The values are listed using C language bindings (for example, a variable or macro name like OL_LEFT). If the resource file bindings differ, they follow the C bindings, separated with a slash and enclosed in double quotes (for example, OL_LEFT/"left").
Widgets

Widget Naming Conventions

The following guidelines provide a vehicle by which programmers can create new widgets and organize a collection of widgets into an application. To ensure that applications need not deal with as many styles of capitalization and spelling as the number of widget classes it uses, the following guidelines should be followed when writing new widgets:

- Use the X naming conventions that are applicable. For example, a record component name is all lowercase and uses underscores (_) for compound words (for example, background_pixmap). Type and procedure names start with uppercase and use capitalization for compound words (for example, ArgList or XtSetValues).

- A resource name string is spelled identically to the field name except that compound names use capitalization rather than underscore. To let the compiler catch spelling errors, each resource name should have a macro definition prefixed with XtN. For example, the background_pixmap field has the corresponding resource name identifier XtNbackgroundPixmap, which is defined as the string “backgroundPixmap”. Many predefined names are listed in X11/StringDefs.h. Before you invent a new name, you should make sure that your proposed name is not already defined or that there is not already a name that you can use.

- A resource class string starts with a capital letter and uses capitalization for compound names (for example, ‘BorderWidth’). Each resource class string should have a macro definition prefixed with XtC (for example, XtCBorderWidth).

- A resource representation string is spelled identically to the type name (for example, “TranslationTable”). Each representation string should have a macro definition prefixed with XtR (for example, XtRTranslationTable).

- New widget classes start with a capital and use uppercase for compound words. Given a new class name AbcXyz you should derive several names:
  - Partial widget instance structure name AbcXyzPart
  - Complete widget instance structure names AbcXyzRec and _AbcXyzRec
- Widget instance pointer type name *AbcXyzWidget*
- Partial class structure name *AbcXyzClassPart*
- Complete class structure names *AbcXyzClassRec* and *_AbcXyzClassRec*
- Class structure variable *abcXyzClassRec*
- Class pointer variable *abcXyzWidgetClass*

Action procedures available to translation specifications should follow the same naming conventions as procedures. That is, they start with a capital letter and compound names use uppercase (for example, “Highlight” and “NotifyClient”).
AbbreviatedMenuButton (3W)

NAME
AbbrevMenuButton – a widget that creates a button with a menu mark and menu allowing the user to select items from the menu

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <AbbrevMenu.h>

static Widget stack, menupane, w;
Arg args[1];
stack = XtCreateWidget(name, abbrevMenuButtonWidgetClass, ...);
XtSetArg(args[0], xtNmenuPane, &menupane);
XtGetValues(stack, args, 1);
w = XtCreateWidget(name, widget-class, menupane, ...);

DESCRIPTION
The AbbrevMenuButton widget provides the end user the same features as the MenuButton widget (menu default selection, menu previewing, menu selection), plus current selection viewing, and the ability to add a new selection by typing in its name.

AbbreviatedMenuButton Components
Each abbreviated menu button has a menu. An application typically identifies an additional component, the Current Selection Widget, where previewing of the default menu choice can be done.
Abbreviated Menu Button

Each abbreviated menu button also has the components of a **Menu** widget. These are not shown in Figure 1.

**AbbreviatedMenuButton Sensitive Area**

While on the Abbreviated Menu Button and the power-user option is on:

- Pressing SELECT previews the default menu item (if a preview widget exists), and releasing it will activate the default menu item.
- Clicking SELECT briefly previews the default menu item (if a preview widget exists) and then activates the default menu item.
- Pressing MENU brings up a pop-up menu.
- Clicking MENU produces a stayup menu.

If the power-user option is off:
- Pressing SELECT or MENU brings up a pop-up menu.
- Clicking SELECT or MENU produces a stayup menu.

(The power-user option is set in the Miscellaneous property sheet of the Workspace Manager. See the *OPEN LOOK™ GUI User’s Guide* for more information on setting this option.)
AbbreviatedMenuButton (3W)

All Features of Menu Button Widget
The `AbbrevMenuButton` widget includes all the features of the `MenuButton` widget, except for the previewing (done instead in the Current Selection Widget) and the behavior in a menu (the `AbbrevMenuButton` widget cannot be used in a menu). The features of the `MenuButton` widget apply here.

Current Selection Widget
The Current Selection Widget is created by the application. Typically, the Current Selection Widget and the `AbbrevMenuButton` widget are placed together in a composite widget that manages their side-by-side placement. The `AbbrevMenuButton` widget uses the Current Selection Widget only for previewing the default item in the menu. The application is responsible for using it to implement the OPEN LOOK user interface needs of showing the current menu selection and acquiring a new item to add to the menu, as appropriate.

AbbreviatedMenuButton Coloration
On a monochrome display, the `AbbrevMenuButton` widget indicates that it has input focus by inverting the foreground and background colors of the control.

On color displays, when the `AbbrevMenuButton` widget receives the input focus, the background color is changed to the input focus color set in the `XtNinputFocusColor` resource.

EXCEPTIONS:
— If the input focus color is the same as the Input Window Header Color and the active control is in the window header, then invert the colors.
— If the input focus color is the same as the window background color, then the `AbbrevMenuButton` widget inverts the foreground and background colors when it has input focus.

Figure 2 illustrates the resources that affect the coloration of the `AbbrevMenuButton` widget.
Keyboard Traversal

The default value of the *XtNtraversalOn* resource is True.

The action of the SELECTKEY depends on whether the user has selected the power-user option. (The power-user option is set in the property sheet of the Workspace Manager. See the **GUI User’s Guide** for more information on setting this option.) While on the AbbrevMenuButton and the power-user option is on, clicking the SELECTKEY activates the default menu item. If the power-user option is off, pressing the SELECTKEY posts the stayup menu.

The AbbrevMenuButton does not control the keyboard traversal between the AbbrevMenuButton widget and the Current Selection widget. The Current Selection widget’s traversal resources can be set up to allow for traversal between it and the AbbrevMenuButton, but it is recommended that the *XtNtraversalOn* resource on the Current Selection widget be False. Normal menu traversal can always be used to access the Current Selection widget.

Keyboard traversal within a Menu is done using the PREV_FIELD, NEXT_FIELD, MOVEUP, MOVEDOWN, MOVELEFT, and MOVERIGHT keys. The PREV_FIELD, MOVEUP, and MOVELEFT keys move the input focus to the previous Menu item with keyboard traversal enabled. If the menu is not pinned, the MOVELEFT key will dismiss the menu. If the input focus is on the first item of the Menu, then pressing one of these keys will wrap to the last item of the Menu with keyboard traversal enabled. The NEXT_FIELD, MOVEDOWN, and MOVERIGHT keys move the input focus to the next Menu item with keyboard traversal enabled.
traversal enabled. If the input focus is on the last item of the Menu, then pressing one of these keys will wrap to the first item of the Menu with keyboard traversal enabled. If input focus is on a MenuButton within a Menu, pressing the MENUKEY will post the cascading Menu associated with the MenuButton, and input focus will be on the first Menu item with traversal enabled.

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

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

The DEFAULTACTION key will activate the default control in the AbbrevMenuButton widget as if the user clicked the SELECT button on the control.

The MENDEFALUTKEY can be used by the user to change the default control in the AbbrevMenuButton widget. When the user presses the MENDEFACTKEY, the control which has input focus will become the default control.

<table>
<thead>
<tr>
<th>Abbreviated MenuButton Activation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>OL SELECTKEY</td>
</tr>
<tr>
<td>OL MENUKEY</td>
</tr>
</tbody>
</table>

Display of Keyboard Mnemonic

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

SUBSTRUCTURE

<table>
<thead>
<tr>
<th>Application Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td>*XtNcenter</td>
</tr>
<tr>
<td>*XtNhPad</td>
</tr>
<tr>
<td>*XtNhSpace</td>
</tr>
<tr>
<td>*XtNlayoutType</td>
</tr>
<tr>
<td>*XtNmeasure</td>
</tr>
<tr>
<td>XtNpushpin</td>
</tr>
<tr>
<td>XtNpushpinDefault</td>
</tr>
<tr>
<td>*XtNsameSize</td>
</tr>
<tr>
<td>XtNtitle</td>
</tr>
<tr>
<td>*XtNvPad</td>
</tr>
<tr>
<td>*XtNvSpace</td>
</tr>
</tbody>
</table>

* See the Menu and ControlArea widgets for more information on these resources.
### AbbreviatedMenuButton (3W)

#### RESOURCES

**AbbrevMenuButton** Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmenuPane</td>
<td>XtCMenuPane</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNpreviewWidget</td>
<td>XtCPreviewWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNforeground**

This resource defines the foreground color for the widget.

See the note about the interaction of this resource with other color resources under the description of the **XtNbackground** resource in Core Resources, “Manual Pages: Introduction”, Appendix A.

**XtNmenuPane**

This is the widget where menu items can be attached; its value is available once the **AbbrevMenuButton** widget has been created.

**XtNpreviewWidget**

Range of Values:

(ID of existing widget)

This resource identifies the Current Selection Widget that the **AbbrevMenuButton** can use for previewing the Default Item.

When the end user presses SELECT over the **AbbrevMenuButton** widget, the **AbbrevMenuButton** widget uses the location and size of the Current Selection Widget to display the label of the Default Item. The preview is constrained to be within the height and width of the Current Selection Widget.
If the Current Selection Widget is not defined or is not mapped, previewing does not take place.

NOTE:
The previewing feature is not accessible with keyboard only operation.

SEE ALSO
MenuShell “Programmatic Menu Popup and Popdown”
NAME

BulletinBoard – a simple manager widget

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <BulletinBo.h>
#include <..h>

widget = XtCreateWidget(name, bulletinBoardWidgetClass, ...);
```

DESCRIPTION

Simple Composite Widget

The BulletinBoard widget is a composite widget that enforces no ordering on its children. It is up to the application to specify the x- and y-coordinates of each child inserted; otherwise, it will be placed in the upper left corner of the BulletinBoard widget.

No Children

The BulletinBoard can be mapped with no children. It displays an empty space, possibly surrounded by a border.

BulletinBoard Coloration

Figure 1 illustrates the resources that affect the coloration of the BulletinBoard widget.

Figure 1. Bulletin Board Coloration
The **BulletinBoard** widget is a composite widget and cannot be accessed via keyboard traversal. Input focus moves between the Primitive children of this widget.

### RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SG†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SG†</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SG†</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SG†</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SG</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SG</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtcallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayout</td>
<td>XtCLayout</td>
<td>OlDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>Name</td>
<td>Class</td>
<td>Type</td>
<td>Default</td>
<td>Access</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>------------</td>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCuserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNlayout**

Range of Values:

- **OL_MINIMIZE**/"minimize"
- **OL_MAXIMIZE**/"maximize"
- **OL_IGNORE**/"ignore"

This resource identifies the layout policy the **BulletinBoard** widget is to follow:

**OL_MINIMIZE**

The **BulletinBoard** widget will always be just large enough to contain all its children, regardless of any provided width and height values. Thus the **BulletinBoard** widget will grow and shrink depending on the size needs of its children.

**OL_IGNORE**

The **BulletinBoard** widget will honor its own width and height; it will not grow or shrink in response to the addition, deletion, or altering of its children.

**OL_MAXIMIZE**

The **BulletinBoard** widget will ask for additional space when it needs it for new or altered children, but will not give up extra space.
NAME

Caption – creates a caption or label for any widget

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Caption.h>

widget = XtCreateWidget(name, captionWidgetClass, ...);
```

DESCRIPTION

Caption Components

The Caption composite widget provides a convenient way to label an arbitrary widget. It has only two parts: the Label and the Child Widget.

![Caption Widget Diagram]

Figure 1. Caption Widget

Layout Control

The application can determine how the Label is placed next to the Child Widget (by specifying that it goes above, below, to the left, or to the right), and by specifying how far away the Label is to be placed.

Child Constraints

The Caption composite allows at most one child; attempts to add more than one are refused with a warning. If the Caption widget is mapped without a Child Widget, or if the Child Widget is not managed, only the Label is shown.
Caption Coloration

Figure 2 illustrates the resources that affect the coloration of the Caption widget.

![Diagram of Caption Coloration]

**Keyboard Traversal**

The Caption is a special Manager widget that can be used to display the mnemonic for its single child. However, the label used as a caption to the child is not accessible via keyboard traversal. The Caption has the XtNmnemonic resources which should be set to the child widget's corresponding values. The return of the OlQueryMnemonicDisplay() on the Caption is used to determine if the Caption should display the mnemonic for the child.

The action of a mnemonic on a Caption widget is used for traversal only since clicking the SELECT button on a caption does not have an affect.

**Display of Keyboard Mnemonic**

The Caption widget displays the mnemonic for its child as part of its label. If the mnemonic character is in the label, then that character is highlighted according to the value of the application resource XtNshowMneumonics(). If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the application resource XtNshowMneumonics().

If truncation is necessary, the mnemonic displayed in parenthesis is truncated as a unit.
Display of Keyboard Accelerators

The Caption widget displays the keyboard accelerator for its child as part of its label. The string in the XtNacceleratorText resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNalignment</td>
<td>XtCAlignment</td>
<td>OI Define</td>
<td>OL_CENTER</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>*</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SG</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct*</td>
<td>(OPEN LOOK font)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black*</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td>OI FontList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>'0'</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNposition</td>
<td>XtCPosition</td>
<td>OI Define</td>
<td>OL_LEFT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNsce</td>
<td>XtCSpace</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTRaversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

XtNalignment

Range of Values:

If XtNposition is OL_LEFT or OL_RIGHT:

OL_TOP/"top"
OL_CENTER/"center"
OL_BOTTOM/"bottom"

If XtNposition is OL_TOP or OL_BOTTOM:

OL_LEFT/"left"
OL_CENTER/"center"
OL_RIGHT/"right"

This specifies how the Label is to be aligned relative to the Child Widget, as described below:
The left edge of the Label is aligned with the left edge of the Child Widget.

The top edge of the Label is aligned with the top edge of the Child Widget.

The center of the Label is aligned with the center of the Child Widget.

The right edge of the Label is aligned with the right edge of the Child Widget.

The bottom edge of the Label is aligned with the bottom edge of the Child Widget.

**XtNlabel**

This resource gives the string to use for the Label. If NULL, the size of the Caption widget will be identical to the size of the child widget.

Note that the Label is displayed as given; no punctuation (such as a colon) is added.

Control characters (other than spaces) are ignored without warning. For example, embedded newlines do not cause line breaks.

**XtNposition**

Range of Values:

- **OL_LEFT/** "left"
- **OL_RIGHT/** "right"
- **OL_TOP/** "top"
- **OL_BOTTOM/** "bottom"

This resource determines on which side of the Child Widget the Label is to be placed. The value may be one of **OL_LEFT**, **OL_RIGHT**, **OL_TOP**, or **OL_BOTTOM** to indicate that the Label is to be placed to the left, to the right, above, or below the Child Widget, respectively.

**XtNspace**

Range of Values:

\[ 0 \leq \text{XtNspace} \]

This resource gives the separation of the Label from the child widget, in pixels, as suggested by Figure 3.
Figure 3. Label and Child Widget Spacing
NAME
CheckBox – creates a label button with a check box to act as a toggle switch

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <CheckBox.h>

    widget = XtCreateWidget(name, checkBoxWidgetClass, ...);

DESCRIPTION
CheckBox Components
The CheckBox widget implements one of the OPEN LOOK button widgets. It
consists of a Label next to a Check Box; the Check Box will have a Check Mark, if
selected.

![CheckBox Widget Diagram]

Figure 1. CheckBox Widget
Figure 2 shows several buttons, in unselected and selected, as well as normal and dim states.

Typical Use of Check Boxes
Check Boxes may be used alone, but are usually used in the Nonexclusives composite widget, where they are used to implement a several-of-many selection. Making the CheckBox widget a child of a different composite widget will not produce an error, but proper layout is not guaranteed.

Operating on Check Boxes
A CheckBox widget has two states: "set" and "not set". When set, the Check Mark is visible. Toggling this state alternates a resource (XtNset) between "true" and "false" and starts an action associated with the check box. Clicking SELECT on a check box toggles the state associated with it. Pressing SELECT, or moving the pointer into the check box while SELECT is pressed, adds or removes the Check Mark to reflect the state the check box would be in if SELECT was released. Releasing SELECT toggles the state. Moving the pointer off the check box before releasing SELECT restores the original CheckBox, but does not toggle the state. Clicking or pressing MENU does not do anything in the CheckBox widget; the event is passed up to an ancestor widget.

Bounds on SELECT
Only the CheckBox box and Check Mark respond to SELECT, as shown in Figure 3.
On a monochrome display, the CheckBox widget indicates that it has input focus by inverting the foreground color and parent's background colors within the bounding box of the widget.

On color displays, when the CheckBox widget receives the input focus, the background color within the bounding box of the widget is changed to the input focus color set in the `XtNinputFocusColor` resource. When the CheckBox widget loses the input focus, the background color reverts to its parent's `XtNbackground` color or `XtNbackgroundPixmap`.

**Exceptions:**

- If the input focus color is the same as the parent's background color, then the CheckBox widget inverts the foreground and background colors when it has input focus.

- If the input focus color is the same as the font color or foreground color, then the CheckBox widget inverts the foreground and background colors when it has input focus.

Figure 4 illustrates the resources that affect the coloration of the `CheckBox` widget.
The default value of the `XtNtraversalOn` resource is `TRUE`.

The CheckBox widget responds to the following keyboard navigation keys:

- **NEXT FIELD**: moves to the next traversable widget in the window
- **PREV_FIELD**: moves to the previous traversable widget in the window
- **MOVEUP**: moves to the CheckBox above the current widget in the Nonexclusives composite
- **MOVEDOWN**: moves to the CheckBox below the current widget in the Nonexclusives composite
- **MOVELEFT**: moves to the CheckBox to the left of the current widget in the Nonexclusives composite
- **MOVERIGHT**: moves to the CheckBox to the right of the current widget in the Nonexclusives composite
- **NEXTWINDOW**: moves to the next window in the application
- **PREVWINDOW**: moves to the previous window in the application
NEXTAPP moves to the first window in the next application
PREVAPP moves to the first window in the previous application

<table>
<thead>
<tr>
<th>CheckBox Widget Activation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>OL_SELECTKEY</td>
</tr>
</tbody>
</table>

**Display of Keyboard Mnemonic**

The CheckBox widget displays the mnemonic accelerator as part of its label. If the mnemonic character is in the label, then that character is highlighted according to the value returned by OIQueryMnemonicDisplay(). If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value returned by OIQueryMnemonicDisplay().

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

**Display of Keyboard Accelerators**

The CheckBox widget displays the keyboard accelerator as part of its label. The string in the XtNacceleratorText resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

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

**RESOURCES**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdim</td>
<td>XtCDim</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct</td>
<td>(OPEN LOOK font)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>BLACK*</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td>XFontList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>RED</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>(class name)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelImage</td>
<td>XtCLabelImage</td>
<td>XImage</td>
<td>(class name)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelJustify</td>
<td>XtCLabelJustify</td>
<td>XImage</td>
<td>OL_LEFT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelTile</td>
<td>XtCLabelTile</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
</tbody>
</table>
### CheckBox Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNlabelType</td>
<td>XtCLabelType</td>
<td>int</td>
<td>OL_STRING</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNposition</td>
<td>XtCPosition</td>
<td>OlDefine</td>
<td>OL_LEFT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNselect</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNset</td>
<td>XtCSet</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNunselect</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

### XtNdim

Range of Values:

- TRUE
- FALSE

If this resource is TRUE, the check box border is dimmed to show that the check box represents the state of one or more of several objects that, as a group, are in different states.

### XtNlabel

This resource is a pointer to the text for the Label. This resource is ignored if the XtNlabelType resource has the value OL_IMAGE.

### XtNlabelImage

This resource is a pointer to the image for the Label of the CheckBox widget. This resource is ignored unless the XtNlabelType resource has the value OL_IMAGE.

If the image is smaller than the space available for it next to the Check Box, it is centered vertically and either centered or left-justified horizontally depending on the value of the XtNlabelJustify resource. If the image is larger than the space available for it, it is clipped so that it does not stray outside the space.

### XtNlabelJustify

Range of Values:

- OL_LEFT/"left"
- OL_RIGHT/"right"
This resource dictates whether the Label should be left- or right-justified within the space left before or after the Check Box, if the XtNwidth resource gives more space than needed.

XtNlabelTile
Range of Values:
TRUE
FALSE

This resource augments the XtNlabelImage/XtNlabelPixmap resource to allow tiling of the sub-object's background. For an image/pixmap that is smaller than the sub-object's background, the label area is tiled with the image/pixmap to fill the sub-object's background if this resource is TRUE; otherwise, the label is placed as described by the XtNlabelImage resource.

The XtNlabelTile resource is ignored for text labels.

XtNlabelType
Range of Values:
OL_STRING/"string"
OL_IMAGE/"image"

This resource identifies the form that the Label takes. It can have the value OL_STRING or OL_IMAGE for text or image, respectively.

XtNposition
Range of Values:
OL_LEFT/"left"
OL_RIGHT/"right"

This resource determines on which side of the Check Box the Label is to be placed. The value may be one of OL_LEFT or OL_RIGHT to indicate that the Label is to be placed to the left or to the right of the Check Box, respectively.

XtNrecomputeSize
Range of Values:
TRUE
FALSE

This resource indicates whether the CheckBox widget should calculate its size and automatically set the XtNheight and XtNwidth resources. If set to TRUE, the CheckBox widget will do normal size calculations that may cause its geometry to change. If set to FALSE, the CheckBox widget will leave its size alone; this may cause truncation of the visible image being shown by the CheckBox widget if the fixed size is too small, or may cause padding if the fixed size is too large. The location of the padding is determined by the XtNlabelJustify resource.

XtNselect
This is the list of callbacks invoked when the widget is selected.

XtNset
Range of Values:
TRUE
FALSE
This resource reflects the current state of the check box. The Check Mark is present if `XtNset` is TRUE and is absent otherwise.

**XtNunselect**
This is the list of callbacks invoked when a `CheckBox` widget is toggled into the "unset" mode by the end user to make `XtNset` be FALSE. Note that simply setting `XtNset` to FALSE with a call to `XtSetValues()` does not issue the `XtNunselect` callbacks.

**XtNdim and XtNset Interaction**
The `XtNdim` and `XtNset` resources can be set independently, as the state table in Figure 5 shows.

<table>
<thead>
<tr>
<th>XtNset</th>
<th>XtNdim</th>
<th>Check Box Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>✓</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>✓</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. Check Box Appearance with Set/Default/Dim

**Label and Check Box Appearance**
The `XtNwidth`, `XtNheight`, `XtNrecomputeSize`, and `XtNlabelJustify` resources interact to produce a truncated, clipped, centered, left-justified, or right-justified
Label and Check Box as shown in Figure 6.

<table>
<thead>
<tr>
<th><strong>XtNwidth</strong></th>
<th><strong>XtNrecomputeSize</strong></th>
<th><strong>XtNLabelJustify</strong></th>
<th><strong>Result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>any value</td>
<td>TRUE</td>
<td>any</td>
<td>Just Fits</td>
</tr>
<tr>
<td>needed for label</td>
<td>FALSE</td>
<td>OL-LEFT</td>
<td>Left Justified</td>
</tr>
<tr>
<td>needed for label</td>
<td>FALSE</td>
<td>OL-RIGHT</td>
<td>Right Justified</td>
</tr>
<tr>
<td>needed for label</td>
<td>FALSE</td>
<td>any</td>
<td>Truncated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>XtNheight</strong></th>
<th><strong>XtNrecomputeSize</strong></th>
<th><strong>XtNLabelJustify</strong></th>
<th><strong>Result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>any value</td>
<td>TRUE</td>
<td>any</td>
<td>Just Fits</td>
</tr>
<tr>
<td>needed for label</td>
<td>FALSE</td>
<td>any</td>
<td>Centered</td>
</tr>
<tr>
<td>needed for label</td>
<td>FALSE</td>
<td>any</td>
<td>Clipped</td>
</tr>
</tbody>
</table>

Figure 6. Label and Check Box Appearance

When the label is left-justified, right-justified, or centered the extra space is filled with the background color of the CheckBox widget’s parent, as determined by the XtNbackground and XtNbackgroundPixmap resources of the parent.

See also the XtNlabelTile resource for how it affects the appearance of a label.
NAME

ControlArea – manages a number of child widgets in rows or columns

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <ControlAre.h>

widget = XtCreateWidget(name, controlAreaWidgetClass, ...);
```

DESCRIPTION

ControlArea Components

The ControlArea widget has zero or more Child Widgets and an optional Border.

Layout Control

The ControlArea composite widget arranges its child widgets, presenting them to the end-user as a group of "controls". The application can choose one of four simple layout schemes: Fixed number of columns in the control pane, fixed number of rows, fixed overall width of the control area, and fixed overall height. The application can also specify the inter-control spacing and the size of the margin around the children.

Equal Height Rows

The children in each row align at the top of the row. The distance between the top of one row and the next is the height of the tallest control in the row plus the application specified inter-row spacing.

ControlArea Coloration

Figure 1 illustrates the resources that affect the coloration of the ControlArea widget.
RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNalignCaptions</td>
<td>XtCAlignCaptions</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCbackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcenter</td>
<td>XtCCenter</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNhPad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhSpace</td>
<td>XtCHSpace</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OIDefine</td>
<td>OL_FIXEDROWS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
</tbody>
</table>
ControlArea Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNsameSize</td>
<td>XtCSameSize</td>
<td>OLDefine</td>
<td>OL_COLUMNS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI+</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvSpace</td>
<td>XtCVSpace</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNalignCaptions**

Range of Values:
- TRUE
- FALSE

This resource controls how the ControlArea widget aligns widgets of the Caption class. If set to TRUE, the ControlArea will align all Caption widgets in each column so that their captions are right justified. This may affect the width calculation for a column: The effective width for the Caption widgets in a column becomes the sum of the width of the widest caption, plus the largest caption/child widget separation and child widget width.

This alignment is only for groups of Caption widgets with all their captions on the left or the right. Mixed orientation, or captions above or below, cannot be aligned well.
If the `XtNalignCaption` resource is set to FALSE, the `ControlArea` will align all `Caption` widgets the same as other widgets—by their overall width. This resource takes precedence over the `XtNcenter` resource, but only for `Caption` widgets.

**XtNcenter**
Range of Values:

- TRUE
- FALSE

This resource controls how the `ControlArea` widget orients each widget within a column (although see also `XtNalignCaptions`). If set to TRUE, the `ControlArea` will center each widget with each column; if set to FALSE, the `ControlArea` will left justify each widget within each column, unless the `XtNalignCaptions` resource is TRUE.

**XtNhPad**
**XtNvPad**
Range of Values:

\[
0 \leq XtnhPad \\
0 \leq XtnvPad
\]

These resources give the amount of padding, in pixels, to leave around the edges of the control area, left and right, and top and bottom, respectively, as suggested by Figure 3.
Figure 3. Padding Around Controls

XtNhSpace
XtNvSpace

Range of Values:

\[ 0 \leq \text{XtNhSpace} \]
\[ 0 \leq \text{XtNvSpace} \]

These resources give the amount of space, in pixels, to leave between controls horizontally and vertically, respectively. If the controls are of different sizes in a row or column, the spacing applies to the widest or tallest dimension of all the controls.
This resource controls the layout of the child widgets by the ControlArea composite. The choices are to specify the number of rows or columns, or to specify the overall height or width of the layout area. Only one of these dimensions can be specified directly; the other is determined by the number of controls added. For instance, if the application specifies that the control area should have four columns, the number of rows will be the number of controls divided by four.

The values of the XtNlayoutType resource can be:

- **OL_FIXEDROWS** if the layout should have a fixed number of rows and enough columns to hold all the controls;
- **OL_FIXEDCOLS** if the layout should have a fixed number of columns and enough rows to hold all the controls;
- **OL_FIXEDWIDTH** if the layout should be of a fixed width but tall enough to hold all the controls;
**OL_FIXEDHEIGHT** if the layout should be of a fixed height but wide enough to hold all the controls.

The **XtNmeasure** resource gives the number of rows or columns or the fixed height or width.

**XtNmeasure**

Range of Values:

0 < XtNmeasure

Default:

- If XtNlayoutType is OL_FIXEDROWS or OL_FIXEDCOLS: 1
- If XtNlayoutType is OL_FIXEDWIDTH or OL_FIXEDHEIGHT:
  width or height of widest or tallest widget, depending on XtNlayoutType.

This resource gives the number of rows or columns in the layout of the child widgets, or the fixed width or height of the control area. When XtNlayoutType is OL_FIXEDWIDTH or OL_FIXEDHEIGHT, the measure includes the padding on both edges and the inter-control spacing, as suggested by Figure 5.

---

**Figure 5. XtNmeasure**

**XtNsamesize**

Range of Values:

- OL_NONE/"none"
- OL_COLUMNS/"columns"
- OL_ALL/"all"
This resource controls which controls, if any, are forced to be the same width within the ControlArea widget:

**OL_NONE**  The controls are placed in fixed-width columns, but the size of each control is left alone. The width of each column is the width of the widest control in the column.

**OL_COLUMNS**  Controls of the same class in each column are made the same width as the widest of them. The width of each column is thus the width of the widest control in the column.

**OL_ALL**  All controls are made the same width, the width of the widest control in the ControlArea widget.
NAME

Exclusives – allows the end-user to select one of a set of choices

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Exclusives.h>

widget = XtCreateWidget(name, exclusivesWidgetClass, ...
```

DESCRIPTION

The Exclusives widget provides a simple way to build a one-of-many button selection object. It manages a set of rectangular buttons, providing layout management.

Grid Layout and Button Labels

The Exclusives widget lays out the rectangular buttons in a grid in the order they are added as child widgets by the application. The number of rows or columns in this grid can be controlled by the application. If the grid has more than one row, the Exclusives widget forces the buttons in each column to be the same size as the widest in the column, and forces their labels to be left-justified. (Note: If the grid has a single row, each button will be only as wide as necessary to display the label.)

![Figure 1. Example of Exclusive Buttons](image-url)
Exclusives (3W)

Selection Control—One Set
In one mode of operation (for example, XtNNoneSet is FALSE), exactly one button in an Exclusives widget must be "set" (for example, the XtNSet resource set to TRUE). An error is generated if an Exclusives is configured with two or more rectangular buttons set or with no button set. The Exclusives widget maintains this condition by ensuring that when a button is set by the user clicking SELECT over it, the button that was set is cleared and its XtNUnselect callbacks are invoked. However, clicking SELECT over a button that was already set does nothing.

Selection Control—None Set
In the other mode of operation (for example, XtNNoneSet is TRUE), at most one button in an Exclusives widget can be set. An error is generated if an Exclusives is configured with two or more rectangular buttons set, but not if configured with no button set. The Exclusives widget maintains this condition by ensuring that when a button is set by the user clicking SELECT over it, or by the application programmer with the XtNSet resource, any button that was previously set is cleared. Also, clicking SELECT over a button that was already set will unset it. Clearing a button in either case invokes its XtNUnselect callbacks.

Use in a Menu
The Exclusives widget can be added as a single child to a menu pane to implement a one-of-many menu choice.

Child Constraint
The Exclusives widget constrains its child widgets to be of the class rectButtonWidgetClass.

Exclusives Coloration
There is no explicit foreground or background in the Exclusives composite widget; each rectangular button has its own coloration.

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

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

The SELECTKEY acts as if the SELECT button had been clicked on the RectButton with input focus. The MENUKEY acts as if the MENU button had been clicked on the RectButton with input focus.
RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OlDefine</td>
<td>OL_FIXEDROWS</td>
<td>SI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SI</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>SI</td>
</tr>
<tr>
<td>XtNnoneSet</td>
<td>XtCNoneSet</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SI</td>
</tr>
</tbody>
</table>

**XtNlayoutType**

Range of Values:

- OL_FIXEDROWS/"fixedrows"
- OL_FIXEDCOLS/"fixedcols"

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

The values of the **XtNlayoutType** resource can be

- OL_FIXEDROWS if the layout should have a fixed number of rows;
- OL_FIXEDCOLS if the layout should have a fixed number of columns.

**XtNmeasure**

Range of Values:

0 < **XtNmeasure**

This resource gives the number of rows or columns in the layout of the child widgets. If there are not enough child widgets to fill a row or column, the remaining space is left blank. If there is only one row (column), and it is not filled with child widgets, the remaining “space” is of zero width (height).
XtNnoneSet
Range of Values:
TRUE
FALSE

This resource controls whether the buttons controlled by the Exclusives composite can be toggled into an unset mode directly. If set to FALSE, at all times exactly one button must be set. Attempting to select the currently set button does nothing. If set to TRUE, at all times no more than one button can be set. However, the user can select the currently set button again to toggle it back into an unset mode.
NAME

FlatCheckBox is a performance improvement over the NonExclusives widget that is populated with CheckBox widgets.

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OPenLook.h>
#include <FCheckBox.h>

widget = XtCreateWidget(name, flatCheckBoxWidgetClass, ...);
```

DESCRIPTION

Keyboard Traversal

The FlatCheckbox widget is a Primitive widget that manages the traversal between a set of sub-objects. When the user traverses to a FlatCheckbox widget, the first sub-object in the set will display itself as having input focus (see the CheckBox Widget for a description of this appearance.) The MOVEUP, MOVE.DOWN, MOVERIGHT, and MOVELEFT keys move the input focus between the sub-objects. To traverse out of the FlatCheckbox widget, the following keys can be used:

- NEXT_FIELD moves to the next traversable widget in the window.
- PREV_FIELD moves to the previous traversable widget in the window.
- NEXTWINDOW moves to the next window in the application.
- PREVWINDOW moves to the previous window in the application.
- NEXTAPP moves to the first window in the next application.
- PREVAPP moves to the first window in the previous application.

Keyboard Operation

<table>
<thead>
<tr>
<th>Activation Type</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL_SELECTKEY</td>
<td>These controls have two states: &quot;set&quot; and &quot;not set&quot;. (When set, its border is thickened.) Pressing the SELECTKEY while a flat checkbox item has focus will toggle the checkbox’s current state. If the control is &quot;set&quot;, then toggling the control will call the XtNunselect callback list. If the control is &quot;not set&quot;, then toggling the control will call the XtNselect callback list.</td>
</tr>
</tbody>
</table>

Display of Keyboard Mnemonic

The FlatCheckbox widget displays the mnemonic accelerator of a sub-object as part of the sub-object’s label. If the mnemonic character is in the label, then that character is highlighted according to the value of the application resource XtNshowMneumonics(). If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the application resource XtNshowMneumonics().
If truncation is necessary, the mnemonic displayed in parenthesis is truncated as a unit.

**Display of Keyboard Accelerators**

The `FlatCheckbox` widget displays the keyboard accelerator as part of the sub-object's label. The string in the `XtNacceleratorText` resource is displayed to the right of the label (or mnemonic) separated by at least one space. The accelerator-Text is right justified.

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

![Diagram of Flat CheckBox Item](image)

**Figure 1. Flat CheckBox Item**

**FlatCheckbox Coloration**

The `FlatCheckbox` container inherits its background color from the container’s parent widget. Setting the background color affects only the sub-objects' background.

On a monochrome display, the FlatCheckbox widget indicates that it has input focus by inverting the foreground color and parent’s background colors within the bounding box of the first sub-object.

On color displays, when the FlatCheckbox widget receives the input focus, the background color within the bounding box of the first sub-object is changed to the input focus color set in the `XtNinputFocusColor` resource. When the FlatCheckbox sub-object loses the input focus, the background color reverts to its parent’s `XtNbackground` color or `XtNbackgroundPixmap`. 
FlatCheckBox (3W)

EXCEPTIONS:

- If the input focus color is the same as the parent’s background color, then the FlatCheckbox widget inverts the foreground and background colors of the sub-object when it has input focus.

- If the input focus color is the same as the font color or foreground color, then the FlatCheckbox widget inverts the foreground and background colors of the sub-object when it has input focus.

RESOURCES

The following table lists the resources for the FlatCheckBox. Resources that have a bullet (•) in the Access column denote sub-object resources. If these resources are not included in the XtNitemFields list, they are inherited from the container widget. An application can change the default values for sub-object resources by setting them directly on the container. Even though a sub-object resource is not included in the XtNitemFields list, the application can query the value of any sub-object resource with OlFlatGetValues().

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G•</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>None</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNclientData</td>
<td>XtCClientData</td>
<td>XiPointer</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>Cardinal</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct*</td>
<td>(OPEN LOOK font)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgravity</td>
<td>XtCGravity</td>
<td>int</td>
<td>CenterGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNHpad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhspace</td>
<td>XtCHSpace</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNitemFields</td>
<td>XtCItemFields</td>
<td>String*</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNitemGravity</td>
<td>XtCItemGravity</td>
<td>int</td>
<td>NorthWestGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMaxHeight</td>
<td>XtCItemMaxHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMaxWidth</td>
<td>XtCItemMaxWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinHeight</td>
<td>XtCItemMinHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinWidth</td>
<td>XtCItemMinWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitems</td>
<td>XtCItems</td>
<td>XiPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemsTouched</td>
<td>XtCItemsTouched</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
</tbody>
</table>
### Flat CheckBox Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlabelImage</td>
<td>XtCLabelImage</td>
<td>XImage*</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlabelJustify</td>
<td>XtCLabelJustify</td>
<td>OIDefine</td>
<td>OL_LEFT</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlabelTile</td>
<td>XtCLabelTile</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlayoutHeight</td>
<td>XtLayoutHeight</td>
<td>OIDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtLayoutType</td>
<td>OIDefine</td>
<td>OL_FIXEDROWS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutWidth</td>
<td>XtLayoutWidth</td>
<td>OIDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmanaged</td>
<td>XtCManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNnumItemFields</td>
<td>XtCNumItemFields</td>
<td>Cardinal</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNnumItems</td>
<td>XtCNumItems</td>
<td>Cardinal</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNposition</td>
<td>XtCPosition</td>
<td>OIDefine</td>
<td>OL_LEFT</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameHeight</td>
<td>XtCSameHeight</td>
<td>OIDefine</td>
<td>OL_ALL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameWidth</td>
<td>XtCSameWidth</td>
<td>OIDefine</td>
<td>OL_COLUMNS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNselectProc</td>
<td>XtCallbackProc</td>
<td>XtCallbackProc</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNset</td>
<td>XtCSet</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI+i</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNunselectProc</td>
<td>XtCallbackProc</td>
<td>XtCallbackProc</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvSpace</td>
<td>XtCVSpace</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNancestorSensitive**

Range of Values:

- **TRUE**/"true"
- **FALSE**/"false"

This resource indicates the sensitivity of the sub-object's ancestors. If TRUE, all the sub-object's ancestors are sensitive and the sub-object is sensitive to user input. If FALSE, one or more of the sub-object's ancestors are insensitive, so the sub-object displays an inactive visual and is not sensitive to user input.

**XtNbackground**

This is the pixel color used to fill in the background of the check box.
**XtNbackgroundPixmap**
This resource specifies the pixmap that is displayed as the sub-object’s label. Any supplied pixmap must have the same depth as the flat widget’s depth. Pixmaps of None and ParentRelative are not considered valid values. If either XtNlabel or XtNlabelImage has a non-NULL value, this resource is ignored.

**XtNclientData**
This is the client data supplied to all callback procedures.

**XtNitems**
This is the list of sub-object items. This value must point to a static list since flat containers reference this list after initialization but do not cache its information.

**XtNitemFields**
This is the list of resource names used to parse the records in the XtNitems list. This resource does not have to point to static information since the flat container does not use this information after initialization. Though the flat container does not reference this resource’s value after initialization, it holds onto it for responding to an XtGetValue() request and supplying it in the OlFlatCallData structure during callbacks. Therefore, if the application plans on querying this resource, it’s recommended that the application make this resource point to static information.

**XtNitemsTouched**
Range of Values:

```plaintext
TRUE
FALSE
```
Whenever the application modifies an item list directly, it must supply this resource (with a value of TRUE) to the flat widget container so that the container can update the visual. If the resource value is supplied, the flat widget container treats its current item list as a new list and hence, updates its entire visual. Since the list is treated as a new list, the flat container may request a change in geometry from its parent.

**Note:**
It is not necessary to use this resource if the application modifies the list with the OlFlatSetValues procedure; nor is it necessary to use this resource whenever the application supplies a new list to the flat container.

**XtNlabel**
This is the text string that appears in the sub-object.

**XtNlabelImage**
This is an XImage pointer that can appear in a sub-object. This resource is ignored if XtNlabel is non-NULL.

**XtNlabelJustify**
Range of Values:

```plaintext
OL_LEFT/"left"
OL_CENTER/"center"
OL_RIGHT/"right"
```
This resource specifies the justification of the label or XImage that appears within a sub-object.
**XtNlabelTile**
Range of Values:

TRUE
FALSE

This resource augments the **XtNlabelImage/XtNlabelPixmap** resource to allow tiling of the sub-object's background. For an image/pixmap that is smaller than the sub-object's background, the label area is tiled with the image/pixmap to fill the sub-object's background if this resource is TRUE; otherwise, the label is placed as described by the **XtNlabelJustify** resource.

The **XtNlabelTile** resource is ignored for text labels.

**XtNmappedWhenManaged**
Range of Values:

TRUE/"true"
FALSE/"false"

This resource specifies whether or not a managed sub-object is displayed. Regardless of this resource's value, all managed sub-objects will be included when determining the layout.

**Note:**
This resource is never inherited from the container, so its default value is always TRUE.

**XtNnumItems**
This resource specifies the number of sub-object items.

**XtNnumItemFields**
This resource indicates the number of resource names contained in **XtNitemFields**.

**XtNposition**
Range of Values:

OL_LEFT/"left"
OL_RIGHT/"right"

This resource determines on which side of the check box the label is to be placed. The value of OL_LEFT or OL_RIGHT indicates the label is placed to the left or to the right of the check box, respectively.

**XtNsameHeight**
Range of Values:

OL_ALL/"all"
OL_ROWS/"rows"
OL_NONE/"none"

This resource specifies the rows that are forced to the same height.
**XtNsameWidth**
Range of Values:

- OL_ALL/"all"
- OL_COLUMNS/"columns"
- OL_NONE/"none"

This resource specifies the columns that are forced to the same width.

**XtNselectProc**
This callback procedure is called whenever the sub-object becomes selected by user input.

**XtNsensitive**
Range of Values:

- TRUE/"true"
- FALSE/"false"

If TRUE, the sub-object is sensitive to user input. If FALSE, the sub-object is insensitive to user input and an inactive visual is displayed to indicate this state.

**Note:**
This resource is never inherited from the container, so its default value is always TRUE.

**XtNset**
Range of Values:

- TRUE/"true"
- FALSE/"false"

This resource reflects the current state of the sub-object.

**Note:**
This resource is never inherited from the container, so its default value is always FALSE.

Even if the application does not use XtNset in its item fields list, the container will correctly maintain the set item and the application can change the set item via OlFlatSetValues.

**XtNunselectProc**
This callback procedure is called whenever the sub-object becomes unselected by user input.
NAME

FlatExclusives – allows the user to select one of a series of choices

SYNOPSIS

```
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <FExclusive.h>

widget = XtCreateWidget(name, flatExclusivesWidgetClass, ...);
```

DESCRIPTION

![Figure 1. Flat Exclusives Widget](image)

**Selection Control - One Set**

In one mode operation (i.e., ` XtNnoneSet` is FALSE), exactly one sub-object in a FlatExclusives widget must be "set," i.e., the `XtNset` resource is TRUE for one of the sub-objects. A warning is generated if two or more sub-objects are set. If no items are set, the FlatExclusives makes the first sub-object that is both managed and mapped when managed be the set item. No warning is produced in this case. The FlatExclusives maintains this condition by ensuring that when a sub-object is set by the user clicking SELECT over it, the sub-object that was set is cleared and its `XtNunselectProc` procedure is called and the sub-object under the pointer is made to be set and its `XtNselectProc` procedure is
called. However, clicking SELECT over a sub-object that is already set does nothing.

Selection Control - None Set
In the other mode of operation (i.e., XtNnoneset is TRUE), at most one sub-object in a FlatExclusives widget can be "set." A warning is generated if two or more sub-objects are set. The FlatExclusives maintains this condition by ensuring that when a sub-object is set by the user clicking SELECT over it, the sub-object that was set is cleared and its XtNunselectProc procedure is called and the sub-object under the pointer is made to be set and its XtNselectProc procedure is called. Clicking SELECT over a sub-object that is already set clears it and its XtNunselectProc procedure is called.

Use in a Menu
The FlatExclusives widget can be added as child in a menu pane to implement a one-of-many menu choice.

FlatExclusives Coloration
The FlatExclusives container inherits its background color from the container's parent widget. Setting the background color affects only the sub-objects' background.

Keyboard Traversal
The FlatExclusives widget is a Primitive widget that manages the traversal between a set of sub-objects. When the user traverses to a FlatExclusives widget, the first sub-object in the set will display itself as having input focus (see the RectButton Widget for a description of this appearance.) The MOVEUP, MOVEDOWN, MOVERIGHT, and MOVELEFT keys move the input focus between the sub-objects. To traverse out of the FlatExclusives widget, the following keys can be used:

- NEXT_FIELD moves to the next traversable widget in the window
- PREV_FIELD moves to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application.
- PREVWINDOW moves to the previous window in the application.
- NEXTAPP moves to the first window in the next application.
- PREVAPP moves to the first window in the previous application.

Keyboard Operation

<table>
<thead>
<tr>
<th>Flat Exclusives Activation Types</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
<td></td>
</tr>
<tr>
<td>OL_MENUDEFAULTKEY</td>
<td>If the FlatExclusives is on a menu, this command will set the item with focus to be menu's default; otherwise, this command is ignored.</td>
</tr>
<tr>
<td>OL_SELECTKEY</td>
<td>This command acts as if the SELECT mouse button had been clicked on the RectButton with focus. See &quot;Selection Control&quot; sections above.</td>
</tr>
</tbody>
</table>
Display of Keyboard Mnemonic
The FlatExclusives widget displays the mnemonic accelerator of a sub-object as part of the sub-object's label. If the mnemonic character is in the label, then that character is highlighted according to the value of the application resource XtNshowMneumonics(). If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the application resource XtNshowMneumonics().

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

Display of Keyboard Accelerators
The FlatExclusives widget displays the keyboard accelerator as part of the sub-object's label. The string in the XtNacceleratorText resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

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

RESOURCES
The following table lists the resources for the FlatExclusives. Resources that have a bullet (•) in the Access column denote sub-object resources. If these resources are not included in the XtNitemFields list, they are inherited from the container widget. An application can change the default values for sub-object resources by setting them on the container directly. Even though a sub-object resource is not included in the XtNitemFields list, the application can query the value of any sub-object resource with OlFlatGetValues().

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G•</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>None</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNclientData</td>
<td>XtCClientData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdefault</td>
<td>XtCDefault</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>Cardinal</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdim</td>
<td>XtCDim</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct</td>
<td>(OPEN LOOK font)</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgravity</td>
<td>XtCGravity</td>
<td>int</td>
<td>CenterGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI•</td>
</tr>
</tbody>
</table>
### Flat Exclusives Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNhPad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemFields</td>
<td>XtCItemFields</td>
<td>String *</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNitemMaxHeight</td>
<td>XtCItemMaxHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMaxWidth</td>
<td>XtCItemMaxWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinHeight</td>
<td>XtCItemMinHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinWidth</td>
<td>XtCItemMinWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitems</td>
<td>XtItems</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemsTouched</td>
<td>XtItemsTouched</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SG</td>
</tr>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlabelImage</td>
<td>XtCLabelImage</td>
<td>XImage *</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlabelJustify</td>
<td>XtCLabelJustify</td>
<td>OlDefine</td>
<td>OL_LEFT</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlabelTile</td>
<td>XtCLabelTile</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNlayoutHeight</td>
<td>XtCLayoutHeight</td>
<td>OlDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OlDefine</td>
<td>OL_FIXEDROWS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutWidth</td>
<td>XtCLayoutWidth</td>
<td>OlDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmanaged</td>
<td>XtCManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmemonic</td>
<td>XtCNemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNnoneSet</td>
<td>XtCNoneSet</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNnumItemFields</td>
<td>XtCNumItemFields</td>
<td>Cardinal</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNnumItems</td>
<td>XtCNumItems</td>
<td>Cardinal</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameHeight</td>
<td>XtCSameHeight</td>
<td>OlDefine</td>
<td>OL_ALL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameWidth</td>
<td>XtCSameWidth</td>
<td>OlDefine</td>
<td>OL_COLUMNS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNselectProc</td>
<td>XtCCallbackProc</td>
<td>XtCallbackProc</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNset</td>
<td>XtCSet</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SG•</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNunselectProc</td>
<td>XtCCallbackProc</td>
<td>XtCallbackProc</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI•</td>
</tr>
<tr>
<td>XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNancestorSensitive**

Range of Values:

- TRUE/"true"
- FALSE/"false"

---

**Flat Exclusives (3W)**

---

**Flat Exclusives (3W)**
This resource indicates the sensitivity of the sub-object’s ancestors. If TRUE, all the sub-object’s ancestors are sensitive and the sub-object is sensitive to user input. If FALSE, one or more of the sub-object’s ancestors are insensitive, so the sub-object displays an inactive visual and is not sensitive to user input.

**XtNbackground**
This is the pixel color used to fill in the background of the sub-object.

**XtNbackgroundPixmap**
This resource specifies the pixmap that is displayed as the sub-object’s label. Any supplied pixmap must have the same depth as the flat widget’s depth. Pixmaps of *None* and *ParentRelative* are not considered valid values. If either *XtNlabel* or *XtNlabelImage* has a non-NULL value, this resource is ignored.

**XtNclientData**
This is the client data supplied to all callback procedures.

**XtNdefault**
Range of Values:

```
TRUE/"true"
FALSE/"false"
```

When used on the sub-object, this resource specifies whether or not the sub-object is a default item. If more than one item is a set as a default item, a warning is generated and all but the first default item is unselected.

When used on the container, this resource indicates whether or not one of the sub-objects is a default item. If a sub-object is a default item, *XtNdefault* has a value of TRUE; else it has a value of FALSE. Setting this resource on the container widget indicates whether or not one of the sub-objects should be a default item. If the application sets this value to TRUE on the container, the container will set the first managed and mapped sub-object as the default item if a default item does not exist. If the application sets this value to FALSE, the container will unset its default item if one exists.

Even if the application does not use *XtNdefault* in its item fields list, the container will correctly maintain the default item and the application can change the default item via *OlFlatSetValues*.

**XtNdim**
Range of Values:

```
TRUE/"true"
FALSE/"false"
```

If TRUE, the sub-object shows a dimmed visual indicating that the item represents the state of one or more objects, that as a group, are in different states.

**XtNitemFields**
This is the list of resource names used to parse the records in the *XtNitems* list. This resource does not have to point to static information since the flat container does not use this information after initialization. Though the flat container does not reference this resource’s value after initialization, it holds onto it for responding to an *XtGetValues()* request and supplying it in the *OlFlatCallData* structure during callbacks. Therefore, if the application plans on querying this
resource, it’s recommended that the application make this resource point to static information.

**XtNItemsTouched**

Range of Values:

```plaintext
TRUE
FALSE
```

Whenever the application modifies an item list directly, it must supply this resource (with a value of TRUE) to the flat widget container so that the container can update the visual. If the resource value is supplied, the flat widget container treats its current item list as a new list and hence, updates its entire visual. Since the list is treated as a new list, the flat container may request a change in geometry from its parent.

Note: It is not necessary to use this resource if the application modifies the list with the `OlFlatSetValues` procedure; nor is it necessary to use this resource whenever the application supplies a new list to the flat container.

**XtNLabel**

This is the text string that appears in the sub-object.

**XtNLabelImage**

This is an `XImage` pointer that can appear in a sub-object. This resource is ignored if `XtNLabel` is non-NULL.

**XtNLabelJustify**

Range of Values:

```plaintext
OL_LEFT/"left"
OL_CENTER/"center"
OL_RIGHT/"right"
```

This resource specifies the justification of the label or `XImage` that appears within a sub-object.

**XtNLabelTile**

Range of Values:

```plaintext
TRUE/"true"
FALSE/"false"
```

This resource augments the `XtNLabelImage/XtNLabelPixmap` resource to allow tiling of the sub-object’s background. For an image/pixmap that is smaller than the sub-object’s background, the label area is tiled with the image/pixmap to fill the sub-object’s background if this resource is TRUE; otherwise, the label is placed as described by the `XtNLabelJustify` resource.

The `XtNLabelTile` resource is ignored for text labels.

**XtNMappedWhenManaged**

Range of Values:

```plaintext
TRUE/"true"
FALSE/"false"
```
This resource specifies whether or not a managed sub-object is displayed. Regardless of this resource’s value, all managed sub-objects will be including when determining the layout.

Note: This resource is never inherited from the container, so its default value is always TRUE.

**XtNnoneSet**
Range of Values:

- TRUE/"true"
- FALSE/"false"

This resource controls whether the settings can be toggled into an unset mode directly. If set to FALSE, exactly one sub-object must be in the set state always. Attempting to select the currently set sub-object does nothing. If set to TRUE, no more than one sub-object can be set at any time. However, the user can select the currently set sub-object and toggle it back to an unset state.

**XtNnumItems**
This resource specifies the number of sub-object items.

**XtNnumItemFields**
This resource indicates the number of resource names contained in XtNitemFields.

**XtNsameHeight**
Range of Values:

- OL_ALL/"all"
- OL_ROWS/"rows"

This resource specifies the rows that are forced to the same height.

**XtNsameWidth**
Range of Values:

- OL_ALL/"all"
- OL_COLUMNS/"columns"

This resource specifies the columns that are forced to the same width.

**XtNselectProc**
This callback procedure is called whenever the sub-object becomes selected by user input.

**XtNsensitive**
Range of Values:

- TRUE/"true"
- FALSE/"false"

If TRUE, the sub-object is sensitive to user input. If FALSE, the sub-object is insensitive to user input and an inactive visual is displayed to indicate this state.

Note: This resource is never inherited from the container, so its default value is always TRUE.
**XtNset**

Range of Values:

- TRUE/"true"
- FALSE/"false"

This resource reflects the current state of the sub-object.

Note: This resource is never inherited from the container, so its default value is always FALSE.

Even if the application does not use *XtNset* in its item fields list, the container will correctly maintain the set item and the application can change the set item via *OlFlatSetValues*.

**XtNunselectProc**

This callback procedure is called whenever the sub-object becomes unselected by user input.
NAME

FlatNonexclusives – a Primitive widget allowing the user to select one or more choices

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <QpenLook.h>
#include <FNonexclus.h>

widget = XtCreateWidget(name, flatNonexclusivesWidgetClass, ...
```

DESCRIPTION

Default Spacing

The default spacing between items is 50% of the prevailing point size for the container’s font.

![Example of Flat Nonexclusive Buttons](image)

**Figure 1. Example of Flat Nonexclusive Buttons**

Selection Control

Clicking the SELECT mouse button over an object that is set will cause the object to become unset and its `XtNunselectProc` procedure is called. If the object is unset, clicking the SELECT mouse button over it causes it to become set and its `XtNselectProc` procedure is called.
FlatNonexclusives (3W)

Use in a Menu

The FlatNonexclusives can be added as a child in a menu pane to implement a several-of-many menu choice.

FlatNonexclusives Coloration

The FlatNonexclusives container inherits its background color from the container’s parent widget. Setting the background color affects only the sub-objects’ background.

Keyboard Traversal

The FlatNonexclusives widget is a Primitive widget that manages the traversal between a set of sub-objects. When the user traverses to a FlatNonexclusives widget, the first sub-object in the set will display itself as having input focus (see the RectButton Widget for a description of this appearance.) The MOVEUP, MOVEDOWN, MOVERIGHT, and MOVELEFT keys move the input focus between the sub-objects. To traverse out of the FlatNonexclusives widget, the following keys can be used:

- NEXT_FIELD moves to the next traversable widget in the window
- PREV_FIELD moves to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application.
- PREVWINDOW moves to the previous window in the application.
- NEXTAPP moves to the first window in the next application.
- PREVAPP moves to the first window in the previous application.

Keyboard Operation

<table>
<thead>
<tr>
<th>Flat Nonexclusives Activation Types</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OL_MENUDEFAULTKEY</strong></td>
<td>If the FlatNonexclusives is on a menu, this command will set the item with focus to be menu’s default; otherwise, this command is ignored.</td>
</tr>
<tr>
<td><strong>OL_SELECTKEY</strong></td>
<td>This command acts as if the SELECT mouse button had been clicked on the RectButton with focus. See &quot;Selection Control&quot; section above.</td>
</tr>
</tbody>
</table>

Display of Keyboard Mnemonic

The FlatNonexclusives widget displays the mnemonic accelerator of a sub-object as part of the sub-object’s label. If the mnemonic character is in the label, then that character is highlighted according to the value of the application resource XtNshowMnemonics(). If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the application resource XtNshowMnemonics(). If truncation is necessary, the mnemonic displayed in parenthesis is truncated as a unit.
FlatNonexclusives (3W)

Display of Keyboard Accelerators
The FlatNonexclusives widget displays the keyboard accelerator as part of the sub-object's label. The string in the XtNacceleratorText resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

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

RESOURCES
The following table lists the resources for the FlatNonexclusives. Resources that have a bullet (*) in the Access column denote sub-object resources. If these resources are not included in the XtNitemFields list, they are inherited from the container widget. An application can change the default values for sub-object resources by setting them directly on the container. Even though a sub-object resource is not included in the XtNitemFields list, the application can query the value of any sub-object resource with OlFlatGetValues().

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>None</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNclientData</td>
<td>XtCClientData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdefault</td>
<td>XtCDefault</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>Cardinal</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdim</td>
<td>XtCDim</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct</td>
<td>(OPEN LOOK font)</td>
<td>SIF*</td>
</tr>
<tr>
<td>XtNforegroundColor</td>
<td>XtCForegroundColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgravity</td>
<td>XtCGravity</td>
<td>int</td>
<td>CenterGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhPad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhSpace</td>
<td>XtCHSpace</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNitemFields</td>
<td>XtCItemFields</td>
<td>String*</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNitemGravity</td>
<td>XtCItemGravity</td>
<td>int</td>
<td>NorthWestGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMaxHeight</td>
<td>XtCItemMaxHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMaxWidth</td>
<td>XtCItemMaxWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinHeight</td>
<td>XtCItemMinHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitemMinWidth</td>
<td>XtCItemMinWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNitems</td>
<td>XtCitems</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
</tbody>
</table>
**Flat Nonexclusives (3W)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNitemsTouched</td>
<td>XtCitemsTouched</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SG</td>
</tr>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNlabelImage</td>
<td>XtCLabelImage</td>
<td>XImage</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNlabelJustify</td>
<td>XtCLabelJustify</td>
<td>OlDefine</td>
<td>OL_LEFT</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNlabelTile</td>
<td>XtCLabelTile</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNlayoutHeight</td>
<td>XtCLayoutHeight</td>
<td>OlDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OlDefine</td>
<td>OL_FIXEDROWS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlayoutWidth</td>
<td>XtCLayoutWidth</td>
<td>OlDefine</td>
<td>OL_MINIMIZE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmanaged</td>
<td>XtCManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNnumItemFields</td>
<td>XtCNumItemFields</td>
<td>Cardinal</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameHeight</td>
<td>XtCSameHeight</td>
<td>OlDefine</td>
<td>OL_ALL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsameWidth</td>
<td>XtCSameWidth</td>
<td>OlDefine</td>
<td>OL_COLUMNS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNselectProc</td>
<td>XtCCallbackProc</td>
<td>XtCallbackProc</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNset</td>
<td>XtCSet</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNunselectProc</td>
<td>XtCCallbackProc</td>
<td>XtCallbackProc</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvSpace</td>
<td>XtCVSpace</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNancestorSensitive**

Range of Values:

- **TRUE**/"true"
- **FALSE**/"false"

This resource indicates the sensitivity of the sub-object’s ancestors. If **TRUE**, all the sub-object’s ancestors are sensitive and the sub-object is sensitive to user input. If **FALSE**, one or more of the sub-object’s ancestors are insensitive, so the sub-object displays an inactive visual and is not sensitive to user input.

**XtNbackground**

This is the pixel color used to fill in the background of the sub-object.
XtNbackgroundPixmap
This resource specifies the pixmap that is displayed as the sub-object’s label. Any supplied pixmap must have the same depth as the flat widget’s depth. Pixmaps of None and ParentRelative are not considered valid values. If either XtNlabel or XtNlabelImage has a non-NULL value, this resource is ignored.

XtNclientData
This is the client data supplied to all callback procedures.

XtNdefault
Range of Values:

TRUE/"true"
FALSE/"false"

When used on the sub-object, this resource specifies whether or not the sub-object is a default item. If more than one item is set as a default item, a warning is generated and all but the first default item is unselected.

When used on the container, this resource indicates whether or not one of the sub-objects is a default item. If a sub-object is a default item, XtNdefault has a value of TRUE; else it has a value of FALSE. Setting this resource on the container widget indicates whether or not one of the sub-objects should be a default item. If the application sets this value to TRUE on the container, the container will set the first managed and mapped sub-object as the default item if a default item does not exist. If the application sets this value to FALSE, the container will unset its default item if one exists.

Even if the application does not use XtNdefault in its item fields list, the container will correctly maintain the default item and the application can change the default item via OlFlatSetValues.

XtNdim
Range of Values:

TRUE/"true"
FALSE/"false"

If TRUE, the sub-object shows a dimmed visual indicating that the item represents the state of one or more objects, that as a group, are in different states.

XtNitems
This is the list of sub-object items. This value must point to a static list since flat containers reference this list after initialization but do not cache its information.

XtNitemFields
This is the list of resource names used to parse the records in the XtNitems list. This resource does not have to point to static information since the flat container does not use this information after initialization. Though the flat container does not reference this resource’s value after initialization, it holds onto it for responding to an XtGetValues() request and supplying it in the OlFlatCallData structure during callbacks. Therefore, if the application plans on querying this resource, it’s recommended that the application make this resource point to static information.
XtNItemsTouched
   Range of Values:
      TRUE
      FALSE

Whenever the application modifies an item list directly, it must supply this resource (with a value of TRUE) to the flat widget container so that the container can update the visual. If the resource value is supplied, the flat widget container treats its current item list as a new list and hence, updates its entire visual. Since the list is treated as a new list, the flat container may request a change in geometry from its parent.

Note: It is not necessary to use this resource if the application modifies the list with the OlFlatSetValues procedure, nor is it necessary to use this resource whenever the application supplies a new list to the flat container.

XtNLabel
   This is the text string that appears in the sub-object.

XtNLabelImage
   This is an XImage pointer that can appear in a sub-object. This resource is ignored if XtNlabel is non-NULL.

XtNLabelJustify
   Range of Values:
      OL_LEFT/"left"
      OL_CENTER/"center"
      OL_RIGHT/"right"

This resource specifies the justification of the label or XImage that appears within a sub-object.

XtNLabelTile
   Range of Values:
      TRUE/"true"
      FALSE/"false"

This resource augments the XtNLabelImage/XtNLabelPixmap resource to allow tiling of the sub-object's background. For an image/pixmap that is smaller than the sub-object's background, the label area is tiled with the image/pixmap to fill the sub-object's background if this resource is TRUE; otherwise, the label is placed as described by the XtNLabelJustify resource.

The XtNLabelTile resource is ignored for text labels.

XtNmappedWhenManaged
   Range of Values:
      TRUE/"true"
      FALSE/"false"

This resource specifies whether or not a managed sub-object is displayed. Regardless of this resource's value, all managed sub-objects will be including when determining the layout.
FlatNonexclusives (3W)

Note: This resource is never inherited from the container, so its default value is always TRUE.

**XtNnumItems**
This resource specifies the number of sub-object items.

**XtNnumItemFields**
This resource indicates the number of resource names contained in XtNitemFields.

**XtNsameHeight**
Range of Values:
- OL_ALL/"all"
- OL_ROWS/"rows"
- OL_NONE/"none"

This resource specifies the rows that are forced to the same height.

**XtNsameWidth**
Range of Values:
- OL_ALL/"all"
- OL_COLUMNS/"columns"
- OL_NONE/"none"

This resource specifies the columns that are forced to the same width.

**XtNselectProc**
This callback procedure is called whenever the sub-object becomes selected by user input.

**XtNsensitive**
Range of Values:
- TRUE/"true"
- FALSE/"false"

If TRUE, the sub-object is sensitive to user input. If FALSE, the sub-object is insensitive to user input and an inactive visual is displayed to indicate this state.

Note: This resource is never inherited from the container, so its default value is always TRUE.

**XtNset**
Range of Values:
- TRUE/"true"
- FALSE/"false"

This resource reflects the current state of the sub-object.

**Note:**
This resource is never inherited from the container, so its default value is always FALSE.
Even if the application does not use `XtNset` in its item fields list, the container will correctly maintain the set item and the application can change the set item via `OlFlatSetValues`.

**XtNunselectProc**

This callback procedure is called whenever the sub-object becomes unselected by user input.
NAME
FooterPanel – provides a convenient way to put a footer at the bottom of a window

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <QpenLook.h>
#include <FooterPane.h>

widget = XtCreateWidget(name, footerPanelWidgetClass, ...);

DESCRIPTION
Consistent Interface for Attaching Footer
The FooterPanel widget is a simple composite that provides a consistent interface for attaching a footer message to the bottom of a base window. The FooterPanel composite accepts two children: a Top Child and a Footer Child. (These are attached to the top and bottom of the FooterPanel widget, respectively.) The children are identified in the order they are added: the Top Child is the first child added; the Footer Child is the second.

Initial Size
The initial height of the FooterPanel widget is the sum of the initial heights of its children. The initial width is the widest of the initial widths of its children.

Sizing
The FooterPanel widget attempts to allow its children to grow or shrink to any size, by asking its parent to allow it to grow to the width of the widest child and the height of the sum of its children’s height. When it is not allowed to grow to this desired size, or when it is resized smaller by its parent, the FooterPanel imposes the size restriction as follows: It resizes both children to its width, but forces the Top Child to absorb all the height restriction; it does not resize the height of the Footer Child. Conversely, when it is resized larger by its parent, the FooterPanel widget gives all the height increase to the Top Child and resizes both children to the new width.

The FooterPanel widget never overlaps its children. If necessary, it will resize the Top Child to zero height. If its height becomes too small to accommodate the Footer Child’s height, it clips the Footer Child.

If queried by its parent about its preferred size, it in turn queries its children. The width in this query is the same for each child: the width in the parent’s query. The height in the query of the Footer Child is the child’s existing height, and the height in the query of the Top Child is the height in the parent’s query minus the existing height of the Footer Child. The FooterPanel widget’s response, then, to its parent’s query is a width equal to the wider preference of its children and a height equal to the sum of their preferred heights.

Works with All Children
The FooterPanel composite widget works with all the widgets defined in this document, except those that are sub-classed from the Shell widget class.
FooterPanel Coloration

The FooterPanel widget is "invisible" in that it imposes no coloration of its own.

RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SG</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNstring</td>
<td>XtCString</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNstring**

This resource is the string that will be drawn. The string must be null terminated.

See Appendix A, "General Resources," for a description of the other FooterPanel resources.
NAME

Form – a composite widget allowing sophisticated management of other widgets in its boundaries

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Form.h>

widget = XtCreateWidget(name, formWidgetClass, ...
```

DESCRIPTION

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

Spanning Constraints

A widget can be created with a set of constraints in such a manner that it spans the width or height of a form. Constraints that cause a widget to span both the width and height of a form can also be specified.

Row Constraints

Sets of widgets can be set up as a row so that resizing a form may increase or decrease the spacing between the widgets. The form may also make the widgets smaller if desired.

Column Constraints

Sets of widgets can be displayed in a single column or in multiple columns. The form may increase or decrease the spacing between widgets or resize the widgets.

Automatic Form Resizing

The form calculates new sizes or positions for its children whenever they change size or position. The new form size thus generated is passed as a geometry request to the parent of the form. Once resized, the form, using its children’s constraints, tries to rearrange its children as intelligently as possible.

Managing, Unmanaging and Destroying Children

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

Works with All Children

The Form composite widget works with all the widgets defined in this document, except those that are sub-classed from the Shell widget class.

Form Coloration

Figure 1 illustrates the resources that affect the coloration of the Form widget.
Figure 1. Form Coloration

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>White</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SG</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>
CONSTRAINT RESOURCES

Each child widget attached to the Form composite widget is constrained by the following resources: (In essence, these resources become resources for each child widget and can be set and read just like any other resources defined for the child.)

<table>
<thead>
<tr>
<th>Form Constraint Resource Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>XtNxAddWidth</td>
</tr>
<tr>
<td>XtNxAttachOffset</td>
</tr>
<tr>
<td>XtNxAttachRight</td>
</tr>
<tr>
<td>XtNxOffset</td>
</tr>
<tr>
<td>XtNxRefName</td>
</tr>
<tr>
<td>XtNxRefWidget</td>
</tr>
<tr>
<td>XtNxResizable</td>
</tr>
<tr>
<td>XtNxVaryOffset</td>
</tr>
<tr>
<td>XtNyAddHeight</td>
</tr>
<tr>
<td>XtNyAttachBottom</td>
</tr>
<tr>
<td>XtNyAttachOffset</td>
</tr>
<tr>
<td>XtNyOffset</td>
</tr>
<tr>
<td>XtNyRefName</td>
</tr>
<tr>
<td>XtNyRefWidget</td>
</tr>
<tr>
<td>XtNyResizable</td>
</tr>
<tr>
<td>XtNyVaryOffset</td>
</tr>
</tbody>
</table>

**XtNxAddWidth**

**XtNyAddHeight**

Range of Values:

TRUE
FALSE

These resources indicate whether to add the width or height of the corresponding reference widget to a widget’s location when determining the widget’s position.

**XtNxAttachOffset**

**XtNyAttachOffset**

Range of Values:

0 ≤ XtNxAttachOffset
0 ≤ XtNyAttachOffset

When a widget is attached to the right or bottom edge of the form, the separation between the widget and the form defaults to zero pixels. These resources allow that separation to be set to some other value. Also, for widgets that are not attached to the right or bottom edge of the form, these resources specify the minimum spacing between the widget and the form.
Widgets are normally referenced from "form left" to "form right" or from "form top" to "form bottom." These resources allow this reference to occur on the opposite edges of the form. When used with the XtNxVaryOffset and XtNyVaryOffset resources, they allow a widget to float along the right or bottom edge of the form. This is done by setting both the XtNxAttachRight (XtNyAttachBottom) and XtNxVaryOffset (XtNyVaryOffset) resources to TRUE. A widget can also span the width (height) of the form by setting the XtNxAttachRight (XtNyAttachBottom) resource to TRUE and the XtNxVaryOffset (XtNyVaryOffset) resource to FALSE.

The location of a widget is determined by the widget it references. As the default, a widget’s position on the form exactly matches its reference widget’s location. There are two additional data used to determine the location. These resources define integer values representing the number of pixels to add to the reference widget’s location when calculating the widget’s location.

When a widget is added as a child of the form, its position is determined by the widget it references. These resources allow the name of the reference widget to be given. The form converts this name to a widget to use for the referencing. Any widget that is a direct child of the form or the form widget itself can be used as a reference widget.

If one of these resources is set and the corresponding resource, XtNxRefWidget or XtNyRefWidget, is also set, they must agree: the name given in XtNxRefName or XtNyRefName must match the name of the identified widget. The advantage of using these resources rather than XtNxRefWidget and XtNyRefWidget is that the references can be used before the widget instances are made.
Instead of naming the reference widget, an application can give the reference widget's ID using these resources.

If both a widget ID and a widget name is given for a reference in the same dimension (x or y), they must refer to the same widget. If not, a warning is made and the widget ID is used.

**XtNxResizable**
**XtNyResizable**

Range of Values:

* TRUE
* FALSE

These resources specify whether the form can resize (expand or shrink) a widget. When a form's size becomes smaller, the form will resize its children only after resizing all the inter-widget spacing of widgets with their **XtNxVaryOffset** (**XtNyVaryOffset**) resource set to TRUE. The form keeps track of a widget's initial size or its size generated through calls to **XtSetValues()**, so that when the form then becomes larger, the widget will grow to its original size and no larger.

**XtNxVaryOffset**
**XtNyVaryOffset**

Range of Values:

* TRUE
* FALSE

When a form is resized, it processes the constraints contained within its children. These resources allow the spacing between a widget and the widget it references to vary (either increase or decrease) when a form's size changes. For widgets that directly reference the form widget, these resources are ignored. The spacing between a widget and its reference widget can decrease to zero pixels if the **XtNxAddWidth** (**XtNyAddHeight**) resource is FALSE or to one pixel if **XtNxAddWidth** (**XtNyAddHeight**) is TRUE.
NAME

Gauge – the graphical equivalent of a read only analog control

SYNOPSIS

```
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Gauge.h>

widget = XtCreateWidget(name, gaugeWidgetClass, ...)
```

DESCRIPTION

Gauge Components

The Gauge widget is merely an indicator of amount of space available in any container it is measuring.

It consists of the following elements:

- Bar (typically)
- Shaded Bar (typically)
- Current Value (not visible)
- Minimum Value (not visible)
- Maximum Value (not visible)

Figure 1. Horizontal Gauge Widget
**Application Notification**

The application is responsible for providing any feedback to the end user deemed appropriate, such as updating the Current Value in a text field.

**RESOURCES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>FontStruct*</td>
<td>(OPEN LOOK default)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td>XtFontList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgranularity</td>
<td>XtCGranularity</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminLabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxLabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNorientation</td>
<td>XtCOrientation</td>
<td>XtFont</td>
<td>OL_VERTICAL</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNslderMax</td>
<td>XtCSliderMax</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNslderMin</td>
<td>XtCSliderMax</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNslderValue</td>
<td>XtCSliderValue</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNspan</td>
<td>XtCSpan</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNticks</td>
<td>XtCTicks</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtickUnit</td>
<td>XtCTickUnit</td>
<td>XtTick</td>
<td>OL_NONE</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNuserDate</td>
<td>XtCUserDate</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNminLabel**

This is the label to be placed next to the minimum value position. For a vertical gauge, the label is placed to the right of the minimum value position. If there is not enough space for the entire label and \( \text{XtNrecomputeSize} \) is FALSE, the label will be truncated from the end. If there is not enough space for the entire label and \( \text{XtNrecomputeSize} \) is TRUE, then the widget will request for more space to show the entire label.
For an horizontal gauge, the label is placed centered and below the minimum value position. If there is not enough room to center the label and \texttt{XtNrecomputeSize} is set to FALSE, the beginning of the label will be aligned with the left anchor and is drawn to the right. If this label collides with the max label, some part of the labels will overlap. If there is not enough room to center the label and \texttt{XtNrecomputeSize} is set to TRUE, the widget will request for more space to center the label below the minimum value position.

\textbf{XtNmaxLabel}

This is the label to be placed next to the maximum value position. For a vertical gauge, the label is placed to the right of the minimum value position. If there is not enough space for the entire label and \texttt{XtNrecomputeSize} is FALSE, the label will be truncated from the end. If there is not enough space for the entire label and \texttt{XtNrecomputeSize} is TRUE, then the widget will request for more space to show the entire label.

For an horizontal gauge, the label is placed centered and below the maximum value position. If there is not enough room to center the label and \texttt{XtNrecomputeSize} is set to FALSE, the end of the label will be aligned with the left anchor. If this label collides with the min label, some part of the labels will overlap. If there is not enough room to center the label and \texttt{XtNrecomputeSize} is set to TRUE, the widget will request for more space to center the label below the maximum value position.

\textbf{XtNorientation}

Range of Values:

\begin{itemize}
  \item OL\_HORIZONTAL/"horizontal"
  \item OL\_VERTICAL/"vertical"
\end{itemize}

This resource defines the direction for the visual presentation of the widget.

\textbf{XtNsiderMax} \textbf{XtNsiderMin}

Range of Values: \texttt{XtNsiderMin < XtNsiderMax}

These two resources give the range of values tracked by the Gauge widget. Mathematically, the range is open on the right; that is, the range is the following subset of the set of integers:

\texttt{XtNsiderMin \leq range \leq XtNsiderMax}

\textbf{XtNsiderValue}

Range of Values: \texttt{XtNsiderMin \leq XtNsiderValue \leq XtNsiderMax}

This resource gives the current position of the Drag Box, in the range \texttt{[XtNsiderMin, XtNsiderMax]}. The Gauge widget keeps this resource up to date.

\textbf{XtNticks}

This is the interval between tick marks. The unit of the interval value is determined by \texttt{XtNtickUnit}.
XtNtickUnit

Range of values:

- OL_NONE/"none"
- OL_SLIDERVALUE/"slidervalue"
- OL_PERCENT/"percent"

This resource can have one of the values: OL_NONE, OL_SLIDERVALUE, and OL_PERCENT. If it is OL_NONE, then no tick marks will be displayed and XtNticks is ignored. If it is OL_PERCENT, then XtNticks is interpreted as the percent of the gauge value range. If it is OL_SLIDERVALUE, the XtNticks is interpreted as the same unit as gauge value.

Note: To be consistent with the scrollbar widget, we recommend that the effective spacing between tick marks, designated in XtNticks and XtNtickUnit be less than or equal to the spacing in XtNgranularity.

Figure 2. Gauge Widget with Tick Marks

XtNrecomputeSize

This resource, if set to TRUE, allows the gauge widget to resize itself whenever needed, to compensate for the space needed to show the tick marks and the labels. The gauge widget uses the XtNspan, the sizes of the labels, and XtNtickUnit to determine the preferred size.

XtNfont

This resource specifies the font used to draw the labels. It defaults to the OPEN LOOK standard font.
**XtNfontcolor**

This resource specifies the color used to draw the labels. It defaults to the foreground color of the gauge widget.

**XtNspan**

If *XtNrecomputeSize* is set to TRUE, then *XtNspan* should be set to reflect the preferred length of the gauge, not counting the space needed for the labels. The gauge widget uses the span value, the sizes of the labels, and *XtNtickUnit* to determine the preferred size.
NAME

**MenuButton** – provides a button with a MenuShell attached

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <MenuButton.h>

static Widget button, menupane, w;
Arg args[1];

button = XtCreateWidget(name, MenuButtonWidgetClass, ...);
OR
button = XtCreateWidget(name, menubuttonGadgetClass, ...);
XtSetArg(args[0], XtNmenuPane, &menupane);
XtGetValues(button, args, 1);
w = XtCreateWidget(name, widget-class, menupane, ...);
```

DESCRIPTION

The **MenuButton** widget provides the features of menu default selection and menu previewing as well as the features of the **Menu** widget.

**MenuButton Components**

Each menu button has the following parts:

- Label
- Border
Each menu button also has the components of a Menu widget. These are not shown in Figure 1.

**All Features of the Menu Widget**

The MenuButton widget includes all the features of the Menu widget; the features of that widget apply here.

**Selecting the Default Item**

As an interface to a menu, each MenuButton widget has a Default Item. If the power-user option is on, this Default Item is selected by clicking SELECT over the MenuButton widget. If the Default Item is inactive (its XtNsensitive resource is FALSE), or busy (its XtNbusy resource is TRUE), the system beeps. (If the power-user option is off, clicking SELECT brings up a pop-up menu. The power-user option is set in the property sheet of the Workspace Manager. See the OPEN LOOK™ GUI User's Guide for more information on setting the power-user option.)

Since the Default Item may be the MenuButton widget itself, selecting it really selects its Default Item; this recurses through the menu tree until a non-MenuButton widget is found as the Default Item. The Default Item may be the pushpin in a menu.
If a pushpin is the Default Item, the menu is brought up as a *pinned menu*.

**Previewing the Default Item**

If the menu button is not in a *pop-up menu* and the power-user option is on, pressing SELECT, or moving the pointer into the menu button while SELECT is pressed, displays the *highlighted* label of the Default Item in place of the menu button’s label. Releasing SELECT restores the original colors and label, and selects the Default Item as described above. Moving the pointer off the menu button before releasing also restores the original colors and label, but does not select the Default Item. (If the power-user option is off, pressing SELECT and releasing it displays a stayup menu. See *Selecting the Default Item* above for comments about the power-user option.)

This Default Item is the one in the menu directly under the menu button, not necessarily the Default Item at the end of the menu tree, that is activated when the Default Item is selected (see above).

**NOTE:**
The previewing feature is not accessible with keyboard only operation. This feature functions only when using a mouse to SELECT an item.

**Popping Up the Menu—Not in a Menu**

When the MenuButton widget is in a control area, pressing or clicking MENU when the pointer is within or on the Border pops up the menu button’s menu in the direction of the menu mark.

**Popping Up the Menu—As a Menu Button in a Menu**

When the MenuButton widget is in a *stay-up menu*, pressing or clicking MENU when the pointer is within or on the Border pops up the menu button’s menu in the direction of the menu mark. When the MenuButton widget is in a pop-up menu, moving the pointer into the menu mark region pops up the menu in the direction of the menu mark. The position is computed when the movement into the menu mark region is first detected, but rapid pointer motion and internal delays in popping up the menu may let the pointer wander.

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

**Menu Placement When There is No Room**

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

**MenuButton Coloration**

On a monochrome display, the MenuButton widget indicates that it has input focus by inverting the foreground and background colors of the control.
On color displays, when the MenuButton widget receives the input focus, the background color is changed to the input focus color set in the XtNinputFocusColor resource.

EXCEPTIONS:

— If the input focus color is the same as the font color for the control labels, then the coloration of the active control is inverted.

— If the input focus color is the same as the Input Window Header Color and the active control is in the window header, then invert the colors.

— If the input focus color is the same as the window background color, then the MenuButton widget inverts the foreground and background colors when it has input focus.

Figure 2 illustrates the resources that affect the coloration of the MenuButton widget. Events that occur outside the Border (but within the MenuButton widget) are still in the domain of the menu button.

Figure 2. Menu Button Coloration

Label Appearance
The XtNwidth, XtNheight, XtNrecomputeSize, and XtNlabelJustify resources interact to produce a truncated, clipped, centered, or left-justified label as shown in Figure 3.
When the label is centered or left-justified, the extra space is filled with the background color of the MenuButton widget, as determined by the XtNbackground and XtNbackgroundPixmap resources.

When a text label is truncated, the truncation occurs at a character boundary and a solid-white triangle is inserted to show that part of the label is missing. The triangle, of course, requires that more of the label be truncated than would otherwise be necessary. If the width of the button is too small to show even one character with the triangle, only the triangle is shown. If the width is so small that the entire triangle cannot be shown, the triangle is clipped on the right.

**Keyboard Traversal**

The default value of the XtNtraversalOn resource is True.

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

---

<table>
<thead>
<tr>
<th>XtNwidth</th>
<th>XtNrecomputeSize</th>
<th>XtNlabelJustify</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>any value</td>
<td>TRUE</td>
<td>any</td>
<td>Just Fits</td>
</tr>
<tr>
<td>needed</td>
<td>FALSE</td>
<td>OL_LEFT</td>
<td>Left Justified</td>
</tr>
<tr>
<td>for label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>needed</td>
<td>FALSE</td>
<td>OL_CENTER</td>
<td>Centered</td>
</tr>
<tr>
<td>for label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>needed</td>
<td>FALSE</td>
<td>any</td>
<td>Truncated</td>
</tr>
<tr>
<td>for label</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XtNheight</th>
<th>XtNrecomputeSize</th>
<th>XtNlabelJustify</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>any value</td>
<td>TRUE</td>
<td>any</td>
<td>Just Fits</td>
</tr>
<tr>
<td>needed</td>
<td>FALSE</td>
<td>any</td>
<td>Centered</td>
</tr>
<tr>
<td>for label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>needed</td>
<td>FALSE</td>
<td>any</td>
<td>Clipped</td>
</tr>
<tr>
<td>for label</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Label Appearance
To traverse out of the menu, the following keys can be used:

- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application

**Keyboard Operation**

The action of the SELECTKEY depends on whether the user has selected the power-user option. (The power-user option is set in the property sheet of the Workspace Manager. See the GUI User's Guide for more information on setting this option.) While focus is on the MenuButton and the power-user option is on, pressing the SELECTKEY activates the default menu item. If the power-user option is off, pressing the SELECTKEY posts the stayup menu.

The DEFAULTACTION key will activate the default control in the MenuButton widget as if the user clicked the SELECT button on the control. To dismiss a MenuButton's submenu while focus is with the submenu, the CANCEL key is used.

The MENUDFAULTKEY can be used by the user to change the default control in the MenuButton widget. When the user presses the MENUDFAULTKEY, the control which has input focus will become the default control.

<table>
<thead>
<tr>
<th>MenuButton/MenuGadget Activation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>OL_MENUKEY</td>
</tr>
<tr>
<td>OL_MENUDFAULTKEY</td>
</tr>
<tr>
<td>OL_SELECTKEY</td>
</tr>
</tbody>
</table>

**Display of Keyboard Mnemonic**

The MenuButton widget displays the mnemonic accelerator for its child as part of its label. If the mnemonic character is in the label, then that character is highlighted according to the value of the application resource XtNshowMnemonics. If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the application resource XtNshowMnemonics.

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

**Display of Keyboard Accelerators**

The MenuButton widget displays the keyboard accelerator as part of its label. The string in the XtNacceleratorText resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.
If truncation is necessary, the accelerator is truncated as a unit. The accelerator is truncated before the mnemonic or the label.

**MenuButton Gadgets**

*MenuButton* gadgets cannot be parents (i.e. cannot be used as the parent parameter when creating a widget or other gadget).

Gadgets share some core fields. But since they are not subclasses of *Core*, they do not have all *Core* fields. In particular, they don’t have a name field or a translation field (so translations cannot be specified/overridden).

**SUBSTRUCTURE**

**Menu Component**

Name: menu  
Class: Menu

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>*XtNcenter</td>
<td>XtCCenter</td>
<td>Boolean</td>
<td>TRUE</td>
<td>I</td>
</tr>
<tr>
<td>*XtNhPad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>*XtNhSpace</td>
<td>XtCHSpace</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>*XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OlDefine</td>
<td>OL_FIXEDROWS</td>
<td>I</td>
</tr>
<tr>
<td>*XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>XtNpushpin</td>
<td>XtCPushpin</td>
<td>OlDefine</td>
<td>OL_NONE</td>
<td>I</td>
</tr>
<tr>
<td>XtNpushpinDefault</td>
<td>XtCPushpinDefault</td>
<td>Boolean</td>
<td>FALSE</td>
<td>I</td>
</tr>
<tr>
<td>*XtNsameSize</td>
<td>XtCSameSize</td>
<td>OlDefine</td>
<td>OL_COLUMNS</td>
<td>I</td>
</tr>
<tr>
<td>XtNtitle</td>
<td>XtCTitle</td>
<td>String</td>
<td>(widget’s name)</td>
<td>I</td>
</tr>
<tr>
<td>*XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>*XtNvSpace</td>
<td>XtCVSpace</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
</tbody>
</table>

* See the *Menu* and *ControlArea* widgets for more information on these resources.

**RESOURCES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCAncestorSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>† XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdefault</td>
<td>XtCDefault</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>† XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
</tbody>
</table>
**MenuButton (3W)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>(class name)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelJustify</td>
<td>XtCLabelJustify</td>
<td>OlDefine</td>
<td>OL_LEFT</td>
<td>SGI</td>
</tr>
<tr>
<td>± XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmenuMark</td>
<td>XtCMenuMark</td>
<td>OlDefine</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

‡ These resources are not available to MenuButton gadgets.

**XtNdefault**

Range of Values:

- **TRUE**
- **FALSE**

If this resource is TRUE, the Border is doubled to two lines to show that the menu button contains the default choice of several buttons.

**XtNlabel**

This resource is a pointer to the text for the Label of the MenuButton widget.

**XtNlabelJustify**

Range of Values:

- OL_LEFT/"left"
- OL_CENTER/"center"

This resource dictates whether the Label should be left-justified or centered within the widget width.

**XtNmenuMark**

Range of Values:

- OL_DOWN
- OL_RIGHT

This resource specifies the direction of the menu arrow.

**XtNmenuPane**

This is the widget where menu items can be attached; its value is available once the MenuButton widget has been created.
XtNrecomputeSize

Range of Values:

TRUE
FALSE

This resource indicates whether the MenuButton widget should calculate its size and automatically set the XtNheight and XtNwidth resources. If set to TRUE, the MenuButton widget will do normal size calculations that may cause its geometry to change. If set to FALSE, the MenuButton widget will leave its size alone; this may cause truncation of the visible image being shown by the MenuButton widget if the fixed size is too small, or may cause padding if the fixed size is too large. The location of the padding is determined by the XtNlabelJustify resource.

SEE ALSO
MenuShell "Programmatic Menu Popup and Popdown"
NAME
MenuShell – used to create a menu not associated with a menu button

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Menu.h>

static Widget menu, menupane, w;
Arg args[1];

menu = XtCreatePopupShell(name, menuShellWidgetClass, ...);
XtSetArg(args[0], XtNmenuPane, &menupane);
XtGetValues(menu, args, 1);

w = XtCreateWidget(name, widget-class, menupane, ...);

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

Menu Components
A menu contains a set of Items that are presented to the end user for his or her selection. These are specified by the application as widgets attached to the menu. One of these Items is a Default Item. (A menu always has exactly one Default Item.) The Items are laid out in a Control Area. A menu also has a Title, a Title Separator, a Border or Window Border, and an optional Pushpin. Sometimes it also has a Drop Shadow. See Figure 1.

The application chooses the label for the Title and whether a menu has a Pushpin.

Sub-class of the Shell Widget
The Menu widget is a sub-class of the Shell widget. Therefore, as the SYNOPSIS shows, the XtCreatePopupShell() routine is used to create a menu instead of the normal XtCreateWidget().

The following table lists the VendorShell resources and defaults for the OPENLOOK Menu Shell. The Menu Shell is also a subclass of VendorShell.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNmenuButton</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>XtNpushpin</td>
<td>OIDefine</td>
<td>OL_NONE</td>
</tr>
<tr>
<td>XtNresizeCorners</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>XtNwindowHeader</td>
<td>Boolean</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Menu Pane

The Menu Pane is not described as a separate widget in these requirements; the only interface to it for the application programmer is as a parent widget to which the widgets comprising the menu items are attached. The menu items are not attached directly to the Menu widget, since a shell widget can take only one child. The SYNOPSIS shows how the widget ID of the Menu Pane is obtained from the Menu widget.

Connecting a Menu to a Widget

A menu can be connected to any widget, including primitive widgets. The connection is made by creating the menu widget as a child of the other widget. Of course, being a shell widget, the Menu widget is not a normal widget-child of its parent, but a pop-up child. If the application allows it, the menu augments the parent’s event list so that the popping-up of the menu is handled automatically.

Popup Control

Pressing MENU when the pointer is over the parent of the Menu widget causes the menu to be popped up. The menu is presented as a pop-up menu, where the Items are available for a press-drag-release type of selection (see below). Clicking MENU when the pointer is over the parent of the Menu widget also causes the menu to be popped up, but the menu is presented as a stay-up menu, where the Items are available for a click-move-click type of selection, instead (see below). A "slow click" (a press with a noticeable delay before the release) may show the menu as a pop-up on the press, then as a stay-up on the release.

Use of the Pushpin

The Pushpin is presented to the end user like any of the items to be selected from the menu, except that it is always the top-most item, and is presented visually as an "adornment" of the header, next to the Title (if present). The end user selects the Pushpin, pushing it in to cause the menu to remain on the display as an OPEN LOOK window or pinned menu, and pulling it out to make the menu a stay-up menu. To the end user, a pinned menu behaves indistinguishably from a command window.

The Default Item

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

Components Shown when Popped Up

A pop-up or stay-up menu shows the Title, Border, Pushpin (if available), Items, and Drop Shadow. The Title is left out if the menu is from either a menu button or an abbreviated menu button. A pinned menu shows the Window Border, Title, Pushpin, Items, but no Drop Shadow.
Figure 1. Menu Components

**Popup Position**

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

For the pop-up position when the menu is from a menu button, see `MenuButton(3W)`.

**Programmatic Menu Popup and Popdown**

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

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

A menuShellWidget id obtained by creating a menu explicitly.

emanate

This field specifies the object that the menu is currently associated with and it is supplied to the position proc when the menu positioning is done. If this field is NULL, the menu’s parent object is used as the emanate object for later positioning calculations.

item_index

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

state

Range of values:

- OL_PINNED_MENU
- OL_PRESS_DRAG_MENU
- OL_STAYUP_MENU

This specifies the state the menu should be in when it visibly appears on the screen.

set_position

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

x y

These coordinates are used by the positioning routine. Typically, these values represent the pointer with respect to the RootWindow. For example, xevent->xbutton.x_root and xevent->xbutton.y_root. However, if the menu’s state is OL_PINNED_MENU, these coordinates represent the desired upper-left hand corner of the pinned menu.

position_proc

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

```c
void (*OlMenuPositionProc)(menu, emanate, item_index, state, mx, my, px, py)
```

- menu: MenuShellWidget id to be positioned.
- emanate: Menu’s emanate widget.
item_index  Emanate item_index or OL_NO_ITEM

state  menu's state, either OL_PRESS_DRAG_MENU or OL_STAYUP_MENU

mx  Pointer containing the menu's current x location. If the position routine wants to move the menu, it should change this value. The position routine should

my  Pointer containing the menu's current y location. If the position routine wants to move the menu, it should change this value. The position routine should not move the menu explicitly.

px  Represents the x location supplied to the OlMenuPopup routine. If the position routine changes this value, the pointer is warped to the new x location.

py  Represents the y location supplied to the OlMenuPopup routine. If the position routine changes this value, the pointer is warped to the new y location.

void OlMenuPost(menu)
    Widget menu;

A convenience routine that is equivalent to OlMenuPopup(menu, NULL, OL_STAYUP_MENU, FALSE, 0, 0, (OlMenuPositionProc)NULL).

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

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

void OlMenuUnpost(menu)
    Widget menu;

A convenience routine that is equivalent to OlMenuPopdown(menu, FALSE).

Press-Drag-Release vs Click-Move-Click Selection Control

The Menu arranges for its children to respond to either the press-drag-release or the click-move-click type of selection. With the press-drag-release type of control, the user keeps MENU pressed and moves the pointer to the Item of choice; releasing MENU selects the Item and pops the menu down. If the pointer is not over an Item when MENU is released, the menu simply pops down. With the click-move-click type of control, the user moves the pointer to the Item of choice (MENU has already been released to end a click); clicking SELECT or MENU selects the Item and pops the menu down. If the pointer is not over an Item when SELECT or MENU is clicked, the menu simply pops down.

These selection methods apply to all menu items except menu buttons. For example, in Figure 1 above, Locate Owner can be selected using the methods described here. For the other items in Figure 1 (which are menu buttons), see MenuButton(3W) for the explanation of menu button selection behavior.
Converting a Stay-up Menu to a Pop-up Menu

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

Highlighting of Menu Items

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

Menu Coloration

Figure 2 illustrates the resources that affect the coloration of the Menu widget.

Figure 2. Menu Coloration

Keyboard Traversal

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

Popping up a Menu via the keyboard is done by traversing to a MenuButton, using NEXT_FIELD, PREV_FIELD, MOVEUP, MOVEDOWN, MOVERIGHT, or LEFT, and pressing the MENUKEY key. If a Menu is attached to a control besides a MenuButton, it can be popped up by traversing to that control and pressing the MENUKEY.
Keyboard traversal within a Menu is done using the PREV_FIELD, NEXT_FIELD, MOVEUP, MOVEDOWN, MOVELEFT and MOVERIGHT keys. The PREV_FIELD, MOVEUP, and MOVELEFT keys move the input focus to the previous Menu item with keyboard traversal enabled. If the input focus is on the first item of the Menu, then pressing one of these keys will wrap to the last item of the Menu with keyboard traversal enabled. The NEXT_FIELD, MOVEDOWN, and MOVERIGHT keys move the input focus to the next Menu item with keyboard traversal enabled. If the input focus is on the last item of the Menu, then pressing one of these keys will wrap to the first item of the Menu with keyboard traversal enabled.

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCEL</td>
<td>dismisses the menu and returns focus to the originating control</td>
</tr>
<tr>
<td>NEXTWINDOW</td>
<td>moves to the next window in the application</td>
</tr>
<tr>
<td>PREVWINDOW</td>
<td>moves to the previous window in the application</td>
</tr>
<tr>
<td>NEXTAPP</td>
<td>moves to the first window in the next application</td>
</tr>
<tr>
<td>PREVAPP</td>
<td>moves to the first window in the previous application</td>
</tr>
</tbody>
</table>

**Keyboard Operation**

If input focus is on a MenuButton with in a Menu, pressing the MENUKEY will post the cascading Menu associated with the MenuButton, and input focus will be on the first Menu item with traversal enabled.

The DEFAULTACTION key will activate the default control in the Menu widget as if the user clicked the SELECT button on the control.

The MENUDEFAULTKEY can be used by the user to change the default control in the Menu widget. When the user presses the MENUDEFAULTKEY, the control which has input focus will become the default control.

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

<table>
<thead>
<tr>
<th>Activation Type</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL_CANCEL</td>
<td>Dismiss this menu and any other menus cascading off of it.</td>
</tr>
<tr>
<td>OL_DEFAULTACTION</td>
<td>Calls olActivateWidget for the default item with the activation_type as OL_SELECTKEY.</td>
</tr>
<tr>
<td>OL_TOGGLEPUSHPIN</td>
<td>Same semantics as TOGGLEPUSHPIN above.</td>
</tr>
</tbody>
</table>

**SUBSTRUCTURE**

Menu Pane component
### Application Resources

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNcenter</td>
<td>XtCCenter</td>
<td>Boolean</td>
<td>TRUE</td>
<td>I</td>
</tr>
<tr>
<td>XtNhPad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>XtNhSpace</td>
<td>XtCHSpace</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OIDefine</td>
<td>OL_FIXEDROWS</td>
<td>I</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>XtNsameSize</td>
<td>XtCSameSize</td>
<td>OIDefine</td>
<td>OL_COLUMNS</td>
<td>I</td>
</tr>
<tr>
<td>XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>XtNvSpace</td>
<td>XtCVSpace</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
</tbody>
</table>

### RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNallowShellResize</td>
<td>XtCallowShellResize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCancestorSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCbackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI+</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcreatePopupChildProc</td>
<td>XtCreatePopupChildProc</td>
<td>XtCreatePopupChildProc</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCcallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfocusWidget</td>
<td>XtCfocusWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheightInc</td>
<td>XtCHHeightInc</td>
<td>int</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinput</td>
<td>XtCInput</td>
<td>Boolean</td>
<td>FALSE</td>
<td>G</td>
</tr>
<tr>
<td>XtNmaxAspectX</td>
<td>XtCmaxAspectX</td>
<td>Position</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxAspectY</td>
<td>XtCmaxAspectY</td>
<td>Position</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxHeight</td>
<td>XtCmaxHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxWidth</td>
<td>XtCmaxWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmenuAugment</td>
<td>XtCMenuAugment</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI</td>
</tr>
<tr>
<td>XtNmenuPane</td>
<td>XtCMenuPane</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNminAspectX</td>
<td>XtCminAspectX</td>
<td>Position</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminAspectY</td>
<td>XtCminAspectY</td>
<td>Position</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminHeight</td>
<td>XtCminHeight</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminWidth</td>
<td>XtCminWidth</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNpopdownCallback</td>
<td>XtCcallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNpopupCallback</td>
<td>XtCcallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNpushpin</td>
<td>XtCpushpin</td>
<td>OIDefine</td>
<td>OL_NONE</td>
<td>GI</td>
</tr>
<tr>
<td>XtNpushpinDefault</td>
<td>XtCpushpinDefault</td>
<td>Boolean</td>
<td>FALSE</td>
<td>GI</td>
</tr>
<tr>
<td>XtNssaveUnder</td>
<td>XtCsSaveUnder</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtitle</td>
<td>XtCTitle</td>
<td>String</td>
<td>(widget's name)</td>
<td>SGI</td>
</tr>
</tbody>
</table>
### Menu Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidthInc</td>
<td>XtCWidthInc</td>
<td>int</td>
<td>-1</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNfocusWidget**

This is the ID of the widget to get focus the next time this shell takes focus. Therefore, as a user traverses objects via the keyboard or explicitly sets focus to an object (for example, clicking SELECT), the value of the XtNfocusWidget resource is updated to reflect this as the object with focus.

**XtNmenuAugment**

Range of Values:

- TRUE
- FALSE

If this resource is TRUE, the Menu widget will augment its parent’s event handling so that the pressing or clicking of MENU automatically pops up the menu. If FALSE, the application is responsible for detecting when the menu should be popped up. (Please see the earlier section on "Programmatic Menu Popup and Popdown.")

**XtNmenuPane**

This is the widget where menu items can be attached; its value is available once the Menu widget has been created.

**XtNpushpin**

Range of Values:

- OL_NONE/"none"
- OL_OUT/"out"

This resource controls whether the Menu widget has a pushpin. If set to OL_NONE, no pushpin will be included in the list of menu items, which means the user cannot pin the menu to keep it around. If set to OL_OUT, a pushpin will be included as an item the user can select; if the end user selects the pushpin, the menu will be made into an OPEN LOOK window. Note that the pushpin item is always at the top of the menu list. (This resource is also available in other widgets, but with three allowed values, including OL_IN. This third value is not allowed for the Menu widget.)

**XtNpushpinDefault**

Range of Values:

- TRUE
- FALSE

Setting this resource to TRUE makes the Pushpin the Default Item.

Note:

If a menu has a pushpin and none of the menu pane items have been designated as the default, the pushpin automatically becomes the menu’s Default Item.
**XtNtitle**

This resource gives the Title of the Menu widget.
Nonexclusives (3W)

NAME
Nonexclusives – allows the end-user to select one or more of a set of choices

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Nonexclusi.h>

widget = XtCreateWidget(name, nonexclusivesWidgetClass, ...);

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

Grid Layout and Button Labels
The Nonexclusives widget lays out the rectangular buttons or check boxes in a grid in the order they are added as child widgets by the application. The number of rows or columns in this grid can be controlled by the application. If the grid has more than one row, the Nonexclusives widget forces the rectangular buttons or check boxes in each column to be the same size as the widest in the column. (Note: If the grid is a single row, each button will be only as wide as necessary to display the label.)

![Figure 1. Example of Nonexclusive Buttons](image)

Figure 1. Example of Nonexclusive Buttons
Use in a Menu

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

Child Constraint

The `Nonexclusives` widget constrains its child widgets to be of class `rectButtonWidgetClass` or `checkBoxWidgetClass`. Additionally, all the child widgets must be of the same class.

Nonexclusives Coloration

There is no explicit foreground or background in the `Nonexclusives` composite widget; each rectangular button has its own foreground and background. The space between the rectangular buttons or check boxes is the same color or pixmap as the parent of the `Nonexclusives` widget.

Keyboard Traversal

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

- `NEXT_FIELD` moves to the next traversable widget in the window
- `PREV_FIELD` moves to the previous traversable widget in the window
- `NEXTWINDOW` moves to the next window in the application.
- `PREVWINDOW` moves to the previous window in the application.
- `NEXTAPP` moves to the first window in the next application.
- `PREVAPP` moves to the first window in the previous application.

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

### RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Type Class</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>XtNancestorSensitive</code></td>
<td><code>XtCSensitive</code></td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td><code>XtNconsumeEvent</code></td>
<td><code>XtCConsumeEvent</code></td>
<td>NULL</td>
<td>SG</td>
</tr>
<tr>
<td><code>XtNdepth</code></td>
<td><code>XtCDepth</code></td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td><code>XtNdestroyCallback</code></td>
<td><code>XtCCallback</code></td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td><code>XtNheight</code></td>
<td><code>XtCHeight</code></td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td><code>XtNlayoutType</code></td>
<td><code>XtCLayoutType</code></td>
<td>OLDefine</td>
<td>OLI_FIXEDROWS</td>
</tr>
<tr>
<td><code>XtNmappedWhenManaged</code></td>
<td><code>XtCMappedWhenManaged</code></td>
<td>TRUE</td>
<td>SG</td>
</tr>
<tr>
<td><code>XtNmeasure</code></td>
<td><code>XtCMeasure</code></td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td><code>XtNreferenceName</code></td>
<td><code>XtCReferenceName</code></td>
<td>NULL</td>
<td>GI</td>
</tr>
</tbody>
</table>
### Nonexclusives Resource Set (cont’d)

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE*</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SCI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

### XtNlayoutType

Range of Values:
- `OL_FIXEDROWS"fixedrows"
- `OL_FIXEDCOLS"fixedcols"

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

The values of the `XtNlayoutType` resource can be:
- `OL_FIXEDROWS` if the layout should have a fixed number of rows;
- `OL_FIXEDCOLS` if the layout should have a fixed number of columns.

### XtNmeasure

Range of Values:
- `0 < XtNmeasure`

This resource gives the number of rows or columns in the layout of the child widgets. If there are not enough child widgets to fill a row or column, the remaining space is left blank. If there is only one row (column), and it is not filled with child widgets, the remaining “space” is of zero width (height).
NAME
NoticeShell – contains a message area which is used for user confirmation

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Notice.h>

static Widget notice, textarea, controlarea, w;
Arg args[2];
notice = XtCreatePopupShell(name, noticeShellWidgetClass, ...);
XtSetArg(args[0], XtNtextArea, &textarea);
XtSetArg(args[1], XtNcontrolArea, &controlarea);
XtGetValues(notice, args, 2);
w = XtCreateWidget(name, widget-class, controlarea, ...);
.

DESCRIPTION
Notice Components
The Notice widget has three components: a Text Area where the message to the end user is displayed; a Control Area containing one or more widgets that the end user uses to control how to continue with an application; and a Default Button. Another important element is the Emanate Widget, which is typically the control activated by the end user that requires immediate attention. The application identifies the Emanate Widget to the Notice widget.
The Notice widget is a sub-class of the Shell widget. Therefore, as the SYNOPSIS shows, the `XtCreatePopupShell` routine is used to create a notice instead of the normal `XtCreateWidget`.

The following table lists the VendorShell resources and defaults for the Notice Shell. The Notice Shell is also a subclass of the OPEN LOOK VendorShell.

<table>
<thead>
<tr>
<th>Default Window Decorations</th>
<th>Resource</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNmenuButton</td>
<td>Boolean</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>XtNpushpin</td>
<td>OlDefine</td>
<td>OL_NONE</td>
<td></td>
</tr>
<tr>
<td>XtNresizeCorners</td>
<td>Boolean</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>XtNwindowHeader</td>
<td>Boolean</td>
<td>FALSE</td>
<td></td>
</tr>
</tbody>
</table>

Popping the Notice Up/Down

The application controls when the Notice widget is to be displayed or popped up. As shown in the SYNOPSIS, the `XtPopup` routine can be used for this.

However, the application does not need to control when the Notice widget is to be popped down. The widget itself detects when to pop down: the end user clicks SELECT on an OblongButton widget in the Control Area. This behavior requires that there be at least one OblongButton widget in the Control Area. If other types of controls are used instead, the application can "manually" pop the notice down using a routine such as `XtPopdown`.
Busy Button, Busy Application
When the Notice pops up, it "freezes" the entire application except the Notice to prevent the end user from interacting with any other part of the application. As feedback of this to the user, the Notice causes the headers of all the base windows and pop-up windows to be stippled in the "busy" pattern, and causes a stipple pattern in the Emanate Widget. The latter stipple pattern is caused by setting the \texttt{XtNbusy} resource to \texttt{TRUE} in the Emanate Widget. If the widget does not recognize this resource, nothing will happen.

On popping down, the Notice widget clears all stipple patterns and "unfreezes" the application.

Text and Control Areas
The Text and Control Areas are handled by separate widget interfaces. The SYNOPSIS shows how the widget IDs of the Text Area (\texttt{textarea}) and the Control Area (\texttt{controlarea}) are obtained from the Notice widget.

The Text and Control Areas abut so that there is no space between the two. An application can control the distance between the text and the controls by setting margins in the Control Area.

Notice Coloration
Figure 2 illustrates the resources that affect the coloration of the Notice widget.
Keyboard Traversal

The Notice widget limits keyboard traversal of the application to the buttons within the ControlArea. The user can traverse between the controls in the ControlArea using the NEXTFIELD, PREVFIELD, MOVEUP, MOVEDOWN, MOVERIGHT, and MOVELEFT keys. The NEXTAPP key will traverse to the next application, and the PREVAPP key will traverse to the previous application, but the NEXTWINDOW and PREVWINDOW keys are disabled. When keyboard traversal is used to move back to the Notice’s application, focus goes to the Notice.

<table>
<thead>
<tr>
<th>Notice Activation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>OL_CANCEL</td>
</tr>
<tr>
<td>OL_DEFAULTACTION</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**SUBSTRUCTURE**

**Control Area component**

Name: controlarea  
Class: ControlArea

<table>
<thead>
<tr>
<th>Application Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>XtNhPad</td>
</tr>
<tr>
<td>XtNhSpace</td>
</tr>
<tr>
<td>XtNlayoutType</td>
</tr>
<tr>
<td>XtNmeasure</td>
</tr>
<tr>
<td>XtNsameSize</td>
</tr>
<tr>
<td>XtNVPad</td>
</tr>
<tr>
<td>XtNvSpace</td>
</tr>
</tbody>
</table>

See the ControlArea widget for the descriptions of these resources.

**Text Area component**

Name: textarea  
Class: StaticText

<table>
<thead>
<tr>
<th>Application Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>XtNalignment</td>
</tr>
<tr>
<td>XtNfont</td>
</tr>
<tr>
<td>XtNfontColor</td>
</tr>
<tr>
<td>XtNfontGroup</td>
</tr>
<tr>
<td>XtNlineSpace</td>
</tr>
<tr>
<td>XtNstring</td>
</tr>
<tr>
<td>XtNstrip</td>
</tr>
<tr>
<td>XtNwrap</td>
</tr>
</tbody>
</table>

See the StaticText widget for the descriptions of these resources.
**RESOURCES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNallowShellResize</td>
<td>XtCAllowShellResize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGH+</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGH+</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGH+</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGH+</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcontrolArea</td>
<td>XtCControlArea</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNcreatePopupChildProc</td>
<td>XtCCreatePopupChildProc</td>
<td>XtCreatePopupChildProc</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNemanateWidget</td>
<td>XtCEmanateWidget</td>
<td>Widget</td>
<td>(parent’s)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfocusWidget</td>
<td>XtCFocusWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgeometry</td>
<td>XtCGeometry</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNpopupCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNnextArea</td>
<td>XtCnextArea</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNresourceData</td>
<td>XtCresourceData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNcontrolArea**

This is the widget ID of the **ControlArea** class composite child widget where controls can be attached; its value is available once the **Notice** widget has been created.

Any widgets of the class **OblongButton** added to the Control Area are assumed to be window disposition controls; that is, when the end user activates one of them, the **Notice** widget pops itself down.

**XtNemanateWidget**

Range of Values:

(ID of existing widget)

This resource identifies the Emanate Widget. On popping up, the **Notice** widget attempts to set this widget to be busy, by making its **XtNbusy** resource TRUE; if the widget doesn’t recognize the resource, nothing happens. On popping down, the **Notice** widget clears the **XtNbusy** resource.
When the Notice widget pops up, it tries not to cover this widget; this may fail depending on its location and the size of the Notice widget.

The default for this resource is the parent. The parent, however, cannot be a Gadget (OblongButtonGadget, for instance). To emanate a Notice from a Gadget, specify another widget as the parent and set XtNemanateWidget to the Gadget.

**XtNtextArea**

This is the widget ID of the StaticText class child widget that controls the Text Area; its value is available once the Notice widget has been created.
NAME

OblongButton - a one-choice element or button used to execute a command

SYNOPSIS

```
#include <Intrinsic.h>
#include <OpenLook.h>
#include <StringDefs.h>
#include <OblongButt.h>

widget = XtCreateWidget(name, oblongButtonWidgetClass, ...);
OR
widget = XtCreateWidget(name, oblongButtonGadgetClass, ...);
```

DESCRIPTION

OblongButton Components

The OblongButton consists of a Label surrounded by a rounded oblong, Border.

![Diagram of OblongButton components](image)

Figure 1. Oblong Buttons

Busy Button while Action Takes Place

Each OblongButton is associated with an application-defined action implemented as a list of callbacks. To let the end user know that an action is still taking place, the OblongButton stipplest the area inside the border before issuing the callbacks. When the last callback returns, the OblongButton restores its original appearance. If the application’s action continues to be "busy" after the callbacks return, the application should set the XtNbusy resource to TRUE before returning from the callbacks, then reset it to FALSE when the action is no longer taking place.
The "busy" stipple pattern is designed to show enough dots to gray the button noticeably, while still leaving a text label legible.

**Oblong Buttons in a Pop-up Menu**
Entering an oblong button while MENU is depressed previews the set appearance of the button. Releasing MENU then restores the original appearance and invokes the action for the button as described above. Leaving the button before releasing MENU restores the appearance but does not invoke the action.

**Oblong Buttons not in a Pop-up Menu**
Clicking SELECT on an oblong button starts the action associated with the button. Pressing SELECT, or moving the pointer into the button while SELECT is pressed, previews the set appearance of the button. Releasing SELECT restores the appearance and invokes the action for the button as described above. Moving the pointer off the button before releasing SELECT also restores the appearance, but does not invoke the action.

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

**OblongButton Gadgets**
OblongButton gadgets cannot be parents (i.e., be used as the parent parameter when creating a widget or other gadget).

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

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

Event Handlers cannot be added to gadgets using XtAddEventHandler.

**Label Appearance**
The XtNwidth, XtNheight, XtNrecomputeSize, and XtNlabelJustify resources interact to produce a truncated, clipped, centered, or left-justified label as shown in Figure 3.
When the label is centered or left-justified, the extra space is filled with the background color of the OblongButton widget, as determined by the XtNbackground and XtNbackgroundPixmap resources.

When a text label is truncated, the truncation occurs at a character boundary and a solid triangle is inserted to show that part of the label is missing. The triangle requires that more of the label be truncated than would otherwise be necessary. If the width of the button is too small to show even one character with the triangle, only the triangle is shown. If the width is so small that the entire triangle cannot be shown, the triangle is clipped on the right.

An image label is simply truncated; no triangle is shown.

See also the XtNlabelTile resource for how it affects the appearance of a label image.

**OblongButton Coloration**

Figure 2 illustrates the resources that affect the coloration of the OblongButton widget.

Note: Events that occur outside the Border (but within the OblongButton widget) are still in the domain of the button.

On a monochrome display, the OblongButton widget indicates that it has input focus by inverting the foreground and background colors of the control.
On color displays, when the OblongButton widget receives the input focus, the background color is changed to the input focus color set in the XtNInputFocusColor resource.

EXCEPTIONS:

— If the input focus color is the same as the font color for the control labels, then the coloration of the active control and fonts is inverted.

— If the input focus color is the same as the Input Window Header Color and the active control is in the window header, then the colors are inverted.

— If the input focus color is the same as the window background color, then the OblongButton widget inverts the foreground and background colors when it has input focus.

![Diagram of Oblong Button Coloration](image)

**Figure 2. Oblong Button Coloration**

**Keyboard Traversal**

The default value of the XtNtraversalOn resource is True.

The OblongButton widget responds to the following keyboard navigation keys:

- NEXT_FIELD, MOVEDOWN, and MOVERIGHT
  move to the next traversable widget in the window

- PREV_FIELD, MOVEUP, and MOVELEFT
  move to the previous traversable widget in the window
OblongButton (3W)

NEXTWINDOW moves to the next window in the application
PREVWINDOW moves to the previous window in the application
NEXTAPP moves to the first window in the next application
PREVAPP moves to the first window in the previous application

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

<table>
<thead>
<tr>
<th>Activation Type</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL_MENUDEFAULTKEY</td>
<td>Set the sub-object as the shell’s default object (if on a menu)</td>
</tr>
<tr>
<td>OL_SELECTKEY</td>
<td>Call its callbacks</td>
</tr>
</tbody>
</table>

### Display of Keyboard Mnemonic

The OblongButton widget displays the mnemonic accelerator for its child as part of its label. If the mnemonic character is in the label, then that character is displayed/highlighted according to the value of the application resource XtNshowMneumonics. It is displayed to the right of the label in parenthesis and displayed/highlighted according to the value of the application resource XtNshowMneumonics. If truncation is necessary, the mnemonic displayed in parenthesis is truncated as a unit.

### Display of Keyboard Accelerators

The OblongButton widget displays the keyboard accelerator as part of its label. The string in the XtNacceleratorText resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

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

### RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI‡</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI‡</td>
</tr>
<tr>
<td>XtNbusy</td>
<td>XtCBusy</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdefault</td>
<td>XtCDefault</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct*</td>
<td>(OPEN LOOK font)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black*</td>
<td>SGI</td>
</tr>
</tbody>
</table>
OblongButton Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>(class name)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelImage</td>
<td>XtCLabelImage</td>
<td>XImage *</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelJustify</td>
<td>XtCLabelJustify</td>
<td>OlDefine</td>
<td>OL_LEFT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelTile</td>
<td>XtCLabelTile</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelType</td>
<td>XtCLabelType</td>
<td>OlDefine</td>
<td>OL_STRING</td>
<td>SGI</td>
</tr>
<tr>
<td>‡ XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNselect</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

‡ These resources are not available to OblongButton gadgets.

**XtNbusy**

Range of Values:

**TRUE**  
**FALSE**

This resource controls whether the button interior should be stippled to show that the action associated with the button is "busy." While **XtNbusy** is TRUE, the system will beep if the end user attempts to select the button; the attempt is refused and no callbacks are invoked.

**XtNdefault**

Range of Values:

**TRUE**  
**FALSE**

If this resource is TRUE, and the button is in a menu, an oval ring is drawn around the button to show that the button is the default choice of one or more buttons.

**XtNlabel**

This resource is a pointer to the text for the Label. This resource is ignored if the **XtNlabelType** resource has the value OL_IMAGE.
XtNlabelImage
This resource is a pointer to the image for the Label. This resource is ignored unless the XtNlabelType resource has the value OL_IMAGE.

If the image is of type XYBitmap, the image is highlighted when appropriate by reversing the 0 and 1 values of each pixel (i.e. by "xor'ing" the image data). If the image is of type XYPixmap or ZPixmap, the image is not highlighted, although the space around the image inside the Border is highlighted.

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

XtNlabelJustify
Range of Values:
OL_LEFT/"left"
OL_CENTER/"center"

This resource dictates whether the Label should be left-justified or centered within the widget width.

XtNlabelTile
Range of Values:
TRUE
FALSE

This resource augments the XtNlabelImage/XtNlabelPixmap resource to allow tiling the sub-object's background. For an image/pixmap that is smaller than the sub-object's background, the label area is tiled with the image/pixmap to fill the sub-object's background if this resource is TRUE; otherwise, the label is placed as described by the XtNlabelJustify resource.

The XtNlabelTile resource is ignored for text labels.

XtNlabelType
Range of Values:
OL_STRING/"string"
OL_IMAGE/"image"
OL_POPUP/"popup"

This resource identifies the form that the Label takes. It can have the value OL_STRING for text, OL_IMAGE for an image, or OL_POPUP for text followed by an ellipsis (such as label...).

XtNrecomputeSize
Range of Values:
TRUE
FALSE
This resource indicates whether the OblongButton widget should calculate its size and automatically set the XtNheight and XtNwidth resources. If set to TRUE, the OblongButton widget will do normal size calculations that may cause its geometry to change. If set to FALSE, the OblongButton widget will leave its size alone; this may cause truncation of the visible image being shown by the OblongButton widget if the fixed size is too small, or may cause padding if the fixed size is too large. The location of the padding is determined by the XtNlabelJustify resource.

XtNselect
This is the list of callbacks invoked when the widget is selected.
NAME

PopupWindowShell - creates an OPEN LOOK property window or command window

SYNOPSIS

#include <Intrinsic.h>
#include <StringDefs.h>
#include <QpenLook.h>
#include <PopupWindo.h>

static void
Apply (w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
{
  printf ("Apply callback!\n");
}

Arg  args[3];
Widget   popupwindow, parent;
Widget   upper, lower, panel;

static XtCallbackRec applycalls[] = {
  { Apply, NULL },
  { NULL, NULL },
};

XtSetArg(args[0], XtNapply, (XtArgVal) applycalls);

popupwindow = XtCreatePopupShel1 ( name,
         popupWindowShellWidgetClass,
         parent, args, 1);

XtSetArg(args[0], XtNupperControlArea, &upper);
XtSetArg(args[1], XtNlowerControlArea, &lower);
XtSetArg(args[2], XtNfooterPanel, &panel);

XtGetValues(popupshell, args, 3);

w = XtCreateWidget (name, widget-class, upper, ...);

w = XtCreateWidget (name, widget-class, lower, ...);

w = XtCreateWidget (name, widget-class, panel,
   ...
   ...
XtPopup(popupwindow, XtGrabNone);
DESCRIPTION

Controls not Automatically Related to Selected Objects

The PopupWindow widget can be used to implement the OPEN LOOK property window. It manages the creation of a property window and provides a simple interface for populating the window with controls. However, it has no innate semantics to relate the controls with a selected object; this must be handled by the application. For example, the application must dim all the controls if an object selected by the end user is incompatible with a displayed property window.

PopupWindow Components

The PopupWindow widget has the following parts:

- Upper Control Area
- Lower Control Area
- Window Border
- Popup Window Menu
- Settings Menu (Conditional)
- Apply Button (Conditional)
- Reset Button (Conditional)
- Reset to Factory Button (Conditional)
- Set Defaults Button (Conditional)
- Header
- Window Mark
- Pushpin (Optional)
- Resize Corners (Optional)
- Footer (Optional)

The Window Border, Popup Window Menu, Header, Window Mark, and Pushpin provide the end user with window management controls over the PopupWindow widget. The Apply, Reset, Reset to Factory, and Set Defaults Buttons are automatically created, if needed, to help create a standard layout of a property window. The application controls which of these, if any, appear in the pop-up window.
Automatic Addition of Buttons, Settings Menu
To aid in the creation of a property window, the PopupWindow has several callbacks typically used in such a pop-up window, for applying, resetting, etc. For each of these callbacks that the application sets in the argument list used for creation of the PopupWindow, the PopupWindow widget automatically creates a button in the Lower Control Area, and the same button in the Settings Menu. If none of the callbacks are defined, no buttons are automatically created and no Settings Menu is created.

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

Sub-class of the Shell Widget
The PopupWindow widget is a sub-class of the Shell widget. Therefore, as the SYNOPSIS shows, the XtCreatePopupShell() routine is used to create a pop-up window instead of the normal XtCreateWidget().

The following table lists the VendorShell resources and defaults for the PopupWindowShell. The PopupWindowShell is also a subclass of the OPEN LOOK VendorShell.
Popping the Pop-up Window Up/Down
The application controls when the `PopupWindow` widget is to be displayed, or popped up. As indicated in the SYNOPSIS, the `XtPopup()` routine can be used for this.

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

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

Upper and Lower Control Areas
The Upper and Lower Control Areas are handled by separate widget interfaces. The SYNOPSIS shows how the widget IDs of the control areas (upper and lower) and footer container (panel) are obtained from the `PopupWindow` widget.

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

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

PopupWindow Coloration
Figure 2 illustrates the resources that affect the coloration of the `PopupWindow` widget.
Keyboard Traversal

The PopupWindow widget has a number of components which the user can traverse between. The buttons in the Lower Control Area and in the Settings Menu have the following mnemonics.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Button Name</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNapply</td>
<td>Apply</td>
<td>A</td>
</tr>
<tr>
<td>XtNreset</td>
<td>Reset</td>
<td>R</td>
</tr>
<tr>
<td>XtNresetFactory</td>
<td>Reset to Factory</td>
<td>F</td>
</tr>
<tr>
<td>XtNsetDefaults</td>
<td>Set Defaults</td>
<td>S</td>
</tr>
</tbody>
</table>

These mnemonics will be displayed in the button labels according to the value returned by OIQueryMnemonicDisplay(). The buttons are created with XtNtraversalOn set to True.

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

<table>
<thead>
<tr>
<th>Activation Type</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL_CANCEL</td>
<td>Popdown window</td>
</tr>
<tr>
<td>OL_DEFAULTACTION</td>
<td>Call OLActivateWidget for the default widget with parameter OL_SELECTKEY</td>
</tr>
<tr>
<td>OL_TOGGLEPUSHPIN</td>
<td>Toggle pushpin state</td>
</tr>
</tbody>
</table>

SUBSTRUCTURE
Lower Control Area and Upper Control Area components
Names: lower, upper
Class: ControlArea

The following resources are directed to both Control Area components. To set different values for the same resources in the different Control Areas, the application must access the resources using the appropriate Control Area widget IDs.

Application Resources

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNalignCaptions</td>
<td>XtCAlignCaptions</td>
<td>Boolean</td>
<td>✓</td>
<td>I</td>
</tr>
<tr>
<td>XtNcenter</td>
<td>XtCCenter</td>
<td>Boolean</td>
<td>FALSE</td>
<td>I</td>
</tr>
<tr>
<td>XtNhPad</td>
<td>XtCHPad</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>XtNhSpace</td>
<td>XtCHSpace</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>XtNlayoutType</td>
<td>XtCLayoutType</td>
<td>OlDefine</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>XtNmeasure</td>
<td>XtCMeasure</td>
<td>int</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>XtNsameSize</td>
<td>XtCSameSize</td>
<td>OlDefine</td>
<td>OL_COLUMNS</td>
<td>I</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvPad</td>
<td>XtCVPad</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>XtNvSpace</td>
<td>XtCVSpace</td>
<td>Dimension</td>
<td>4</td>
<td>I</td>
</tr>
</tbody>
</table>

† The default is TRUE for the Upper Control Area and FALSE for the Lower Control Area.

‡ The default is OL_FIXEDCOLUMNS for the Upper Control Area and OL_FIXEDROWS for the Lower Control Area.

Footer
Names: panel
Class: FooterPanel

RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNallowShellResize</td>
<td>XtCallowShellResize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G+*</td>
</tr>
<tr>
<td>XtNapply</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>I</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
</tbody>
</table>
# PopupWindow Shell (3W)

## PopupWindow Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtConsumeEvent</td>
<td>XiCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcreatePopupChildProc</td>
<td>XtCCreatePopupProc</td>
<td>XtCreatePopupProc</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XiCallbackList</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNfooterPanel</td>
<td>XtCFooterPanel</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNgeometry</td>
<td>XtCGeometry</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheightInc</td>
<td>XtCHeightInc</td>
<td>int</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfocusWidget</td>
<td>XtCFocusWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinput</td>
<td>XtCInput</td>
<td>Boolean</td>
<td>FALSE</td>
<td>G</td>
</tr>
<tr>
<td>XtNlowerControlArea</td>
<td>XtCLowerControlArea</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNmaxAspectX</td>
<td>XtCMaxAspectX</td>
<td>int</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxAspectY</td>
<td>XtCMaxAspectY</td>
<td>int</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxHeight</td>
<td>XtCMaxHeight</td>
<td>int</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxWidth</td>
<td>XtCMaxWidth</td>
<td>int</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminAspectX</td>
<td>XtCMinAspectX</td>
<td>int</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminAspectY</td>
<td>XtCMinAspectY</td>
<td>int</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminHeight</td>
<td>XtCMinHeight</td>
<td>int</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminWidth</td>
<td>XtCMinWidth</td>
<td>int</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNpopdownCallback</td>
<td>XtCCallback</td>
<td>XiCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNpopupCallback</td>
<td>XtCCallback</td>
<td>XiCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNpushpin</td>
<td>XtCPushpin</td>
<td>OIDefine</td>
<td>OL_OUT</td>
<td>G</td>
</tr>
<tr>
<td>XtNresetButton</td>
<td>XtCResetButton</td>
<td>Widget</td>
<td>NULL</td>
<td>G</td>
</tr>
<tr>
<td>XtNresetFactory</td>
<td>XtCCallback</td>
<td>XiCallbackList</td>
<td>NULL</td>
<td>I</td>
</tr>
<tr>
<td>XtNresizeCorners</td>
<td>XtCResizeCorners</td>
<td>Boolean</td>
<td>True</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsaveUnder</td>
<td>XtCSaveUnder</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNsetDefault</td>
<td>XtCCallback</td>
<td>XiCallbackList</td>
<td>NULL</td>
<td>I</td>
</tr>
<tr>
<td>XtNtitle</td>
<td>XtCTitle</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNupperControlArea</td>
<td>XtCUpperControlArea</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNverify</td>
<td>XtCCallback</td>
<td>XiCallbackList</td>
<td>NULL</td>
<td>I</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Position</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidthInc</td>
<td>XtCWidthInc</td>
<td>Position</td>
<td>-1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

The following resources provide the widget identifier for the buttons that are created in the PopupWindow widget. The application can add an accelerator to
these buttons, or change the default mnemonic by accessing the Button's resources.

**XtNfooterPanel**
This is the widget ID of the FooterPanel class composite child widget that handles the Footer; its value is available once the PopupWindow widget has been created. If the application wants a footer, it can add one to the composite identified by this resource.

**XtNlowerControlArea**
**XtNupperControlArea**
These are the widget IDs of the ControlArea class composite child widgets that handle the Lower Control Area and Upper Control Area, respectively. The application can use each widget ID to populate the PopupWindow with controls. These widget IDs are available once the PopupWindow widget has been created.

Any widgets of the class OblongButton added to the Lower Control Area are assumed to be window disposition controls; that is, when the end user activates one of them the PopupWindow widget should pop itself down, if allowed by the application and the state of the pushpin.

**XtNverify**
This resource defines the callbacks to be invoked when the PopupWindow attempts to pop itself down. The call_data parameter is a pointer to a variable of type Boolean. It is initially set to TRUE, and the application should set a value that reflects whether the pop-down is allowed. Typically, the application will use this to prevent a pop-down so that an error message can be displayed.

Since more than one callback routine may be registered for this resource, each callback routine can first check the value pointed to by the call_data parameter to see if a previous callback in the list has already rejected the pop-down attempt. If one has, the subsequent callback need not continue evaluating whether a pop-down is allowed. If the value is still TRUE after the last callback returns, the pop-down continues.

Since these callbacks are issued before the PopupWindow checks the state of the pushpin, the application should not assume that the pop-down will occur even though it has allowed it.
RectButton (3W)

NAME
RectButton – a primitive widget consisting of a label surrounded by a rectangular border

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <RectButton.h>

widget = XtCreateWidget(name, rectButtonWidgetClass, ...);

DESCRIPTION
RectButton Components
The RectButton widget implements one of the OPEN LOOK button widgets. It consists of a Label surrounded by a rectangular Border. The Border can change to reflect that the button may be a default of several buttons (double border), or represents a current state of an object (thick border), or represents a current state of one of several objects with different states (dimmed border).

Figure 1 shows several buttons, in normal, default, and current states (two versions).

<table>
<thead>
<tr>
<th>Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Default Value</td>
</tr>
<tr>
<td>Current Value</td>
<td>Current Value</td>
</tr>
</tbody>
</table>

Figure 1. Rectangular Buttons

Use of Rectangular Buttons
Rectangular buttons are not used alone but in one of the Exclusives or Non-exclusives composite widgets for implementing a one-of-many or several-of-
many selection. Making this widget a child of a different composite widget will not produce an error message, but proper behavior is not guaranteed.

Toggling the State
A RectButton widget has two states: "set" and "not set". (When set, its border is thickened.) Toggling this state alternates a resource (XtNset) between "true" and "false" and starts an action associated with the button. The RectButton widget is typically toggled by the user using SELECT or MENU, except that it is possible to disable user toggling of a button that is already set.

Work in Exclusives/Nonexclusives Only
If a RectButton widget is not the child of an Exclusives or NonExclusives widget, it will toggle between set and unset status, as in Nonexclusives.

Rectangular Buttons in a Pop-up Menu
Entering a rectangular button while MENU is depressed changes the appearance of the button from unset to set state or vice versa, to reflect the state the button would be in if MENU were released. Releasing MENU toggles the state associated with the button. Leaving the button before releasing MENU restore the original state appearance and does not toggle the button.

Rectangular Buttons Not in a Pop-up Menu
Clicking SELECT on a rectangular button toggles the state associated with it. Pressing SELECT, or moving the pointer into the button while SELECT is pressed, changes the border from unset to set state or vice versa, to reflect the state the button would be in if SELECT were released. Releasing SELECT toggles the state. Moving the pointer off the button before releasing SELECT restores the state appearance and does not toggle the button.

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

XtNdím
Range of Values:
TRUE
FALSE

If this resource is TRUE, then the button border is dimmed to show that the button represents the state of one or more of several objects that, as a group, are in different states.

XtNlabel
This resource is a pointer to the text for the Label. This resource is ignored if the XtNlabelType resource has the value OL_IMAGE.

XtNlabelImage
This resource is a pointer to the image for the Label. This resource is ignored unless the XtNlabelType resource has the value OL_IMAGE.

If the image is of type XYBitmap, the image is highlighted when appropriate by reversing the 0 and 1 values of each pixel (for example, by "xor'ing" the image data). If the image is of type XYPixmap or ZPixmap, the image is not highlighted, although the space around the image inside the Border is highlighted.
If the image is smaller than the space available for it inside the Border and `XtNlabelTile` is FALSE, the image is centered vertically and either centered or left-justified horizontally, depending on the value of the `XtNlabelJustify` resource. If the image is larger than the space available for it, it is clipped so that it does not stray outside the Border. If the `XtNdefault` resource is TRUE so that the Border is doubled, the space available is that inside the inner line of the Border.

**XtNlabelJustify**

Range of Values:

- OL_LEFT/"left"
- OL_CENTER/"center"

This resource dictates whether the Label should be left-justified or centered within the widget width.

**XtNlabelTile**

Range of Values:

- TRUE
- FALSE

This resource augments the `XtNlabelImage/XtNlabelPixmap` resource to allow tiling of the sub-object’s background. For an image/pixmap that is smaller than the sub-object’s background, the label area is tiled with the image/pixmap to fill the sub-object’s background if this resource is TRUE; otherwise, the label is placed as described by the `XtNlabelJustify` resource.

The `XtNlabelTile` resource is ignored for text labels.

**XtNlabelType**

Range of Values:

- OL_STRING/"string"
- OL_IMAGE/"image"

This resource identifies the form that the Label takes. It can have the value `OL_STRING` or `OL_IMAGE` for text or image, respectively.

**XtNrecomputeSize**

Range of Values:

- TRUE
- FALSE

This resource indicates whether the RectButton widget should calculate its size and automatically set the `XtNheight` and `XtNwidth` resources. If set to TRUE, the RectButton widget will do normal size calculations that may cause its geometry to change. If set to FALSE, the RectButton widget will leave its size alone; this may cause truncation of the visible image being shown by the RectButton widget if the fixed size is too small, or may cause padding if the fixed size is too large. The location of the padding is determined by the `XtNlabelJustify` resource.

**XtNselect**

This is the list of callbacks invoked when the widget is selected.
RectButton (3W)

**XtNset**

Range of Values:

```
TRUE
FALSE
```

This resource reflects the current state of the button. The button’s border is thickened to show a TRUE state.

**XtNunselect**

This is the list of callbacks invoked when a RectButton widget is toggled into the “unset” mode by the end user to make XtNset be FALSE. Note that simply setting XtNset to FALSE with a call to XtSetValues() does not issue the XtNunselect callbacks.

**XtNdim, XtNdefault, XtNset**

The XtNdim, XtNdefault, and XtNset resources can be set independently; however, all these states cannot be reflected in the visual appearance of the rectangular button, as the state table in Figure 3 shows.

<table>
<thead>
<tr>
<th>XtNset</th>
<th>XtNdefault</th>
<th>XtNdim</th>
<th>Border appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>TRUE</td>
<td>Dimmed</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>FALSE</td>
<td>Thickened</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Open</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>Open</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>Normal</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>Normal</td>
</tr>
</tbody>
</table>

Figure 3. Rectangular Button Appearance when Set/Default/Dim

**Label Appearance**

The XtNWidth, XtNheight, XtNrecomputeSize, and XtNlabelJustify resources interact to produce a truncated, clipped, centered, or left-justified label as shown in Figure 4.
Figure 4. Label Appearance

When the label is centered or left-justified, the extra space is filled with the background color of the RectButton widget, as determined by the XtNbackground and XtNbackgroundPixmap resources.

When the label is truncated, a solid-black triangle is inserted to show that part of the label is missing. The triangle requires that more of the label be truncated than would otherwise be necessary. If the width of the button is too small to show even one character with the triangle, only the triangle is shown. If the width is so small that the entire triangle cannot be shown, the triangle is clipped on the right.

See also the XtNlabelTile resource for how it affects the appearance of a label.

RectButton Coloration

Figure 2 illustrates the resources that affect the coloration of the RectButton widget.

On a monochrome display, the RectButton widget indicates that it has input focus by inverting the foreground and background colors of the control.

On color displays, when the RectButton widget receives the input focus, the background color is changed to the input focus color set in the XtNinputFocusColor resource.
**RectButton (3W)**

**Exceptions:**
- If the input focus color is the same as the font color for the control labels, then the coloration of the active control is inverted.
- If the input focus color is the same as the Input Window Header Color and the active control is in the window header, then invert the colors.
- If the input focus color is the same as the window background color, then the RectButton widget inverts the foreground and background colors when it has input focus.

**Figure 2. Rectangular Button Coloration**

**Keyboard Traversal**

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

The RectButton widget responds to the following keyboard navigation keys:
- **NEXT FIELD** moves to the next traversable widget in the window
- **PREV_FIELD** moves to the previous traversable widget in the window
- **MOVEUP** moves to the RectButton above the current widget in the Nonexclusives or Exclusives composite
- **MOVEDOWN** moves to the RectButton below the current widget in the Nonexclusives or Exclusives composite
RectButton (3W)

LEFT moves to the RectButton to the left of the current widget in the Nonexclusives or Exclusives composite
MOVERIGHT moves to the RectButton to the right of the current widget in the Nonexclusives or Exclusives composite
NEXTWINDOW moves to the next window in the application
PREVWINDOW moves to the previous window in the application
NEXTAPP moves to the first window in the next application
PREVAPP moves to the first window in the previous application

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

<table>
<thead>
<tr>
<th>RectButton Activation Types</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL_MENUDEFAULTKEY</td>
<td>Set the sub-object as the shell’s default object (if on a menu)</td>
</tr>
<tr>
<td>OL_SELECTKEY</td>
<td>Call its callbacks</td>
</tr>
</tbody>
</table>

Display of Keyboard Mnemonic

The RectButton widget displays the mnemonic accelerator for its child as part of its label. If the mnemonic character is in the label, then that character is highlighted according to the value of the application resource XtNshowMnemonics. If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the application resource XtNshowMnemonics.

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

Display of Keyboard Accelerators

The RectButton widget displays the keyboard accelerator as part of its label. The string in the XtNacceleratorText resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

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

RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
</tbody>
</table>
### RectButton Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNdefault</td>
<td>XtCDefault</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNdim</td>
<td>XtCDim</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct *</td>
<td>(OPEN LOOK font)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black*</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI*</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension (calculated)</td>
<td></td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>(class name)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelImage</td>
<td>XtCLabelImage</td>
<td>XImage *</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelJustify</td>
<td>XtCLabelJustify</td>
<td>OlDefine OL_LEFT</td>
<td></td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelTile</td>
<td>XtCLabelTile</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlabelType</td>
<td>XtCLabelType</td>
<td>int</td>
<td>OL STRING</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNselect</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNset</td>
<td>XtCSet</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNunselect</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension (calculated)</td>
<td></td>
<td>SGI</td>
</tr>
</tbody>
</table>

#### XtNdefault

Range of Values:

- **TRUE**
- **FALSE**

If this resource is TRUE, the Border is doubled to two lines, to show that the button is the default choice of one or more buttons.
NAME
Scrollbar – moves or scrolls the view of an associated pane

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Scrollbar.h>

static Widget scrollbar, menupane, w;
Arg args[1];

scrollbar = XtCreateWidget(name, scrollbarWidgetClass, ...);

/*use the following instruction to add a button to the scrollbar menu*/

XtSetArg(args[0], XtNmenuPane, &menupane);
XtGetValues(scrollbar, args, 1)
w = XtCreateWidget(name, widget-class, menupane, ...);

DESCRIPTION
Scrollbar Components
Each full scrollbar has the following parts:
— Top (Left) Anchor
— Bottom (Right) Anchor
— Up (Left) Arrow
— Down (Right) Arrow
— Drag Area
— Elevator
— Cable
— Proportion Indicator
— Scrollbar Menu
— Page Indicator (optional)

A scrollbar consists of two Anchors, an Elevator, a Cable, a Proportion Indicator, a Menu, and optionally a Page Indicator. The Anchors are located at both ends of the cable. They are used to move the view to the corresponding extreme of the item or list of items being viewed. The Elevator, which slides along the length of the cable, consists of the Up Arrow, Down Arrow, and a Drag Area in the middle for moving the view. The arrow boxes are used to move the view in the direction of the arrow by one unit of granularity. The Drag Area is used to move the view by tracking the position of the mouse pointer relative to the scrollbar. The Proportion Indicator moves along with the Elevator to indicate the size of the view and its position relative to the entire item or list of items being viewed. The optional Page Indicator located next to the Drag Area indicates the page number of the content being viewed. The Page Indicator will be displayed only when the SELECT button is pressed in the Drag Area.
Figure 1. Horizontal Scroll Bar
Because a scrollbar can be seen and used horizontally as well as vertically, the parts Top Anchor and Bottom Anchor have the aliases Left Anchor and Right Anchor, respectively.

Each scrollbar is associated with a Content, as defined by the application. The Content is composed of Units (for example, lines of text) that are visible in a viewing area. For a scrollbar to be useful, the Content typically has more Units than can fit in the viewing area. Hence, "scrolling" the Content brings Units into view as other Units move out of view. The amount of the Content that is visible at one time is called a pane in the descriptions below.

**Abbreviated Scrollbar**

The Scrollbar widget responds to a parent's request to resize smaller by shortening the Cable (and Proportion Indicator) but leaving the other elements full-sized. The Scrollbar widget will eliminate the Cable and drag area entirely, if necessary, to meet a resize request. These abbreviated scrollbars are shown in Figure 3.
Minimum Scrollbar

If necessary, the Scrollbar widget will eliminate the anchors (in addition to the Cable and drag area) to meet a resize request to form a minimum scrollbar.

Elevator Motion

As visual feedback to the user, the Elevator moves up and down (or left and right) along the line of the Cable as the Content scrolls or changes panes. The range of motion of the Elevator is not necessarily the full distance between the Anchors. The application decides how far the Elevator can be moved by evaluating each attempt to move it.

The user manipulates the scrollbar by pressing or clicking SELECT. The action performed depends on the position of the pointer and whether the application is willing to scroll the Content.

Scrolling One Unit

Clicking SELECT on one of the Up, Down, Left, or Right Arrows moves the Elevator in the direction of the arrow, moves the pointer to stay on the Arrow, and changes the Content to move one Unit out of view and another Unit into view, such that the view scrolls in the opposite direction of the Elevator motion. If the application cannot scroll this time, the Elevator and pointer do not move, and the view does not change.
Pressing SELECT on an Arrow repeats the action described above. When SELECT is clicked or pressed, the Arrow highlights while the scrolling action takes place. If SELECT is pressed, the highlighting stays until SELECT is released.

When the Elevator has reached the end of a Cable, the Arrow in that direction is made inactive.

**Scrolling Several Units**

Dragging SELECT on the Drag Area moves the Elevator along the Cable, to track the component of the pointer motion parallel to the Cable. The Content scrolls in the opposite direction, bringing one or more Units into view as other Units move out of view.

If granularity is enforced and the Elevator is moved to a position that represents a non-integral number of Units, the closest integral number of Units is considered instead. If granularity is not enforced, the Elevator is moved by the non-integral number of Units. The `XtSliderMoved` callback allows the application to enforce granularity.

When the application reaches the limit that it can scroll, the view no longer changes and the Elevator stops moving.

While dragging SELECT, the Drag Area highlights. The pointer is constrained to stay within the Drag Area as the Elevator moves.

**Scrolling to Limits**

Clicking SELECT on one of the Top, Bottom, Left, or Right Anchors causes the view of the Content to change to the top-most, bottom-most, left-most, or right-most pane, respectively, and moves the Elevator to the limit in the direction of the Anchor. If the Elevator is already at the limit, nothing happens.

Clicking SELECT on an Anchor highlights the Anchor while the scrolling action takes place.

**Scrolling a Pane of Units**

Clicking SELECT on the Cable above/left-of or below/right-of the Elevator causes the view of the Content to change to the previous or next pane, respectively. The pointer is moved along the direction of the Elevator travel to keep it off the Elevator. If only a partial pane remains before the limit of the Content is reached, the effect is as if the user clicked SELECT on the corresponding Anchor. If the application cannot move to another pane, the view does not change and the Elevator and pointer do not move.

Pressing SELECT on the Cable repeats the action described above.

**Elevator Approaching Limits**

The application calibrates the scrollbar so that the position of the elevator on the scrollbar is in units useful to the application. In general, these units will not be pixels or points. If the scrollbar is close enough to an Anchor, the separation in application units may equate to zero pixels, because of the discrete nature of pixels. Here, the elevator is kept away from the Anchor so that two points of the Cable length are visible. The Elevator is placed at the limit of motion only when the user explicitly moves the elevator to an Anchor by clicking SELECT on the Anchor, or drags the Elevator until it reaches the limit.
Indicating View Proportion

The Proportion Indicator gives a gross measure of what part of the Content is in view. Its size relative to the length of the Cable is the same as the size of the pane relative to the size of the Content. However, the scrollbar widget does not maintain this relation but relies on the application to provide the length of the Proportion Indicator.

The Proportion Indicator moves with the Elevator such that both reach the limits together. When the Content is scrolled to the beginning, the proportion indicator and the elevator align at the left or top end of the scrollbar, as in Figure 4. When the Content is scrolled to the end, the proportion indicator and the elevator align at the right or bottom end of the scrollbar, as in Figure 5. For intermediate positions, the Elevator is positioned proportionally between the ends of the Proportion Indicator. Thus, as the Content is scrolled at a constant rate (for example, by dragging SELECT), the Elevator creeps from one end of the Proportion Indicator to the other at a constant rate.

![Figure 4. Elevator and Proportion Indicator at Left/Top Limits](image-url)
Scrollbar Menu

The Scrollbar Menu (not shown in the figures) pops up when the user presses MENU anywhere over the scrollbar widget. The menu has three default choices depending on the scrollbar orientation.

Here to Top

Here to Left

This choice scrolls the Content so that the Unit next to the pointer is placed at the top or left of the viewing area.

Top to Here

Left to Here

This choice scrolls the Content so that the Unit at the top or left of the viewing area is placed next to the pointer.

Previous

This choice scrolls the Content to restore the previous view. The scrollbar widget remembers only the last two scroll positions, so repeated access to this choice alternates the Content between two views. Note: If the Scrollbar menu was invoked from the keyboard, then only the "Previous" button is usable. In this case, the "Here To" button and the "To Here" button are not sensitive.

An application can add choices to this menu, using the same technique for populating other menus (see MENU WIDGET(3W)). The ID of the menu widget is available as a resource of the scrollbar.
Scrollbar Coloration

When the Scrollbar widget receives the input focus through keyboard traversal, the background color of the widget changes to the input focus color, found in the resource `XtNinputFocusColor`. If the user traverses out of the Scrollbar widget, the background of the widget reverts to its original background color.

**EXCEPTION:** If the input focus color is the same as either the foreground or background color, then the widget shows input focus by switching the background and foreground colors.

Figure 6 illustrates the resources that affect the coloration of the Scrollbar widget.

![Scrollbar Coloration Diagram](image)

Figure 6. Scrollbar Coloration

Keyboard Traversal

The Scrollbar’s default values of the `XtNtraversalOn` resource is True.

The user can operate the Scrollbar by using the keyboard to move the Elevator and access the Anchors. The following keys manipulate the Scrollbar:

- **SCROLLUP and SCROLLDOWN** (SCROLLLEFT and SCROLLRIGHT for horizontal Scrollbars) move the Elevator one Unit in the given direction. The Content changes to move one Unit out of view and another Unit into view, such that the view scrolls in the opposite direction of the Elevator motion. If the application cannot scroll at this time, the Elevator does not move and the view does not change.
The appropriate arrow in the Elevator highlights while the scrolling action takes place.

- SCROLLTOP and SCROLLBOTTOM (SCROLLLEFTEDGE and SCROLLRIGHTEDGE for horizontal Scrollbars) cause the view of the Content to change to the top-most, bottom-most, left-most, or right-most pane respectively. The Elevator moves to the limit in the direction of the Anchor.

These keys cause the appropriate Anchor to highlight while the scrolling action takes place.

- PAGEUP and PAGEDOWN (PAGELEFT and PAGERIGHT for horizontal Scrollbars) cause the view of the Content to change to the previous or next pane, respectively.

- MENUKEY posts the Scrollbar's Menu.

<table>
<thead>
<tr>
<th>Vertical Scrollbar Activation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>OL_MENUKEY, or</td>
</tr>
<tr>
<td>OL_VSBMENU</td>
</tr>
<tr>
<td>OL_PAGEUP</td>
</tr>
<tr>
<td>OL_PAGEDOWN</td>
</tr>
<tr>
<td>OL_SCROLLUP</td>
</tr>
<tr>
<td>OL_SCROLLDOWN</td>
</tr>
<tr>
<td>OL_SCROLLTOP</td>
</tr>
<tr>
<td>OL_SCROLLBOTTOM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Horizontal Scrollbar Activation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>OL_MENUKEY, or</td>
</tr>
<tr>
<td>OL_HSBMENU</td>
</tr>
<tr>
<td>OL_PAGELEFT</td>
</tr>
<tr>
<td>OL_PAGERIGHT</td>
</tr>
<tr>
<td>OL_SCROLLRIGHT</td>
</tr>
<tr>
<td>OL_SCROLLLEFT</td>
</tr>
<tr>
<td>OL_SCROLLRIGHTEDGE</td>
</tr>
<tr>
<td>OL_SCROLLLEFTEDGE</td>
</tr>
</tbody>
</table>

The Scrollbar widget responds to the following keyboard navigation keys:

- NEXT FIELD, MOVEDOWN, and MOVERIGHT move to the next traversable widget in the window
- PREV FIELD, MOVEUP, and MOVELEFT move to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
— NEXTAPP moves to the first window in the next application
— PREVAPP moves to the first window in the previous application

Operating the Scrollbar with keyboard disables any pointer warping.

Scrollbar Menu

The default choices in the Scrollbar Menu are created with XtNtraversalOn set to True and XtNmemonic set to the first character of their label.

When the Scrollbar Menu is posted via keyboard traversal, the "Here to Top" and "Top to Here" buttons are not sensitive. These buttons depend on the position of the pointer when the menu is posted, and so they are not applicable when the menu is posted from the keyboard.

Display of Keyboard Mnemonic

The Scrollbar does not display the mnemonic accelerator. If the Scrollbar is the child of a Caption widget, the Caption widget can be used to display the Scrollbar's mnemonic.

Display of Keyboard Accelerators

The Scrollbar does not respond to a keyboard accelerator because clicking the SELECT button on a Scrollbar activates depending on the pointer position. So, the Scrollbar does not display a keyboard accelerator.

SUBSTRUCTURE

Scrollbar Menu component

Name: ScrollMenu
Class: Menu

<table>
<thead>
<tr>
<th>Application Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>*XtNcenter</td>
</tr>
<tr>
<td>*XtNhPad</td>
</tr>
<tr>
<td>*XtNhSpace</td>
</tr>
<tr>
<td>*XtNlayoutType</td>
</tr>
<tr>
<td>*XtNmeasure</td>
</tr>
<tr>
<td>XtNpushpin</td>
</tr>
<tr>
<td>XtNpushpinDefault</td>
</tr>
<tr>
<td>*XtNsameSize</td>
</tr>
<tr>
<td>XtNtitle</td>
</tr>
<tr>
<td>*XtNvpad</td>
</tr>
<tr>
<td>*XtNvSpace</td>
</tr>
</tbody>
</table>

*See the Menu and ControlArea widgets for the descriptions of these resources.
## RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCAncestorsensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultbackground</td>
<td>SI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCBackgroundPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SI†</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcurrentPage</td>
<td>XtCCurrentPage</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCDestroyCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNdragCType</td>
<td>XtCDragCType</td>
<td>OlDefine</td>
<td>OL_CONTINUOUS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgranularity</td>
<td>XtCGranularity</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinitialDelay</td>
<td>XtCInitialDelay</td>
<td>int</td>
<td>500</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmenuItem</td>
<td>XtCMenuPane</td>
<td>Widget</td>
<td>(none)</td>
<td></td>
</tr>
<tr>
<td>XtNorientation</td>
<td>XtCOrientation</td>
<td>OlDefine</td>
<td>OL_VERTICAL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNproportionLength</td>
<td>XtCProportionLength</td>
<td>int</td>
<td>(variable)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrepeatRate</td>
<td>XtCRepeatRate</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNshowPage</td>
<td>XtCShowPage</td>
<td>OlDefine</td>
<td>OL_NONE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsliderMax</td>
<td>XtCSliderMax</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsliderMin</td>
<td>XtCSliderMin</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsliderMoved</td>
<td>XtCSliderMoved</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsliderValue</td>
<td>XtCSliderValue</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNstopPosition</td>
<td>XtCStopPosition</td>
<td>OlDefine</td>
<td>OL_ALL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNbackground**

Whenever this resource is changed, the Anchors, the Elevator, and the Proportion Indicator will be redisplayed with the new color.
**XtNcurrentPage**
This resource is typically not selectable. However, there are two exceptions. If **XtNshowPage** is set via `SetValues()`, **XtNcurrentPage** should also be set in the same call. If **XtNshowPage** is already set, and later **XtNsliderValue** is changed via `SetValues()`, **XtNcurrentPage** should also be set in the same call.

**XtNdndCBType**
Range of values:
- OL_CONTINUOUS/"continuous"
- OL_GRANULARITY/"granularity"
- OL_RELEASE/"release"

This resource determines the frequency of issuing **XtNnestedMovement** callbacks. If set to OL_CONTINUOUS, callbacks will be issued continuously (just like in Xt+ 2.0). If set to OL_GRANULARITY, callbacks will only be issued when the drag box crosses any granularity positions. If set to OL_RELEASE, callback will only be issued once when the SELECT button is released.

**XtNggranularity**
Range of Values:
\[ 1 \leq \text{XtNggranularity} \leq \text{XtNsliderMax} - \text{XtNsliderMin} \]

Clicking or pressing SELECT on an Up, Left, Down, or Right Arrow attempts to change the position of the Elevator by the distance given in this resource. Normally, the drag operation does not honor granularity unless enforcement is set in the XtNnestedMovement callback procedure.

**XtNinitialDelay**
Range of Values:
\[ 0 < \text{XtNinitialDelay} \]

This resource gives the time, in milliseconds, before the first action occurs when SELECT is pressed on the Cables or Arrows. Note that millisecond timing precision may not be possible for all implementations, so the value may be rounded up to the nearest available unit by the toolkit.

**XtNmenuPane**
This is the widget where scrollbar menu items can be attached; its value is set once the scrollbar is created. Menu items can be added to this widget just as they are to a menu pane for a Menu or MenuButton widget.

The menu initially contains the items
- Here to Top (Here to Left)
- Top to Here (Left to Here)
- Previous

If these items are removed from the menu by the application, a warning is generated (see Error(3W) in the Convenience Routines section).

**XtNorientation**
Range of Values:
- OL_HORIZONTAL/"horizontal"
- OL_VERTICAL/"vertical"
This resource defines the direction for the visual presentation of the widget. This resource cannot be changed via `SetValues()`.

**XtNproportionLength**  
Range of Values:  
\[ 1 \leq \text{XtNproportionLength} \leq (\text{XtNsliderMax} - \text{XtNsliderMin}) \]  
Default:  
\[(\text{XtNsliderMax} - \text{XtNsliderMin})\]

This resource gives the size of the Proportion Indicator. The application uses the `XtNsliderMax` and `XtNsliderMin` resources to calibrate the scrollbar, making its overall length correspond to the overall length of the Content, and uses the `XtNproportionLength` resource to indicate how much of the Content is visible.

While this resource gives the overall length of the Proportion Indicator, the Elevator always covers part of it. If the Elevator would completely hide the Proportion Indicator, 3-point sections of it are shown above and below (or left of and right of) the Elevator. If the Elevator is too close to an Anchor to show all of a 3-point section, as much as possible of the section is shown on that side (this may be a zero-length section).

For example, if you have 100 items to be displayed and only one item is viewable in the pane at a time, then set `XtNsliderMin` to 0, `XtNsliderMax` to 100, and `XtNproportionLength` to 1. The possible sliderValues are from 0 to 99, inclusive. Another example, 100 items, but 25 items viewable at a time, then set `XtNsliderMin` to 0, `XtNsliderMax` to 100, `XtNproportionLength` to 25. The possible sliderValues are from 0 to 75, inclusive.

**XtNrepeatRate**  
Range of Values:  
\[ 0 < \text{XtNrepeatRate} \]

This resource gives the time, in milliseconds, between repeated actions when `SELECT` is pressed on the Cables or Arrows. Note that millisecond timing precision may not be possible for all implementations, so the value may be rounded up to the nearest available unit by the toolkit.

**XtNshowPage**  
Range of Values:  
- `OL_NONE`/ "none"  
- `OL_LEFT`/ "left"  
- `OL_RIGHT`/ "right"

If `XtNshowPage` changes from `OL_NONE` to one of the other values, a pop-up window for the page indicator is created. If the value changes to `OL_NONE`, then the pop-up is destroyed.

This value is checked when dragging is initiated. If it is not set to `OL_NONE`, the page indicator will be popped to the screen. While dragging, the page number in the indicator is constantly updated.

**NOTE:**  
The page indicator feature is not popped to the screen when using the keyboard rather than the mouse for drag operations.
XtN sliderMax
XtN sliderMin
Range of Values:

XtN sliderMin < XtN sliderMax

These two resources are used to calibrate the Scrollbar widget. An application should set their values to correspond to the range of the Content, and should set the value of the XtN proportionLength resource to the length of the view into the Content. This calibrates the scrollbar.

The Scrollbar uses the calibration to convert the pixel location of the Elevator into a value in the range

XtN sliderMin \leq \text{range} \leq \text{XtN sliderMax} - \text{XtN proportionLength}

The explanation for this range relation follows: First, an application calibrates the scrollbar as described above, so that XtN sliderMin and XtN sliderMax span the length of the Content and XtN proportionLength gives the length of the view of the Content. Consider that the Elevator tracks a fixed position in the view; the position is arbitrary, but remains the same as the view is scrolled over the Content. This can be the first line in the view. As Figure 7 shows, when the view is at the top of the Content, the Elevator is at the top of the scrollbar and the calibrated position of the first line is XtN sliderMin. However, when the view is at the bottom of the content, the Elevator is at the bottom of the scrollbar and the calibrated position of the first line is XtN sliderMax - XtN proportionLength.

Figure 7. Elevator Range of Movement
This resource defines the callback lists used when the scrollbar is manipulated in various ways. The Scrollbar widget passes the final location of the Elevator, as an integer between `XtNslderMin` and `XtNslderMax` inclusive, in a structure pointed to by the `call_data` parameter. The structure, `OIScrollbarVerify`, looks like this:

```c
typedef struct _OIScrollbarVerify {
    int new_location;
    int new_page;
    Boolean ok;
    int slider_min;
    int slider_max;
    int delta;
    Boolean move_cb_pending;
} OIScrollbarVerify;
```

**new_location**

When the `XtNslderMoved` callbacks are made, the `new_location` member gives the position of the attempted scroll. This will be the new value of the `XtNslderValue` resource if the scroll attempt is successful; however, the `XtNslderValue` resource is not updated until after the callbacks return.

**new_page**

This will be the new value of the `XtNCurrentPage` resource if the scroll attempt is successful. `new_page` is used to set the page number. To see the page number, you have to set `XtNshowPage` to OL_LEFT or OL_RIGHT.

**ok**

The `ok` member of this structure is initially set to TRUE; the application should set a value that reflects whether the scroll attempt is allowed. Since more than one callback routine may be registered for these resources, each callback routine can first check the `ok` member to see if a previous callback routine in the list has already rejected the scroll attempt. The scrollbar will complete the scroll attempt only if, after the last callback has returned, the `ok` member is still TRUE.

If the `ok` member is FALSE after the last callback returns, the Scrollbar restores the Elevator to the position it was in before the user attempted to move it. This is required only when the Elevator has been dragged. The Scrollbar does not move the Elevator for other scrollbar manipulations until the scroll attempt has been verified.

**slidermin**

**slidermax**

These are the same values as in the `XtNslderMin` and `XtNslderMax` resources.

**delta**

This is the distance between the new scroll position and the old, as a signed value:

```
delta = new_location - old location
```
A callback can change the `new_location` value to reflect a partial scroll. For example, if the scrolling granularity causes a scroll attempt past the end of an application's partially full buffer, the application should adjust `new_location` to a value representing the end of the buffer. The adjusted value must lie between the values present before the attempted scroll and the new values given in the `OlScrollbarVerify` structure.

**move_cb_pending**

The boolean, `move_cb_pending`, is set to TRUE, if more callbacks are pending. If an application received a callback with this set to TRUE, it is guaranteed that a callback with `move_cb_pending` set to FALSE will follow shortly, before or when an operation is completed. Currently, `move_cb_pending` is set to TRUE only during a drag operation.

The `XtNsliderMoved` callbacks are issued when the Elevator position has been conditionally changed by the user:

- clicking or pressing SELECT on the Up/Left or Down/Right arrow buttons;
- moving the Elevator to a new position by dragging SELECT on the Drag Area;
- clicking SELECT on the Top/Left or Bottom/Right Anchors;
- clicking or pressing SELECT on the Cable.

**XtNSliderValue**

Range of Values:

```
XtNSliderMin ≤ XtNSliderValue ≤ XtNSliderMax - XtNproportionLength
```

This resource gives the current position of the Elevator. The `Scrollbar` widget keeps this resource up to date.

**XtNstopPosition**

Range of values:

```
OL_ALL/"all"
OL_GRANULARITY/"granularity"
```

This resource determines the disposition of the drag box at the end of an drag operation. If set to `OL_ALL`, upon the release of the SELECT button in a drag operation, the drag box will be positioned at where it stops. If set to `OL_GRANULARITY`, the drag box will snap to the nearest granularity position.
NAME

ScrolledWindow – used as the basis for implementing a scrollable pane

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <ScrolledWi.h>

static Widget scrolledwindow, controlarea1, controlarea2, w;
Arg args[2];
scrolledwindow = XtCreateWidget(name, scrolledWindowWidgetClass, ...

/*Use the following instructions to add two buttons to the
scrolling window. */
XtSetArg(args[0], XtNhMenuPane, &controlarea1);
XtSetArg(args[1], XtNvMenuPane, &controlarea2);
XtGetValues(scrolledwindow, args, 2);
w = XtCreateWidget(name, widget-class, ...

w = XtCreateWidget(name, widget-class, controlarea1, ...);
w = XtCreateWidget(name, widget-class, controlarea2, ...);
```

DESCRIPTION

No Text or Graphics Semantics

The ScrolledWindow can be used as the basis for implementing an OPEN LOOK scrollable text or graphics pane. However, it has no innate text or graphics semantics. "Window" does not refer to an OPEN LOOK pop-up window or base window; it is a general term used because the ScrolledWindow widget provides a "window" onto a larger widget.

ScrolledWindow Components

The ScrolledWindow widget has the following components:

- Vertical Scrollbar (typically)
- Horizontal Scrollbar (typically)
- Content (not all visible)
- View of the Content (visible part of Content)
- View Border
View Border
The View Border is a 1-point outline around the View of the Content.

View onto Larger Data Display
The `ScrolledWindow` widget incorporates the features of the `ScrollBar` class of widgets to implement a visible window (the View of the Content) onto another, typically larger, data display (the Content). The View of the Content can be scrolled through the Content using the scroll bars.

Child Widget as Content
To use the `ScrollWindow`, the application creates a widget capable of displaying the entire Content as a child of the `ScrollWindow` widget. The `ScrollWindow` widget positions the child widget "within" the View of the Content, and creates scroll bars for the horizontal and/or vertical dimensions, as needed. When the end user performs some action on the scroll bars, the child widget will be repositioned accordingly within the View of the Content.

The word "within" is used strictly in the widget sense: the larger child widget is positioned within the smaller View of the Content part of the `ScrollWindow` widget, which necessarily forces the child widget to display only the visible part of itself. The protocol for this is through normal widget geometry interactions.
Upper Left Corner Fixed on Resize
If the ScrolledWindow widget is resized, the upper left corner of the View stays fixed over the same spot in the Content, unless this would cause the View to extend past the right or bottom edge of the Content. If necessary, the upper left corner will shift left or up only enough to keep the View from extending past the right or bottom edge.

View Never Larger than Content
The View of the Content is never made larger than needed to show the Content. Unless forced to appear, a scrollbar is removed from the side where it is no longer needed. Remaining scrollbars stay a fixed distance from the View.

Scrolling Sensitivity
The scrollbars are configured to scroll integer values, in pixels, through the width and length of the Content. This allows the finest degree of control of the positioning of the View of the Content. However, the application can set the step rate through these values to avoid a large number of view updates as the end user scrolls through the Content.

ScrolledWindow Coloration
Figure 2 illustrates which resources affect the coloration of the ScrolledWindow widget.

Figure 2. Scrolled Window Coloration
Application Controlled Scrolling

The ScrolledWindow widget also provides support for the application to control the scrolling of the content data within the view. In this mode of operation, the application creates a content window no larger than the view window. The application monitors user interaction with the Scrollbars and displays the appropriate data in the content window.

This mode of operation supports the scrolling of large amounts of data such as text.

The application specifies this mode of operation by setting the XtNVAutoScroll and/or XtNhAutoScroll resources to FALSE. Normally, these settings are combined with the setting of the XtNVSliderMoved and/or XtNhSliderMoved callbacks. Also, the application will specify an XtNcomputeGeometries callback which is used to layout the ScrolledWindow.

Keyboard Traversal

The ScrolledWindow controls the keyboard traversal between the Content, the Horizontal Scrollbar, and the Vertical Scrollbar. The Scrollbars that are created by the ScrolledWindow have the XtNtraversalOn resource set to False. A Content widget added to the ScrolledWindow with traversal enabled will be added to the traversable widgets in the window with the Scrollbars so that the user can move between them with the NEXT FIELD (or MOVEUP or MOVELEFT) and PREV FIELD (or MOVEDOWN or MOWERIGHT) keys.

<table>
<thead>
<tr>
<th>Scrolled Window Activation Types (if it has a Vertical Scrollbar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>OL_VSBMENU</td>
</tr>
<tr>
<td>OL_PAGEUP</td>
</tr>
<tr>
<td>OL_PAGEDOWN</td>
</tr>
<tr>
<td>OL_SCROLLUP</td>
</tr>
<tr>
<td>OL_SCROLLDOWN</td>
</tr>
<tr>
<td>OL_SCROLLTOP</td>
</tr>
<tr>
<td>OL_SCROLLBOTTOM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scrolled Window Activation Types (if it has a Horizontal Scrollbar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>OL_HSBMENU</td>
</tr>
<tr>
<td>OL_PAGELEFT</td>
</tr>
<tr>
<td>OL_PAGERIGHT</td>
</tr>
<tr>
<td>OL_SCROLLRIGHT</td>
</tr>
<tr>
<td>OL_SCROLLLEFT</td>
</tr>
<tr>
<td>OL_SCROLLRIGHTEDGE</td>
</tr>
<tr>
<td>OL_SCROLLLEFTEDGE</td>
</tr>
</tbody>
</table>
ScrolledWindow (3W)

SUBSTRUCTURE
Vertical Scrollbar and Horizontal Scrollbar components
Names: HScrollbar, VScrollbar
Class: Scrollbar

See the regular resource list for alternate names used for some key Scrollbar resources.

RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNalignHorizontal</td>
<td>XtCAlignHorizontal</td>
<td>int</td>
<td>OL_BOTTOM</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNalignVertical</td>
<td>XtCAlignVertical</td>
<td>int</td>
<td>OL_RIGHT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultbackground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcomputeGeometries</td>
<td>XtCComputeGeometries</td>
<td>Function</td>
<td>Null Function</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNcurrentPage</td>
<td>XtCCurrentPage</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNforceHorizontalSB</td>
<td>XtCForceHorizontalSB</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNforceVerticalSB</td>
<td>XtCForceVerticalSB</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhAutoScroll</td>
<td>XtCAutoScroll</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhInitialDelay</td>
<td>XtCInitialDelay</td>
<td>int</td>
<td>500</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhMenuPane</td>
<td>XtCMenuPane</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNhRepeatRate</td>
<td>XtCRepeatRate</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhScrollbar</td>
<td>XtCScrollbar</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNhSliderMoved</td>
<td>XtCSliderMoved</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNhStepSize</td>
<td>XtCStepSize</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinitialX</td>
<td>XtCInitialX</td>
<td>Position</td>
<td>0</td>
<td>GI</td>
</tr>
<tr>
<td>XtNinitialY</td>
<td>XtCInitialY</td>
<td>Position</td>
<td>0</td>
<td>GI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrecomputeHeight</td>
<td>XtCRecomputeHeight</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrecomputeWidth</td>
<td>XtCRecomputeWidth</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNshowPage</td>
<td>XtCShowPage</td>
<td>Define</td>
<td>OL_NONE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNviewHeight</td>
<td>XtCViewHeight</td>
<td>Dimension</td>
<td>(n/a)</td>
<td>SGI</td>
</tr>
</tbody>
</table>
ScrolledWindow (3W)

**ScrolledWindow** Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNviewWidth</td>
<td>XtCViewWidth</td>
<td>Dimension</td>
<td>(n/a)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvAutoScroll</td>
<td>XtCAutoScroll</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvInitialDelay</td>
<td>XtCInitialDelay</td>
<td>int</td>
<td>500</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvMenuPane</td>
<td>XtCMenuPane</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNvRepeatRate</td>
<td>XtCRepeatRate</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNvScrollbar</td>
<td>XtCScrollbar</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNvSliderMoved</td>
<td>XtCSliderMoved</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNvStepSize</td>
<td>XtCStepSize</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNalignHorizontal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range of values:</td>
<td></td>
<td></td>
<td>OL_BOTTOM/&quot;bottom&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OL_TOP/&quot;top&quot;</td>
<td></td>
</tr>
</tbody>
</table>

This resource is used to specify whether the horizontal scrollbar should be placed at the top or bottom of the ScrolledWindow. The default placement is at the bottom.

**XtNalignVertical**

Range of values:

OL_RIGHT/"right"

OL_LEFT/"left"

This resource is used to specify whether the vertical scrollbar should be placed at the left or right of the ScrolledWindow. The default placement is at the right.

**XtNcomputeGeometries**

This resource is used to allow intelligent cooperation during the layout stage between the ScrolledWindow and its content widget. The content widget sets this resource to a pointer to a function which is to be called whenever the ScrolledWindow needs to layout its children. The function is called as:

```c
typedef struct _OlSWGeometries {
    Widget sw;
    Widget vsb;
    Widget hsb;
    Dimension bb_border_width;
    Dimension vsb_width;
    Dimension vsb_min_height;
    Dimension hsb_height;
    Dimension hsb_min_width;
    Dimension sw_view_width;
    Dimension sw_view_height;
    Dimension bbc_width;
    Dimension bbc_height;
} _OlSWGeometries;
```
The ScrolledWindow widget populates the values in this structure prior to the call and examines them after the call to perform the layout operation.

The callback function is responsible for populating the bbc_width, bbc_height elements of this structure with the desired size of its window; the bbc_real_width, bbc_real_height elements with the logical size of the data, and force_hsb, force_vsb flags to indicate which scrollbars the ScrolledWindow should include in the layout.

XtNcurrentPage
The value of this resource is passed through to the vertical scrollbar of the ScrolledWindow. See a SCROLLBAR WIDGET(3W) for more details.

XtNshowPage
These resources are directed to the vertical scrollbar in the ScrolledWindow widget. See ScrollBar(3W) for more detail.

XtNforceHorizontalSB
XtNforceVerticalSB
Range of Values:

TRUE
FALSE

When the child widget is created and positioned within the ScrolledWindow, its width and height are examined. If the entire child widget will fit within the width (length) of the ScrolledWindow, the horizontal (vertical) scrollbar will not be created, since there is no need to scroll in that direction. Setting these resources to TRUE disables this checking and will force a horizontal (vertical) scrollbar to be attached to the window regardless of the dimension of the child widget. If a scrollbar is forced but not needed because the Content fits within the View, the scrollbar is made insensitive.

XtNhAutoScroll
XtNvAutoScroll
This resource is used to set the scrolling mode in the horizontal (vertical) direction. When set to TRUE, the ScrolledWindow widget is responsible for all interaction with the scrollbar and the positioning of the content window within the view. When set to FALSE, the application is responsible for all scrollbar interaction and scrolling of the data within the content window.
ScrolledWindow(3W)

XtNhInitialDelay
XtNvInitialDelay
These resources are used to specify the time in milliseconds of the initial repeat
delay to be used when the scrolling arrows of the horizontal (vertical) scrollbar
cOMPONENT OF THE ScrolledWindow are pressed.

XtNhMenuPane
XtNvMenuPane
These resources mimic the XtNmenuPane resources for the horizontal and vertical
scrollbars, respectively. See ScrollBar(3W) for more details.

XtNhRepeatRate
XtNvRepeatRate
These resources are used to specify the time in milliseconds of the repeat delay to
be used when the scrolling arrows of the horizontal (vertical) scrollbar component
OF THE ScrolledWindow are pressed.

XtNhScrollbar
XtNvScrollbar
These resources provide the widget id's of the horizontal and vertical scrollbars.
An application can use these values to set scrollbars' characteristics, such as
coloration.

XtNhSliderMoved
XtNvSliderMoved
An application may track the position of the child within the ScrolledWindow by
linking into these callbacks. They mimic the XtNsliderMoved resources of the
horizontal and vertical scrollbars, respectively.

The call_data parameter for these callbacks is a pointer to an OlScrollBar-
Verify structure, as in the Scrollbar widget. The application can validate a
scroll attempt before the ScrolledWindow widget will reposition the View of the
Content, and can update the page number and adjust the scrollbar elevator posi-
tion. See ScrollBar(3W) for more details.

XtNhStepSize
XtNvStepSize
Range of Values:

\[
0 < \text{XtNhStepSize} < \text{XtNStepSize}
\]

These resources are related to the XtNgranularity resource for the horizontal
and vertical scrollbars, respectively, but have an important distinction: their
values are the size in pixels of the minimum scrollable unit in the Content. For
instance, to allow the end user to scroll a single pixel in either direction, these
values would be 1. Or, to allow the end user to scroll a character at a time hori-
zontally and a line at a time vertically, these values would be the width of a
character and the height of a line, respectively. (Scrolling a character at a time
requires a constant width font, of course.) The ScrolledWindow widget uses
these values to calibrate the minimum scrolling step, \texttt{XtNgranularity}, of the scrollbars.

\textbf{XtNinitialX}  
\textbf{XtNinitialY}  
Range of Values:  
\begin{itemize}
  \item \texttt{XtNinitialX} \leq 0
  \item \texttt{XtNinitialY} \leq 0
\end{itemize}

The child widget is initially positioned at the upper left corner \((x,y)\) coordinates \((0,0)\). This positioning can be changed by specifying a new \(x,y\) location. The scrollbars are adjusted to give a visual indication of the offset specified in these resources.

Note that the Content is positioned within the View of the Content, so as the View of the Content moves progressively further through the Content, the coordinates of the position become more \textit{negative}. Thus the initial coordinates given in these resources should be zero or negative to assure proper operation of the scrolled window.

\textbf{XtNrecomputeHeight}  
\textbf{XtNrecomputeWidth}  
Range of Values:  
\begin{itemize}
  \item TRUE
  \item FALSE
\end{itemize}

These resources control how the \texttt{ScrolledWindow} widget should respond to requests to resize itself. Where one of these resources is \texttt{TRUE}, the \texttt{ScrolledWindow} shrinks the View of the Content in the corresponding direction to absorb the change in the \texttt{ScrolledWindow} widget's size. Where one of these resources is \texttt{FALSE}, the \texttt{ScrolledWindow} does not shrink the View in that direction.

These resources, together with the \texttt{XtNviewWidth} and \texttt{XtNviewHeight} resources, are typically used to set a preferred dimension in a direction that should not be scrolled.

\textbf{XtNviewHeight}  
\textbf{XtNviewWidth}  
Range of Values:  
\begin{itemize}
  \item \(0 \leq \texttt{XtNviewHeight}\)
  \item \(0 \leq \texttt{XtNviewWidth}\)
\end{itemize}

These resources define the preferred size of the View of the Content in pixels. For each, if a nonzero value is given, the corresponding \texttt{XtNheight} or \texttt{XtNwidth} resource is computed by adding the thickness of any scrollbar that appears. Any value in the \texttt{XtNheight} or \texttt{XtNwidth} resource is overwritten. If a zero value is given in the \texttt{XtNviewHeight} or \texttt{XtNviewWidth} resource, the corresponding \texttt{XtNheight} or \texttt{XtNwidth} resource is used instead.
Regardless of which resources identify the preferred height or width, the height or width of the View is never smaller than any scrollbar next to it. These resources also represent the maximum size of the View. While the `ScrolledWindow` may resize the View smaller than indicated in these resources (cf. `XtNrecomputeHeight` and `XtNrecomputeWidth`), it will never resize the View larger than indicated.
NAME
ScrollingList – a pane containing a scrollable list of text items

SYNOPSIS
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <ScrollingL.h>

static Widget scrollinglist, textfield;
Args args[l];
scrollinglist = XtCreateWidget(name, scrollingListWidgetClass, ...);
XtSetArg(args[0], XtNtextField, &textfield);
XtGetValues(scrollinglist, args, 1);

DESCRIPTION
ScrollingList Components
Each ScrollingList widget has the following parts:
— Border
— Current Item
— Current Item Border
— Items
— Scrollbar
— View
If the application allows the list to be edited in place (in the View), the `ScrollingList` widget uses the following components:

- Editable Text Field
Figure 2. Editable Scrolling List Additional Components

Editable Scrolling List
The application can choose whether to allow the end user to add, change, and/or delete the items in a scrolling list. The Editable Text Field is the interface for entering, or changing the Item, and is described later. Other aspects of the user interface for editing are controlled by the application. For example, the application can attach a menu to the scrolling list to allow the end user to select where a new Item is to be inserted, and can employ pop-up windows to gather additional information about a new Item.

Editing Directly in the List—the Editable Text Field
The application can request that the ScrollingList widget manage part of the visual aspect of changing an existing Item in the View. The ScrollingList widget automatically creates a widget of class TextField that implements the Editable Text Field. The ScrollingList widget manages the Editable Text Field widget as follows:

- The application asks the ScrollingList widget to "open" and "close" the Editable Text Field. Opening the Editable Text Field widget maps it and positions it so that, as the end user types in the name of a new or changed Item, the name lines up with the existing Item names. Closing the Editable Text Field widget unmaps it. (As described below, there may be times when the widget is unmapped yet still open.) If an existing Item is being edited, the application requests the Editable Text Field to overlay the Item. If a new Item is being inserted, the application requests Items to be scrolled down in the View to accommodate the Editable Text Field.
• The ScrollList widget maps and unmaps the Editable Text Field widget; the application does not.

• If the end user scrolls the list while the Editable Text Field is still open, the ScrollList widget scrolls it with the rest of the Items. If it has to be scrolled out of the View, it is scrolled out entirely, causing it to be unmapped but not closed. The application should not try to remap the child since it will be remapped when the list is scrolled back again.

• If the end user attempts to make a selection or set a Current Item, the Editable Text Field is automatically closed.

The application is responsible for handling the verification callbacks of the Editable Text Field and for telling the ScrollList widget to add a new Item or change an existing Item as a result of the user input.

Selectable Scroll List
The application can choose whether to allow the end user to select Items from a scrolling list. If Items can be selected, they can be copied elsewhere as text, and may be deletable ("cut"); see below for details.

Deleting Selected Items
The end user can delete selected Items. The ScrollList widget provides some deletion capabilities through the selection mechanisms (see the discussion under "Selecting and Operating on the Items" below), and the application can provide other capabilities, such as with a pop-up menu choice. The application verifies that each selected Item can be deleted; it is responsible for providing feedback to the end user for any Items it will not delete. The ScrollList widget updates the View to remove any deleted Items.

Virtual List
The ScrollList widget "virtualizes" the list to allow the application to use list data structures best suited to its needs. The ScrollList widget provides routines the application uses to build and maintain a version of the list for the ScrollList widget to use. With these routines, the application:

— adds new Items to the list;
— deletes Items from the list;
— edit items and mark them as changed;
— shifts the View to show a particular Item;
— and opens and closes the Editable Text Field for a new or changed Item.

The application is responsible for defining callbacks that the ScrollList widgets invoke when the end user attempts to change a Current Item, or cuts Items from the list. Each Item is identified by the Item name that is shown in the View for the end user, a token assigned by the ScrollList widget that uniquely identifies the Item, and an attributes bit-vector that identifies if the Item is a Current Item.

Order of Items in the Virtual List
The list is assumed to have an order defined by the application. As it adds Items, the application tells the ScrollList widget where to insert them: either before or after an Item already in the list.
Changeable List
The application may change the content of a list at any time, including while it is displayed. The widget updates the View, if necessary, to reflect the changed list. To avoid unnecessary updates to the View when several changes need to be made, the application can tell the ScrollingList widget to avoid updates until the changes are finished.

Setting a Current Item
The end user can make one or more of the Items a Current Item, as determined by the application, by

- clicking or pressing SELECT over it,
- or moving the input focus inside the Border and typing the first letter of the Item’s name.

Either of these actions causes a callback to the application, which can decide if the Item should be made a Current Item, remain a Current Item, or be changed to a regular Item, depending on the current state of the Item and the needs of the application. Thus, the application can make the scrolling list behave as a set of exclusive or nonexclusive Items.

Clicking or pressing SELECT also starts a selection, as described below.

Selecting and Operating on the Items
The ScrollingList widget allows selection operations on the Items. Items that are moved or copied from the View are treated as a newline-separated list of text items, in the order they appear in the scrolling list, with no leading or trailing blanks on any Item.

selecting a single Item
Clicking SELECT on an Item selects it and deselects any other active selection on the screen.

selecting other Items
Clicking ADJUST on an Item toggles its state, making an unselected Item selected and a selected Item unselected.

two-through selection, with SELECT
Pressing and dragging SELECT over Items selects them and deselects any other active selection on the screen. The selection starts with the Item where SELECT is pressed and extends to the Item where SELECT is released. If the pointer moves above or below the View, the View scrolls additional Items into the View, selecting them as well. The rate at which Items scroll into the View is the same as when pressing SELECT on the up or down arrows of the Scrollbar. The pointer can move out of the View to the left or right without interrupting the selection.

two-through selection, with ADJUST
Pressing and dragging ADJUST marks the bounds of a selection the same way as pressing and dragging SELECT, except that the Items covered are "toggled". (Previously selected Items are deselected and previously unselected Items are selected.)
copying Items
Pressing COPY copies any selected Items to the clipboard and deselects them.

cutting Items
Pressing CUT moves any selected Items to the clipboard and deletes them from the list. This operation is allowed only if the scrolling list is editable.

Coloration
On a monochrome display, the ScrollingList widget indicates that it has input focus by inverting foreground and background colors. When an editable Text Field has input focus, it shows that it has input focus by showing an active caret.

On color displays, the ScrollingList widget shows that the Current Item has input focus by filling the background of the Current Item with the input focus color set in the XtNinputFocusColor resource. When a selected item has input focus, the label is drawn with the input focus color. When an item is both Selected and Current, it shows that it has input focus by drawing the text of the label in the input focus color. When an editable Text Field has input focus, it shows that it has input focus by showing an active caret in the input focus color.

EXCEPTIONS:
If the input focus color is the same as either the background, foreground, or font color, then revert to the monochrome coloration scheme.

Figures 3 and 4 illustrate the resources that affect the coloration of the ScrollingList widget.
Figure 4. Selected Item and Unselected Item Coloration

**Keyboard Traversal**

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

The ScrollingList widget responds to the following keyboard navigation keys:

- NEXT_FIELD moves to the next traversable widget in the window
- PREV_FIELD moves to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application
- MOVEUP moves the input focus up one line
- MOVEDOWN moves the input focus down one line
- PANESTART moves the input focus to the first item in the pane
- PANEEND moves the input focus to the last item in the pane
- SCROLLUP scrolls up one item in the list
- SCROLLDOWN scrolls down one item in the list
ScrollingList (3W)

SCROLLTOP  scrolls to the first item in the list
SCROLLBOTTOM  scrolls to the last item in the list
PAGEUP  scrolls up one page so that the first item visible is the last item visible in the pane
PAGEDOWN  scrolls down one page so that the last item visible is the first item visible in the pane

When an Editable Text Field is in the ScrollingList, the keyboard traversal keys defined for TextField widgets apply.

The SELECTKEY selects the Current Item and unselects any other active selection on the screen. The ADJUSTKEY toggles the Current Item’s state, making an unselected Item selected and a selected Item unselected.

Note that the scrolling keys of interest are defined with in the ScrollingList and traversal to the Scrollbar is not necessary to manipulate the ScrollingList.

<table>
<thead>
<tr>
<th>Scrolling List Activation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Type</td>
</tr>
<tr>
<td>OL MENUKEY</td>
</tr>
</tbody>
</table>

**Display of Keyboard Mnemonic**

The ScrollingList widget displays the mnemonic accelerator for each item as part of its label. If the mnemonic character is in the label, then that character is displayed/highlighted according to the value of the application resource XtNshowMneumonics. If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the application resource XtNshowMneumonics.

If truncation is necessary, the mnemonic displayed in parenthesis is truncated as a unit.
ScrollingList (3W)

SUBSTRUCTURE
Scrollbar component
Name: scrollbar
Class: Scrollbar

Editable Text Field component
Name: textfield
Class: TextField

<table>
<thead>
<tr>
<th>Application Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>XtNfont</td>
</tr>
<tr>
<td>XtNfontColor</td>
</tr>
<tr>
<td>XtNforeground</td>
</tr>
<tr>
<td>XtNmaximumSize</td>
</tr>
<tr>
<td>XtNstring</td>
</tr>
<tr>
<td>XtNverification</td>
</tr>
</tbody>
</table>

‡ The defaults are set to agree with the values of these resources for the ScrollingList widget itself.

RESOURCES

<table>
<thead>
<tr>
<th>ScrollingList Resource Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
</tr>
<tr>
<td>XtNapplAddItem</td>
</tr>
<tr>
<td>XtNapplDeleteItem</td>
</tr>
<tr>
<td>XtNapplEditClose</td>
</tr>
<tr>
<td>XtNapplEditOpen</td>
</tr>
<tr>
<td>XtNapplITouchItem</td>
</tr>
<tr>
<td>XtNapplUpdateView</td>
</tr>
<tr>
<td>XtNapplViewItem</td>
</tr>
<tr>
<td>XtNbackground</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
</tr>
<tr>
<td>XtNborderColor</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
</tr>
<tr>
<td>XtNfont</td>
</tr>
<tr>
<td>XtNfontColor</td>
</tr>
<tr>
<td>XtNfontGroup</td>
</tr>
<tr>
<td>XtNforeground</td>
</tr>
<tr>
<td>XtNheight</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
</tr>
<tr>
<td>XtNrecomputeWidth</td>
</tr>
</tbody>
</table>
ScrollingList Resource Set (cont'd)

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNselectable</td>
<td>XtCSelectable</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNtextField</td>
<td>XtCTextField</td>
<td>Widget</td>
<td>(none)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserDeleteItems</td>
<td>XtCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNuserMakeCurrent</td>
<td>XtCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNviewHeight</td>
<td>XtCViewHeight</td>
<td>Cardinal</td>
<td>(none)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

OListltem Structure

Several of the resources defined below use the following `OListltem` structure:

```c
typedef struct _OListltem {
    OListltem
    OlDefine label_type;
    XtPointer label;
    XImage *glyph;
    OlBitMask attr;
    XtPointer user_data;
    unsigned char mnemonic;
} OListltem;
```

`label_type`

identifies the type of label to display for the Item in the View. It can have one of the values

- `OL_STRING` for a text label;
- `OL_IMAGE` for an image label.

Note:

Only text labels are supported in this version of the `ScrollingList` widget, so the only value allowed is `OL_STRING`. Any other legal values generate an error message that tells the application programmer that the value is not yet supported. Any illegal values generate a different error message.

`label` is what to display for the Item in the View. The type of the value of this member depends on the value of the `label_type` member:

- `OL_STRING` String
glyph is currently unused.

attr defines attributes of the Item. It is a bit vector with the bit references:

- **OL_LIST_ATTR_APPL**: for application use. This is a mask of 16 contiguous bits that can be subdivided as the application sees fit. These are the low 16 bits of the value, so no shifting is necessary to access the bits as an integer value.

- **OL_LIST_ATTR_CURRENT**: if the Item is a Current Item.

Other bit values are undefined but should not be used by the application.

**mnemonic**

is a single character that is used as a mnemonic accelerator for keyboard traversal.

**OlListToken Structure**

The ScrollingList widget identifies each Item with a "token" of type OlListToken. The ScrollingList widget assigns the token when an Item is added by the application, and the application uses the token in later references to the Item. A zero value is allowed in some contexts where an OlListToken is expected, as a way to refer to no Item.

As a convenience to the application, the function OlListItemPointer(token) converts an OlListToken value into a pointer to the corresponding OlListItem. The application can change the values of the OlListItem members, but should let the ScrollingList widget know that they have changed, using the XtNapplTouchItem routine. No checking is done for incorrect OlListToken arguments to the OlListItemPointer() function.

The OlListToken value can be coerced into the type XtPointer and back without loss of precision.

**XtNapplAddItem**

This resource gives a pointer to a routine the application can call when it adds a new Item to the list. This routine is also used to build the list from scratch.

**Synopsis:**

```c
OlListToken (*applAddItem)(), token;
static Arg query[] = {
    { XtNapplAddItem, (XtArgVal)&applAddItem }
};
XtGetValues(widget, query, XtNumber(query));
token = (*applAddItem)(widget, parent, reference, item)
Widget widget;
OlListToken parent, reference;
OlListItem item;
```
widget identifies the ScrollingList widget instance.

parent should be set to 0, for compatibility with future changes.

reference identifies an Item before which to insert the new Item. This value can be zero to append the new Item to the list.

item describes the new Item. The content of the OllListltem structure is copied by the ScrollingList widget into space that it maintains; however, the data pointed to by the label and glyph members are not copied. The application can access the copied data directly, using the OllListltemPointer() function to get a pointer to the OllListltem structure for the Item. If it changes the data, the application should use the XtNapplTouchltem routine to let the ScrollingList widget know the data has changed.

If mapped and if allowed by the application (see XtNapplUpdateView), the ScrollingList widget updates the View if the new Item will be in the View. The View is changed as little as possible: if the new Item is in the upper half of the View, the Items above it are scrolled up and the top Item is scrolled off; if the new Item is in the lower half of the View, the Items below it are scrolled down and the bottom Item is scrolled off.

XtNapplDeleteItem
This resource gives a pointer to a routine the application can call when it deletes an Item from the list.

Synopsis:

```c
void (*applDeleteItem)();
static Arg query[] = {
    { XtNapplDeleteItem, (XtArgVal)&applDeleteItem }
};
XtGetValues(widget, query, XtNumber(query));
(*applDeleteItem)(widget, token)
OllListToken token;
```

widget identifies the ScrollingList widget instance.

token identifies the deleted Item.

If mapped and if allowed by the application (see XtNapplUpdateView), the ScrollingList widget updates the View if the deleted Item was visible. The View is changed as little as possible: if the deleted Item was in the upper half of the View, Items above it are scrolled down and an Item is scrolled in from the top; if the deleted Item was in the lower half of the View, Items below it are scrolled up and an Item is scrolled in from the bottom. If the View is already at the top or bottom, the additional Item is scrolled in from the other end, if possible.
**XtNapplEditClose**

This resource gives a pointer to a routine the application can call when the user has finished editing an Item in the View.

Synopsis:

```c
void (*applEditClose)();
static Arg query[] = {
    {XtNapplEditClose, (XtArgVal)&applEditClose }
};
XtGetValues(widget, query, XtNumber(query));
(*applEditClose)(widget)
```

- `widget` identifies the `ScrollingList` widget instance.

When this routine is called, the `ScrollingList` widget unmaps the Editable Text Field widget, scrolling up the Items below it if they had been scrolled down to allow an insert. The application is responsible for calling the `XtNapplAddItem` routine to add the new Item, or calling the `XtNapplTouchItem` routine to mark the Item as changed. To avoid unnecessary updates to the View, the application should add the new Item (`XtNapplAddItem`) or mark the changed Item (`XtNapplTouchItem`) before closing the Editable Text Field.

A later call to the `XtNapplEditClose` routine without an intervening call to the `XtNapplEditOpen` routine is ignored.

If mapped, the `ScrollingList` widget updates the View, even if the application had halted updates (see `XtNapplUpdateView`). If the application had halted updates, they will continue to be halted afterwards.

**XtNapplEditOpen**

This resource gives a pointer to a routine the application can call when it wants to allow the end user to insert a new Item or change an existing Item in the View.

Synopsis:

```c
void (*applEditOpen)();
static Arg query[] = {
    {XtNapplEditOpen, (XtArgVal)&applEditOpen }
};
XtGetValues(widget, query, XtNumber(query));
(*applEditOpen)(widget, insert, reference)
```

- `widget` identifies the `ScrollingList` widget instance.

- `insert` tells whether Items should be scrolled down to make room for inserting a new Item. A value of FALSE implies that an Item is being edited in place and no Items are to be scrolled.
reference
identifies an Item before which a new Item is to be inserted (insert is TRUE) or identifies the Item that is being changed (insert is FALSE). If insert is TRUE, this value can be zero to append a new Item at the end of the list. If insert is FALSE, this value must refer to an existing Item. The referenced Item does not have to be in the View—see below.

If a new Item is being inserted, the ScrollingList widget makes room for the Editable Text Field by scrolling down the referenced Item and any Items below it. If the referenced Item is not in the View, it is automatically made visible just as if the application had called the XtNapplViewItem routine first.

The XtNapplEditOpen routine can be called again before an intervening call to the XtNapplEditClose routine. The effect is as if the XtNapplEditClose routine was called, but without multiple updates to the View. For example, this allows the application to let the end user insert several new Items in succession: the Editable Text Field moves down as each Item is inserted, but is never removed from the View.

If mapped, the ScrollingList widget updates the View, even if the application had halted updates (see XtNapplUpdateView). If the application had halted updates, they will continue to be halted afterwards.

**XtNapplTouchItem**
This resource gives a pointer to a routine the application can call when it changes an Item in the list.

Synopsis:

```c
void (*applTouchItem)();
static Arg query[] = {
    { XtNapplTouchItem, (XtArgVal)&applTouchItem } }
);
XtGetValues(widget, query, XtNumber(query));
(*applTouchItem)(widget, token)

Widget widget;
O1ListToken token;

widget identifies the ScrollingList widget instance.
token identifies the Item that has changed.

If mapped and if allowed by the application (see XtNapplUpdateView), the ScrollingList widget updates the View if the changed Item is visible.

**XtNapplUpdateView**
This resource gives a pointer to a routine the application can call to keep the ScrollingList widget from updating the View, or to let it update the View again.

Synopsis:

```c
void (*applUpdateView)();
static Arg query[] = {
```
ScrollingList (3W)

```c
{ XtNapplUpdateView, (XtArgVal)&applUpdateView }
};
XtGetValues(widget, query, XtNumber(query));

(*applUpdateView) (widget, ok)
Widget widget;
Boolean ok;

ok is either TRUE or FALSE, depending on whether the ScrollingList can update the View as it changes, or not, respectively.

From the time the `XtNapplUpdateView` routine is called with a FALSE argument until it is called with a TRUE argument, the `ScrollingList` does not update the View in response to application-made changes, except:
- if the application opens or closes the Editable Text Field (cf `XtNapplEditOpen` and `XtNapplEditClose`);
- if the end user manipulates the list by scrolling it, selecting an Item, cutting, etc.

The `ScrollingList` widget updates the View once for each of these exceptions, each time an exception occurs.

An application should use this routine to bracket a set of changes to avoid spurious changes to the View. This routine is not needed if only one change is made to the list. The following example illustrates the use of the `XtNapplUpdateView` routine.

```c
/*
  * Stop View updates.
  */
(*applUpdateView) (widget, FALSE);

/*
  * Make some changes.
  */
(*applDeleteItem) (widget, ...);
(*applDeleteItem) (widget, ...);
(*applDeleteItem) (widget, ...);
(*applAddItem) (widget, ...);
(*applTouchItem) (widget, ...);

/*
  * Allow the View to be updated again.
  */
(*applUpdateView) (widget, TRUE);
```

`XtNapplViewItem`

This resource gives a pointer to a routine the application can call when it wants a particular Item placed in the View.

Synopsis:
```c
void (*applViewItem)();
static Arg query[] = {
```
{ XtNapplViewItem, (XtArgVal)&applViewItem }
};
XtGetValues(widget, query, XtNumber(query));
(*applViewItem)(widget, token)
Widget widget;
OListToken token;

widget identifies the ScrollingList widget instance.
token identifies the Item to move into the View.

The Item is moved into the View in a way that minimizes the change to the View. If the Item is currently in the View, nothing is changed. If scrolling the list up or down brings the Item into the View while keeping at least one previously viewed Item in the View, the list is scrolled. Otherwise, the Item is placed at the top of the View, or as close to the top as possible if there aren’t enough Items in the current Level to fill the View below it.

If mapped and if allowed by the application (see XtNapplUpdateView), the ScrollingList widget updates the View.

XtNrecomputeWidth
Range of Values:
TRUE
FALSE

This resource controls how the ScrollingList widget should respond to requests to resize itself. If this resource is TRUE, the ScrollingList shrinks the View of the Content in the corresponding direction to absorb the change in the ScrollingList widget’s size. If this resource is FALSE, the ScrollingList does not shrink the View in that direction.

This resource, together with the XtNviewHeight resource, are typically used to set a preferred dimension in a direction that should not be scrolled.

XtNselectable
Range of Values:
TRUE
FALSE

This resource controls whether the end user can select Items in the scrolling list. If set to TRUE, then Items can be selected with SELECT and ADJUST and copied with the COPY key. Items may be deleted with the CUT key, although the application can stop some or all selected Items from being deleted. If set to FALSE, then Items cannot be selected and the COPY and CUT keys have no effect.

XtNtextField
This is the widget ID of the Editable Text Field widget; its value is available once the ScrollingList widget has been created.

The ScrollingList widget resets the following values before returning from each invocation of the XtNapplEditOpen routine:
XtNtraversalOn
This resource specifies whether this widget is selectable during traversal.

XtNuserDeleteItems
This resource defines the callbacks issued when the end user tries to delete Items from the list. (Currently, the only way the ScrollingList widget handles deletions is through a cut operation.)

The call_data parameter points to a structure OlListDelete that looks like this:

```c
typedef struct _OlListDelete {
    OlListToken *tokens
    Cardinal num_tokens;
} OlListDelete;
```

tokens is a list identifying the Items to be deleted. The application is expected to act on each Item separately, calling the XtNapplDeleteItem routine to delete each from the list. The application may refuse to delete some or all of the Items, and is responsible for providing any feedback to the end-user.

num_tokens
is the number of Items to delete.

XtNuserMakeCurrent
This resource defines the callbacks issued when the end user presses SELECT over an Item.

The call_data parameter is the OlListToken value that identifies the Item. The application is expected to decide if the Current Item status of this Item should change. The attr member of the OlListItem structure for this Item is not automatically changed by the ScrollingList widget.

XtNviewHeight
Range of Values: $0 \leq \text{XtNviewHeight}$

This resource gives the preferred height of the View as the number of Items to show. If a nonzero value is given, the corresponding XtNheight resource is computed by converting this number to pixels and adding any padding or border thickness. In this case, any value given in the XtNheight resource is overwritten.

If a zero value is given in the XtNviewHeight resource, the XtNheight resource is used as an estimate. The View is sized to show an integral number of Items, such that the overall height of the ScrollingList widget is less than or equal to XtNheight, if possible. However, the View is always large enough to show at least one Item, and is no shorter than the minimum scrollbar size.
If neither the `XtNviewHeight` resource nor the `XtNheight` resource is set, or both are set to zero, the View is made as small as possible, limited as described above.
NAME

Slider – sets a numeric value and gives a visual indication of the setting range

SYNOPSIS

```
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Slider.h>

widget = XtCreateWidget(name, sliderWidgetClass, ...)
```

DESCRIPTION

Slider Components

The Slider widget implements a simple control used to change a numeric value. It consists of the following elements:

- Top (Left) Anchor (optional)
- Bottom (Right) Anchor (optional)
- Drag Box
- Bar (typically)
- Shaded Bar (typically)
- Current Value (not visible)
- Minimum Value (not visible)
- Minimum Value Label (optional)
- Maximum Value (not visible)
- Maximum Value Label (optional)
- Tick Marks (optional)

The Current Value is the numeric value a user attempts to change with the Slider widget.

![Slider Widget Diagram](image)

Figure 1. Horizontal Slider
Drag Box Motion
As visual feedback to the user, the Drag Box moves up or down (or left or right) along the Bar as the Current Value changes.

Dragging SELECT
The user can change the Current Value by dragging the Drag Box with SELECT. The pressing of SELECT must start with the pointer in the Drag Box, but the Drag Box (and the Current Value) track the pointer motion regardless of where it goes while SELECT is pressed. This means it is not possible for the user to change the Current Value by first pressing SELECT outside the Drag Box and then moving the pointer into it. Only the component of the pointer motion parallel to the Bar is tracked, and the motion of the Drag Box (and change in the Current Value) are limited by the length of the Bar.

Clicking SELECT
Clicking SELECT above the Drag Box for a vertical slider, or to the right for a horizontal slider, increases the Current Value by an application-specified amount, moves the Drag Box to correspond to the new Current Value, and moves the pointer to keep it on the Drag Box. Clicking SELECT to the other side of the Drag Box decreases the value by the same amount and moves the Drag Box and pointer accordingly. Pressing SELECT repeats this action.

Moving Drag Box to Limits
Clicking SELECT on one of the Bottom/Left or Top/Right Anchors causes the Current Value to take on the Minimum Value or Maximum Value, respectively,
and moves the Drag Box to the limit in the direction of the Anchor. If the Drag Box is already at the limit, nothing happens.

Clicking SELECT on an Anchor highlights the Anchor while the Current Value is changed.

**Application Notification**

The application finds out about a change in the Current Value on the release of SELECT for either the drag or click. It is responsible for providing any feedback to the end user deemed appropriate, such as updating the Current Value in a text field.

**Coloration**

When the Slider widget receives the input focus through keyboard traversal, the background color of the widget changes to the input focus color, found in the resource `XtNinputFocusColor`. If the user traverses out of the Slider widget, the background of the widget shall revert to its original background color.

**EXCEPTION:**

If the input focus color is the same as either the foreground or background color, then the widget shows input focus by switching the background and foreground colors.

Figure 3 illustrates the resources that affect the coloration of the **Slider** widget.

![Figure 3. Slider Coloration](image)
Keyboard Traversal

The Slider's default values of the XtNtraversalOn resource is TRUE.

The user can operate the Slider by using the keyboard to move the Drag Box and access the Anchors. The following keys manipulate the Current Value:

- SCROLLUP and SCROLLRIGHT increase the Current Value by an application-specified amount, and moves the Drag Box to correspond to the new Current Value.
- SCROLLDOWN and SCROLLLEFT decrease the Current Value by an application-specified amount, and moves the Drag Box to correspond to the new Current Value.
- SCROLLTOP and SCROLLRIGHTEDGE cause the Current Value to take on the Maximum Value, and moves the Drag Box to a vertical slider's top anchor or a horizontal slider's right anchor. The anchor is briefly highlighted while the Current Value is changed and the Drag Box is moved.
- SCROLLBOTTOM and SCROLLLEFTEDGE cause the Current Value to take on the Minimum Value, and moves the Drag Box to a vertical slider's bottom anchor or a horizontal slider's left anchor. The anchor is briefly highlighted while the Current Value is changed and the Drag Box is moved.

### Vertical Slider Activation Types

<table>
<thead>
<tr>
<th>Activation Type</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL_SCROLLUP</td>
<td>Drag Box moves up one Unit</td>
</tr>
<tr>
<td>OL_SCROLLDOWN</td>
<td>Drag Box moves down one Unit</td>
</tr>
<tr>
<td>OL_SCROLLTOP</td>
<td>Drag Box moves to top anchor</td>
</tr>
<tr>
<td>OL_SCROLLBOTTOM</td>
<td>Drag Box moves to bottom anchor</td>
</tr>
</tbody>
</table>

### Horizontal Slider Activation Types

<table>
<thead>
<tr>
<th>Activation Type</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL_SCROLLRIGHT</td>
<td>Drag Box moves right one Unit</td>
</tr>
<tr>
<td>OL.ScrollLeft</td>
<td>Drag Box moves left one Unit</td>
</tr>
<tr>
<td>OL_SCROLLRIGHTEDGE</td>
<td>Drag Box moves to right anchor</td>
</tr>
<tr>
<td>OL_SCROLLLEFTEDGE</td>
<td>Drag Box moves to left anchor</td>
</tr>
</tbody>
</table>

The Slider widget responds to the following keyboard navigation keys:

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

Display of Keyboard Mnemonic

The Slider does not display the mnemonic accelerator. If the Slider is the child of a Caption widget, the Caption widget will display the mnemonic as part of the label.

Display of Keyboard Accelerators

The Slider does not respond to a keyboard accelerator because clicking the SELECT button on a Slider activates depending on the pointer position. So, the Slider does not display a keyboard accelerator.

RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdragCBType</td>
<td>XtCDragCBType</td>
<td>OlDefine</td>
<td>OL_CONTINUOUS</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNendBoxes</td>
<td>XtCEndBoxes</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>FontStruct *</td>
<td>(OPEN LOOK default)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black*</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgranularity</td>
<td>XtCGranularity</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinitialDelay</td>
<td>XtCInitialDelay</td>
<td>int</td>
<td>500</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNminLabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaxLabel</td>
<td>XtCLabel</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNorientation</td>
<td>XtCOrientation</td>
<td>OlDefine</td>
<td>OL_VERTICAL</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrepeatRate</td>
<td>XtCRepeatRate</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNsliderMax</td>
<td>XtCSliderMax</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsliderMin</td>
<td>XtCSliderMax</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsliderMoved</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsliderValue</td>
<td>XtCSliderValue</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNspar</td>
<td>XtCSpan</td>
<td>Dimension</td>
<td>OL_IGNORE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNstopPosition</td>
<td>XtCStopPosition</td>
<td>OlDefine</td>
<td>OL_ALL</td>
<td>SGI</td>
</tr>
</tbody>
</table>
## Slider Resource Set (cont’d)

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNticks</td>
<td>XtCTicks</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtickUnit</td>
<td>XtCTickUnit</td>
<td>OIDefine</td>
<td>OL_NONE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

### XtNendBoxes

This resource selects the display of the end boxes.

### XtNgranularity

Range of Values:

\[
1 \leq \text{XtNgranularity} \leq (\text{XtNsliderMax} - \text{XtNsliderMin})
\]

Clicking SELECT on the Bar or Shaded Bar attempts to change the Current Value by the amount given in this resource. Dragging the Drag Box with SELECT changes the Current Value by this amount before the `XtNsliderMoved` callbacks are issued.

### XtNinitialDelay

Range of Values:

\[
0 < \text{XtNinitialDelay}
\]

This resource gives the time, in milliseconds, before the first action occurs when SELECT is pressed on the Bar or Shaded Bar. Note that millisecond timing precision may not be possible for all implementations, so the value may be rounded up to the nearest available unit by the toolkit.

### XtNminLabel

This is the label to be placed next to the maximum value position. For a vertical slider, the label is placed to the right of the minimum value position. If there is not enough space for the entire label and `XtNrecomputeSize` is FALSE, the label will be truncated from the end. If there is not enough space for the entire label and `XtNrecomputeSize` is TRUE, then the widget will request for more space to show the entire label.

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

### XtNmaxLabel

This is the label to be placed next to the maximum value position. For a vertical slider, the label is placed to the right of the minimum value position. If there is not enough space for the entire label and `XtNrecomputeSize` is FALSE, the label will be truncated from the end. If there is not enough space for the entire label
and \texttt{XtNrecomputeSize} is \texttt{TRUE}, then the widget will request for more space to show the entire label.

For an horizontal slider, the label is placed centered and below the maximum value position. If there is not enough room to center the label and \texttt{XtNrecomputeSize} is set to \texttt{FALSE}, the end of the label will be aligned with the left anchor. If this label collides with the min label, some part of the labels will overlap. If there is not enough room to center the label and \texttt{XtNrecomputeSize} is set to \texttt{TRUE}, the widget will request for more space to center the label below the maximum value position.

\texttt{XtNorientation}

Range of Values:

- \texttt{OL_HORIZONTAL/"horizontal"}
- \texttt{OL_VERTICAL/"vertical"}

This resource defines the direction for the visual presentation of the widget.

\texttt{XtNrepeatRate}

Range of Values:

- \texttt{0 < XtNrepeatRate}

This resource gives the time, in milliseconds, between repeated actions when \texttt{SELECT} is pressed on the Bar or Shaded Bar. Note that millisecond timing precision may not be possible for all implementations, so the value may be rounded up to the nearest available unit by the toolkit.

\texttt{XtNsliderMax} \\
\texttt{XtNsliderMin}

Range of Values:

- \texttt{XtNsliderMin < XtNsliderMax}

These two resources give the range of values tracked by the \texttt{Slider} widget. Mathematically, the range is open on the right; that is, the range is the following subset of the set of integers:

- \texttt{XtNsliderMin \leq range \leq XtNsliderMax}

This is independent of the Drag Box displayed in the \texttt{Slider} widget. The \texttt{Slider} widget is responsible for taking into account the size of the Drag Box when relating the physical range of movement to the range of values.
**XtNsliderMoved**

This resource defines the callback list used when the `Slider` widget is manipulated. The `call_data` parameter is a pointer to the Current Value; an `XtGet-Value()` inside the callback will return the previous value.

**XtNsliderValue**

Range of Values:

\[ \text{XtNsliderMin} \leq \text{XtNsliderValue} \leq \text{XtNsliderMax} \]

This resource gives the current position of the Drag Box, in the range \([\text{XtNsliderMin}, \text{XtNsliderMax}]\). The `Slider` widget keeps this resource up to date.

**XtNticks**

This is the interval between tick marks. The unit of the interval value is determined by `XtNtickUnit`.

**XtNtickUnit**

Range of values:

- OL_NONE/"none"
- OL_SLIDERVALUE/"slidervalue"
- OL_PERCENT/"percent"

This resource can have one of the values: OL_NONE, OL_SLIDERVALUE, and OL_PERCENT. If it is OL_NONE, then no tick marks will be displayed and `XtNticks` is ignored. If it is OL_PERCENT, then `XtNticks` is interpreted as the
percent of the slider value range. If it is OL_SLIDERVALUE, the XtNticks is interpreted as the same unit as slider value.

**XtNdragCBType**

Range of values:

- OL_CONTINUOUS/"continuous"
- OL_GRANULARITY/"granularity"
- OL_RELEASE/"release"

This resource determines the frequency of issuing XtNsliderMoved callbacks during a drag operation. If set to OL_CONTINUOUS, callbacks will be issued continuously. If set to OL_GRANULARITY, callbacks will only be issued when the drag box crosses any granularity positions. If set to OL_RELEASE, callback will only be issued once when the SELECT button is released.

**XtNstopPosition**

Range of values:

- OL_ALL/"all"
- OL_TICKMARK/"tickmark"
- OL_GRANULARITY/"granularity"

This resource determines the behavior of the drag box at the end of an drag operation. If set to OL_ALL, upon the release of the SELECT button in a drag operation, the drag box will be positioned at where it stops. If set to OL_TICKMARK, the drag box will snap to the nearest tickmark position. If set to OL_GRANULARITY, the drag box will snap to the nearest granularity position.

**XtNrecomputeSize**

This resource, if set to TRUE, allows the slider widget to resize itself whenever needed, to compensate for the space needed to show the tick marks and the labels. The slider widget uses the XtNspan, the sizes of the labels, and XtNtickUnit to determine the preferred size.

**XtNfont**

This resource specifies the font used to draw the labels. It defaults to the OPEN LOOK standard font.

**XtNfontcolor**

This resource specifies the color used to draw the labels. It defaults to the foreground color of the slider widget.

**XtNspan**

If XtNrecomputeSize is set to TRUE, then XtNspan should be set to reflect the preferred length of the slider, not counting the space needed for the labels. The slider widget uses the span value, the sizes of the labels, and XtNtickUnit to determine the preferred size.
NAME

StaticText – displays read-only text

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <StaticText.h>

widget = XtCreateWidget(name, staticTextWidgetClass, ...);
```

DESCRIPTION

The **StaticText** widget provides a way to present an uneditable block of text using a few simple layout controls.

**Word Wrapping**  
If the text is too long to fit in the width provided by the **StaticText** widget, the text may be "wrapped" if the application requests it. The wrapping occurs at a space between words, if possible, leaving as many words on a line as will fit. If a word is too long for the width, it is wrapped between characters. An embedded newline will always cause a wrap.

**Text Clipping—in Width**  
If the text is not wrapped, it will be truncated if it cannot fit in the width of the **StaticText** widget. The application can choose whether the truncation occurs on the left, right, or evenly on both sides of each line of the text.

**Text Clipping—in Height**  
If the text is too large to fit in the height provided by the **StaticText** widget, the text is clipped on the bottom. The clipping falls on a pixel boundary, not between lines, so that it is possible that only the upper part of the last line of text may be visible.

**Stripping of Spaces**  
The application can choose to have leading spaces, trailing spaces, or both leading and trailing space stripped from the text before display, or can choose to have no stripping done.

**Selecting and Operating on the Text**  
The **StaticText** widget allows text to be selected in several ways and then copied. See **Text Selection** earlier in this manual for the description of these operations.

**Coloration**  
Figure 1 illustrates the resources that affect the coloration of the **StaticText** widget.
The quick brown fox jumped over the lazy widget.

**Figure 1. Static Text Coloration**

**Keyboard Traversal**

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

The widget responds to the following keyboard navigation keys:

- `NEXT_FIELD` moves to the next traversable widget in the window
- `PREV_FIELD` moves to the previous traversable widget in the window
- `NEXTWINDOW` moves to the next window in the application
- `PREVWINDOW` moves to the previous window in the application
- `NEXTAPP` moves to the first window in the next application
- `PREVAPP` moves to the first window in the previous application

**Display of Keyboard Mnemonic And Accelerator**

The `StaticText` does not have keyboard mnemonic or keyboard accelerator capabilities.
## RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNalignment</td>
<td>XtCAlignment</td>
<td>OLDefine</td>
<td>OL_LEFT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XtFontStruct</td>
<td>(OPEN LOOK font)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black*</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XtCFontGroup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNgavity</td>
<td>XtCGravity</td>
<td>int</td>
<td>CenterGravity</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>OLDefine</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlineSpace</td>
<td>XtCLineSpace</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI*</td>
</tr>
<tr>
<td>XtNstatusWidget</td>
<td>XtCstatusWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>I</td>
</tr>
<tr>
<td>XtNstring</td>
<td>XtCString</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNstrip</td>
<td>XtCStrip</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwrap</td>
<td>XtCWrap</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

### XtNalignment

Range of Values:

- **OL_LEFT**/"left"
- **OL_CENTER**/"center"
- **OL_RIGHT**/"right"

This specifies the alignment to be applied when drawing the text, as described below:

- **OL_LEFT** causes the left sides of the lines to be vertically aligned
- **OL_CENTER** causes the centers of the lines to be vertically aligned
- **OL_RIGHT** causes the right sides of the lines to be vertically aligned
The application can set a width and height to the `StaticText` widget that exceeds the size needed to display the string. This resource controls the use of any extra space with the `StaticText` widget:

**CenterGravity**
- The string is centered vertically and horizontally in the extra space.

**NorthGravity**
- The top edge of the string is aligned with the top edge of the space and centered horizontally.

**SouthGravity**
- The bottom edge of the string is aligned with the bottom edge of the space and centered horizontally.

**EastGravity**
- The right edge of the string is aligned with the right edge of the space and centered vertically.

**WestGravity**
- The left edge of the string is aligned with the left edge of the space and centered vertically.

**NorthWestGravity**
- The top and left edges of the string are aligned with the top and left edges of the space.

**NorthEastGravity**
- The top and right edges of the string are aligned with the top and right edges of the space.

**SouthWestGravity**
- The bottom and left edges of the string are aligned with the bottom and left edges of the space.

**SouthEastGravity**
- The bottom and right edges of the string are aligned with the bottom and right edges of the space.
XtNlineSpace
Range of Values:
\[-100 \leq \text{XtNlineSpace}\]
This resource controls the amount of space between lines of text. It is specified as a percentage of the font height, and is the distance between the baseline of one text line and the top of the next font line. Thus, the distance between successive text baselines, in percentage of the font height, is
\[\text{XtNlineSpace} + 100\]

XtNrecomputeSize
Range of Values:
TRUE
FALSE
This resource indicates whether the StaticText widget should calculate its size and automatically set the XtNheight and XtNwidth resources. If set to TRUE, the StaticText widget will do normal size calculations that may cause its geometry to change. If set to FALSE, the StaticText widget will leave its size alone; this may cause truncation of the visible image being shown by the StaticText widget if the fixed size is too small, or may cause centering if the fixed size is too large.

XtNstatusWidget
This resource allows an internationalized application to specify an area in order to create a status window via Input Method.

XtNstring
This resource is the string that will be drawn. The string must be null terminated.

XtNstrip
Range of Values:
TRUE
FALSE
This resource controls the stripping of leading and trailing spaces during the layout of the text string.

<table>
<thead>
<tr>
<th>XtNstrip</th>
<th>XtNalignment</th>
<th>Spaces stripped</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>OL_LEFT</td>
<td>Leading spaces stripped.</td>
</tr>
<tr>
<td></td>
<td>OL_RIGHT</td>
<td>Trailing spaces stripped.</td>
</tr>
<tr>
<td></td>
<td>OL_CENTER</td>
<td>Both leading and trailing spaces stripped.</td>
</tr>
<tr>
<td>FALSE</td>
<td>(any)</td>
<td>None.</td>
</tr>
</tbody>
</table>

XtNwrap
Range of Values:
TRUE
FALSE
This resource controls the wrapping of lines that are too long to fit in the width of the StaticText widget.
<table>
<thead>
<tr>
<th>XtNwrap</th>
<th>XtNalignment</th>
<th>Wrap action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>OL_LEFT</td>
<td>Clipped on the right</td>
</tr>
<tr>
<td></td>
<td>OL_RIGHT</td>
<td>Clipped on the left</td>
</tr>
<tr>
<td></td>
<td>OL_CENTER</td>
<td>Clipped equally on both left and right</td>
</tr>
<tr>
<td>TRUE</td>
<td>(any)</td>
<td>Long text is broken at spaces between words so that each line of the displayed text has as many words as can fit.</td>
</tr>
</tbody>
</table>
NAME

**Stub** – used for widget prototyping

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Stub.h>

Widget parent, stub;

stub = XtCreateWidget(name, stubWidgetClass, ...
```

DESCRIPTION

The **Stub** widget is essentially a method-driven widget that allows the application to specify procedures at creation and/or `XtSetValues()` time which normally are restricted to a widget’s class part. Most class part procedures have been attached to the instance part. For example, with the stub widget, it’s possible to set the procedure that’s called whenever an exposure occurs. It’s also possible to set the `SetValues` and `Initialize` procedures.

**Build Unique Widgets within an application**

By allowing the application to specify procedures outside the widget class structure, applications can use the stub widget to build local widgets without having to go through the formal steps. For example, suppose an application wanted to create a menu separator widget that inherits its parent’s background color at creation time. It would be wasteful to create a new widget to perform these trivial tasks. Instead, the application would use a stub widget and specify an `Initialize` procedure for it.

**Graphics Applications**

The stub widget also implements graphics applications. Since the application has direct access to the widget’s internal expose procedure, the application can take advantage of the exposure compression provided with the `region` argument. This field is not accessible if the application used an Event Handler to trap exposures. Also, since the application has access to the `SetValues` and `SetValuesHook` procedures, the application can programmatically modify graphic-related resources of the stub widget.

**Inheriting Procedures from Existing Widgets**

Once a stub widget is created, other stub widgets can inherit its methods without the application having to specify them again. All the application has to do is specify a reference stub widget in the creation Arg list and the new stub widget will inherit all instance methods from the referenced stub widget.

**Wrapping Widgets around an existing Window**

The **Stub** widget also allows the application to give widget functionality to existing X windows. For example, if the application wanted to track button presses on the root window, the application would create a stub widget using the `RootWindow` id as the `XtNwindow` resource. Once this has been done, the application can monitor events on the root window by attaching event handlers to the stub widget.
Keyboard Traversal
The Stub is a Primitive widget and it inherits the translations for traversal actions from the Primitive class. The user of a Stub widget should add translations for dealing with the navigation events listed in the section of VIRTUAL KEYS/BUTTONS that apply to the particular use of the Stub.

Display of Keyboard Mnemonic
The Stub does not display the mnemonic accelerator. If the Stub is the child of a Caption widget, the Caption widget can be used to display the Stub's mnemonic.

Display of Keyboard Accelerators
The Stub does not display the keyboard accelerator. If the Stub is the child of a Caption widget, the Caption widget can be used to display the Stub's accelerator as part of the label.

Coloration
The Stub widget should display a state which indicates that it has input focus. The general heuristic used for this display in widgets is that the background color is replaced with the input focus color found in the resource XtNinputFocusColor.

RESOURCES
The following table lists the resources available to the stub widget.

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNacceptFocusFunc</td>
<td>XtCAcceptFocus</td>
<td>XtRFunction</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNaccelerator</td>
<td>XtCAccelerator</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNacceleratorText</td>
<td>XtCAcceleratorText</td>
<td>String</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNactivatefunc</td>
<td>XtCActivateFunc</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCBackgroundPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>Cardinal</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroy</td>
<td>XtCDestroy</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCDestroyCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>I</td>
</tr>
<tr>
<td>XtNexpose</td>
<td>XtCExpose</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNgetValuesHook</td>
<td>XtCGetValuesHook</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhilightHandlerProc</td>
<td>XtCHighlightHandler</td>
<td>XtRFunction</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinitialize</td>
<td>XtCInitialize</td>
<td>Function</td>
<td>(private)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNinitializeHook</td>
<td>XtCInitializeHook</td>
<td>Function</td>
<td>NULL</td>
<td>GI</td>
</tr>
</tbody>
</table>


**Table:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmnemonic</td>
<td>XtCMnemonic</td>
<td>unsigned char</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNqueryGeometry</td>
<td>XtCQueryGeometry</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrealize</td>
<td>XtCRealize</td>
<td>Function</td>
<td>(private)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceStub</td>
<td>XtCReferenceStub</td>
<td>Widget</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNregisterFocusFunc</td>
<td>XtCRegisterFocus</td>
<td>XtRFunction</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNresize</td>
<td>XtCresize</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI</td>
</tr>
<tr>
<td>XtNsetValues</td>
<td>XtCSetValues</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsetValuesAlmost</td>
<td>XtCSetValuesAlmost</td>
<td>Function</td>
<td>(superclass)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsetValuesHook</td>
<td>XtCSetValuesHook</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalHandlerFunc</td>
<td>XtCTraversalHandlerFunc</td>
<td>Function</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwindow</td>
<td>XtCWindow</td>
<td>Window</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNacceptFocusFunc**

This procedure has the same semantics as the XtAcceptFocusFunc Core Widget Class procedure and it's called by the Stub Widget Class's accept focus class procedure. Applications that want to override the default accept focus procedure should use this function. When overriding the default accept focus procedure, the convenience routine `OlCanAcceptFocus()` can be used to check the widget's focus-taking eligibility. `OlSetInputFocus` should be used instead of `XSetInputFocus` when explicitly setting focus to a window. (See the section "Input Focus" for more on setting and accepting focus.)

**XtNactivateFunc**

```c
void activateProc(w, activation_type, data)
    Widget w;
    OlVirtualName activation_type;
    XtPointer data;
```

This procedure is called whenever `OlActivateWidget()` is called with the stub widget id for which this routine was assigned. The procedure has the following declaration:

```c
Boolean (*OlActivateFunc)(w, activation_type, data)
    Widget w;
    OlVirtualName activation_type;
    XtPointer data;
```
If the activation_type is valid, the routine should take the appropriate action and return TRUE; otherwise, the routine should return FALSE.

**XtNdestroy**

```c
void destroy(w)
    Widget w;
```

Specifies the procedure called when this stub instance is destroyed.

**XtNexpose**

```c
void expose(w, xevent, region)
    Widget w;
    XEvent * xevent;
    Region region;
```

Procedure called whenever the a stub instance receives an exposure event. Since the Stub Widget class has requested exposure compression, the region field is valid.

**XtNgetValuesHook**

```c
void getValuesHook(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal * num_args;
```

Procedure called whenever the application does an XtGetValues() call on a stub widget instance.

**XtNhighlightHandlerProc**

This procedure has the same semantics as the OlHighlightProc class procedure and it is called by the Stub Widget Class’s HighlightProc class procedure. Applications that have Stub widgets which accept focus should set this routine so that the Stub widget can display an appropriate visual whenever it gains or loses focus.

**XtNinitialize**

```c
void initialize(request, new)
    Widget request;
    Widget new;
```

Procedure called by XtCreateWidget() for a stub widget instance. The default initialize procedure knows how to deal with the XtNwindow resource (see the section on XtNwindow). If the application supplies its own initialize procedure, it’s the application’s responsibility to deal with the XtNwindow resource.

When the XtNwindow resource is non-NULL, the default initialize procedure fills in XtNx, XtNy, XtNwidth and XtNheight with the attributes specified by the XtNwindow id.

**XtNinitializeHook**

```c
void initializeHook(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal * num_args;
```
This procedure is called by XtCreateWidget() for a stub widget instance after the initialize procedure has been called. The application can access the creation arg list through this routine. The widget specified with the w argument is the new widget from the initialize procedure.

XtNqueryGeometry

XtGeometryResult queryGeometry(w, request, preferred_return)

XtNrealize

void realize(w, value_mask, attributes)

Procedure called whenever the application does an XtQueryGeometry() request on a stub widget instance.

Procedure called to realize a stub widget instance. The default realize procedure knows how to deal with the XtNwindow resource (see the section on XtNwindow). If the application supplies its own realize procedure, it’s the application’s responsibility to deal with the XtNwindow resource.

When XtNwindow is non-NULL, the realize procedure uses this window for the widget instance instead of creating a new window. The default realize procedure gives an error message if another widget in its process space is referencing the window already. Note, the default realize procedure does not reparent the specified window, if one is supplied.

XtNreferenceStub

This is a pointer to an existing Stub widget. If this pointer is non-NULL, the new Stub will inherit all instance methods from the referenced stub widget. An XtSetValues request on the new Stub widget should be used to change any inherited methods.

XtNresize

void resize(w)

Procedure called whenever a stub widget instance is resized.

XtNregisterFocusFunc

This is a Stub Widget resource that points to a function of type

void (*OIRegisterFocusFunc)(w)

Whenever a stub widget gains focus this procedure is called and the stub’s shell sets the "current focus widget" (See OIGetCurrentFocusWidget) to the value returned by it. If this function is NULL or returns NULL, the stub widget is set to the current focus widget. This is the typical case. If this procedure returns a widget id other than the stub widget’s, that id is used to update the current focus widget so that a subsequent call to OIGetCurrentFocusWidget would return it.
Note, returning a widget id other than the stub widget’s will not move the focus away from the stub widget.

**XtNsetValues**

```c
Boolean setValues(current, request, new)

Widget current;
Widget request;
Widget new;
```

Procedure called whenever the application does an `XtSetValues()` call on a stub widget instance.

**XtNsetValuesAlmost**

```c
void setValuesAlmost(w, new_widget, request, reply)

Widget w;
Widget new_widget;
XtWidgetGeometry * request;
XtWidgetGeometry * reply;
```

This procedure is called when the application attempts to set a stub widget’s geometry via an `XtSetValues()` call and the stub widget’s parent did not accept the requested geometry. The default `setValuesAlmost` procedure simply accepts the suggested compromise.

**XtNsetValuesHook**

```c
Boolean setValuesHook(w, args, num_args)

Widget w;
ArgList args;
Cardinal * num_args;
```

This procedure is called whenever the application does an `XtSetValues()` on a stub widget instance. Since this procedure is called after the `setValues` procedure, the widget specified by the `w` argument is the new widget from the `setValues` procedure.

**XtNtraversalHandlerFunc**

If an application wants the stub widget to process traversal commands whenever the stub widget has focus, this resource is used to supply the traversal routine. An example of a case when this is desirable is when a stub widget is used to implement a spreadsheet. In this case the stub widget would trap the OL_MOVERIGHT, OL_MOVELEFT, etc. commands to move focus between the cells in the spreadsheet. The traversal handling routine has the following declaration:

```c
Widget (*OLTraversalFunc)(w, start, direction, time)

Widget w; /* Stub widget id */
Widget start; /* Stub widget id */
OLVirtualName direction;
Time time;
```

If the traversal routine can process the traversal command, it returns the id of the widget which now has focus. (Note: the widget id returned can be the stub widget’s id. This is the case when the traversal command was processed, but focus did not leave the stub widget.) If the traversal routine cannot process the
given command, it should return NULL. (See the section "Input Focus" for a discussion on valid direction values and focus movement.)

**XtNwindow**

This resource specifies a window id that the Stub widget should associate with its instance data at realization time. The **XtNwindow** resource can be specified at initialization time only. If a window id is supplied, that stub widget instance will trap events on the given window. After the stub widget instance is realized, the function **XtWindow()** will return this window id.

If the stub widget is managed by its parent widget, the supplied window will be included in geometry calculations even though the stub widget (by default) does not reparent the supplied window to be a child of the parent widget’s window.

Explicit calls to **XtMoveWidget**, **XtResizeWidget**, **XtConfigureWidget**, or **XtSetValues** can be used to modify the window’s attributes.

Note: When the stub widget instance is destroyed, the window will be destroyed along with it.

If the Stub widget is managed, the window ID supplied with **XtNwindow** will be included in the geometry calculation causing undesirable reconfiguration. This is an anomaly that exists only with this resource.

**XtNwidth**

**XtNheight**

If **XtNwindow** has a NULL value, the application must insure that the dimensions of **XtNwidth** and **XtNheight** are non-NULL. The application can specify the width and height with an Arg list or specify an initialize procedure that sets them with non-NULL values. If either of these dimensions are NULL when the application attempts to realize the stub widget, an error will result.

**EXAMPLE**

The following example illustrates how an application can use the stub widget to perform some particular type of exposure handling. Since an initialize procedure was not specified and the **XtNwindow** resource was not used, the initial Arg list includes non-NULL values for the widget’s width and height.

```c
static void
Redisplay(w, xevent, region)
  Widget w;
  XEvent * xevent;
  Region region;
{
  /*
  * do something interesting here
  */
  } /* END OF Redisplay() */

main(...)
{
  Widget base;
  Widget stub;
  static Arg args[] = {
```
(3W)

\begin{verbatim}
( XtNexpose, (XtArgVal) Redisplay ),
( XtNwidth, (XtArgVal) 1 ),
( XtNheight, (XtArgVal) 1 )

base = OlInitialize(...);

stub = XtCreateManagedWidget("graphics pane", stubWidgetClass,
   base, args, XtNumber(args));

......

} /* END OF main() */

SEE ALSO
   Input Focus for a discussion on keyboard focus manipulation
\end{verbatim}
NAME
TextEdit – provides multiple line editing

SYNOPSIS
#include <stdio.h>
#include <buffutil.h>
#include <textbuff.h>
#include <Intrinsic.h>
#include <OpenLook.h>
#include <TextEdit.h>

    widget = XtCreateWidget(name, textEditWidgetClass, . . .);

DESCRIPTION
The TextEdit widget provides an n-line text editing facility that has both a customizable user interface and a programmatic interface. It can be used for single line string entry as well as full-window editing. It provides a consistent editing paradigm for textual data.

The TextEdit widget provides three text wrap modes: wrapoff, wrapany, and wrapwhitespace.

The TextEdit widget provides several distinct callback lists used to monitor the state of the textual data: insertion cursor movement, modification of the text, and post modification notification. Each of these callbacks provide information to the application regarding the intended action. The application can simply examine this information to maintain its current state or can disallow the action and perform any of the programmatic manipulations instead.

The TextEdit widget also provides distinct callback lists for user input: mouse button down and key press. The call data for these callbacks decodes the input for the application. The application can examine the input and either consume the action, and perform any of the programmatic manipulations, or allow the widget to act upon it.

The TextEdit widget also provides the application with a callback list used when the widget is redisplayed. With this callback the application can add callbacks which can be used to display information in the margins of the TextEdit such as line numbers or update marks.

Editing Capabilities
The TextEdit widget provides editing capabilities to move the insert point, select text, delete text, scroll the display, perform cut, copy, paste, and undo operations, and refresh the text display. All of these capabilities are bound to global resources stored in the X server. Many of these settings can be changed using the property windows available in the OPEN LOOK Workspace Manager. All of these settings dynamically change immediately after new settings are stored in the server. The following table lists all of the key bindings used by the TextEdit widget:
Hierarchical Text
Text is considered to be hierarchically composed of white space, spans, lines, paragraphs, within a document. Whitespace is defined as any non-empty sequence of the ASCII characters space, tab, linefeed, or carriage return (decimal values 32, 9, 10, 13; respectively); a span is any non-empty sequence of characters bounded by whitespace. A source line is any (possibly empty) sequence of characters bounded by newline characters; a display line is any (possibly empty) sequence of characters appearing on a single screen display line. A paragraph is any sequence of characters bounded by sets of two or more adjacent newline characters. A document is the entire content of the text.

In all cases, the beginning or end of the edit text is an acceptable bounding element in the previous definitions.

Sizing the Display
When making display decisions, the TextEdit widget first will use either the application specified width and height or, if these values are not specified, calculate width and height by applying the values of the XtNcharsVisible and XtNlinesVisible resources. Once the width and height are determined the TextEdit widget will request an appropriate size from its parent (considering the margins). If the request is denied or only partially satisfied, no future growth requests will be made unless there is an intervening resize operation externally imposed.

Once the size of the display is settled, the TextEdit widget calculates the display lines based on this size, the various margins, the font, tab table, and wrap mode.

Wrapping
If the wrap mode is OL_WRAP_ANY, as many characters from the source line as will entirely fit before the right margin are written to the current display line, then the next character starts at the left margin of the next display line, and so on. If the wrap mode is OL_WRAP_WHITE_SPACE, the line wrap occurs at the first whitespace character that follows the last full word that does fit on the current display line. If, however, under OL_WRAP_WHITE_SPACE, the first full word that does not fit is the first word on the display line, the wrap is made as if OL_WRAP_ANY were selected. If the wrap mode is OL_WRAP_OFF the lines are not wrapped but are clipped at the right margin. In this mode the text is horizontally scrollable.

ScrolledWindow
The application can place the TextEdit widget within a ScrolledWindow widget. When this arrangement is used, the text is easily scrollable using the Scrollbars provided by the ScrolledWindow.

The proportion indicators on the scrollbars show relatively how much of the text is currently in the display.

As the user enters text, the view automatically scrolls when the insert point moves beyond a margin boundary (right or bottom) to keep the insert point in view.
**TextEdit**

**Application Access**

The **TextEdit** widget provides complete programmatic control of the text and its display.

**Keyboard Traversal**

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

The widget responds to the following keyboard navigation keys:

- **NEXT_FIELD** moves to the first control in the next group
- **PREV_FIELD** moves to the first control in the previous group
- **NEXTWINDOW** moves to the next window in the application
- **PREVWINDOW** moves to the previous window in the application
- **NEXTAPP** moves to the first window in the next application
- **PREVAPP** moves to the first window in the previous application
- **ROWUP** moves the caret up one line in the current column
- **ROWDOWN** moves the caret down one line in the current column
- **CHARBAK** moves the caret backward one character
- **CHARFWD** moves the caret forward one character
- **WORDFWD** moves the caret forward one word
- **WORDBAK** moves the caret back one word
- **LINESTART** moves the caret to the beginning of the line
- **LINEEND** moves the caret to the end of the line
- **PANESTART** moves the caret to the first row and column on the display
- **PANEEND** moves the caret to the last row and column on the display
- **DOCSTART** moves the caret to the beginning of the document
- **DOCEND** moves the caret to the end of the document

Note: It is expected that the user will use the alternate bindings for **NEXT_FIELD** and **PREV_FIELD** because the primary binding, TAB and SHIFT Tab, are valid characters in a Text pane.

The TextEdit widget responds to the following selection keys:

- **SELCHARFWD** adjusts the selection one character forward
- **SELWORDFWD** adjusts the selection to the end of the current (or next) word
- **SELLINEFWD** adjusts the selection to the end of the current (or next) line
- **SELCHARBAK** adjusts the selection one character backward
SELPARTBET

adjusts the selection to the beginning of the current (or previous) word
SELLINEBET

adjusts the selection to the beginning of the current (or previous) line
SELLINE

adjusts the selection to include the entire current line
SELFIPENDS

reverses the "anchor" and cursor position of the selection

The TextEdit widget responds to the following scrolling keys:
SCROLLUP

scroll the view up one screen line
SCROLLDOWN

scroll the view down one screen line
PAGEUP

scroll to the next page up
PAGEDOWN

scroll to the next page down
PAGERIGHT

scroll to the next page to the right
PAGELEFT

scroll to the next page to the left
SCROLLLEFT

scroll the view one screen to the left
SCROLLRIGHT

scroll the view one screen to the right
SCROLLTOP

scroll to the beginning of the document
SCROLLBOTTOM

scroll to the end of the document
SCROLLLEFTEDGE

scroll to the left edge of the document
SCROLLRIGHTEDGE

scroll to the right edge of the document

The TextEdit widget responds to the following edit keys:
DELCHARFWD

deletes the character to the right of the caret
DELCHARBAK

deletes the character to the left of the caret
DELWORDFWD

deletes the word to the right of the caret
DELWORDBAK

deletes the word to the left of the caret
DELLINEFWD

deletes to the end of the line from the caret
DELLINEBAK

deletes from the beginning of the line to the caret
DELLINE

deletes the line containing the caret
UNDO

undoes the previous edit operation

Coloration

When this widget receives the input focus, it changes the text caret in the text field to the active caret and the color specified in XtNinputFocusColor.

Display of Keyboard Mnemonic

The TextEdit widget does not display the mnemonic. If the TextEdit widget is the child of a Caption widget, the Caption widget can be used to display the mnemonic.
## RESOURCES

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XTCBackground</td>
<td>Pixel</td>
<td>XDefaultBackground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XTPixmap</td>
<td>Pixmap</td>
<td>Unspecified</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNblinkRate</td>
<td>XTCBlinkRate</td>
<td>Int</td>
<td>1000</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XTCBorderColor</td>
<td>Pixel</td>
<td>XDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XTPixmap</td>
<td>Pixmap</td>
<td>Unspecified</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XTCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNbottomMargin</td>
<td>XTCMargin</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcharsVisible</td>
<td>XTCCharsVisible</td>
<td>Int</td>
<td>50</td>
<td>GI</td>
</tr>
<tr>
<td>XtNcolormap</td>
<td>XTC colormap</td>
<td>Pointer</td>
<td>DYNAMIC</td>
<td>G</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XTCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcursorPosition</td>
<td>XTCCursorPosition</td>
<td>Int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XTCDepth</td>
<td>Int</td>
<td>DYNAMIC</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdisplayPosition</td>
<td>XTCCursorPosition</td>
<td>Int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNeditType</td>
<td>XTCEditType</td>
<td>OLEditType</td>
<td>OL_TEXT_EDIT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XTCFont</td>
<td>FontStruct</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XTCFontColor</td>
<td>Pixel</td>
<td>XDefaultForeground</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfontGroup</td>
<td>XTCFontGroup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XtNheight</td>
<td>XTCHeight</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XTCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinsertTab</td>
<td>XTCInsertTab</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNleftMargin</td>
<td>XTCMargin</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNlinesVisible</td>
<td>XTCLinesVisible</td>
<td>Int</td>
<td>16</td>
<td>GI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XTCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmargin</td>
<td>XTCallback</td>
<td>XTCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmodifyVerification</td>
<td>XTCallback</td>
<td>XTCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmotionVerification</td>
<td>XTCallback</td>
<td>XTCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNpostModifyNotification</td>
<td>XTCallback</td>
<td>XTCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtReferenceName</td>
<td>XTCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtReferenceWidget</td>
<td>XTCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtRightMargin</td>
<td>XTCMargin</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNselectEnd</td>
<td>XTCCursorPosition</td>
<td>Int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNselectStart</td>
<td>XTCCursorPosition</td>
<td>Int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNsensitive</td>
<td>XCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI</td>
</tr>
<tr>
<td>XTNsource</td>
<td>XTSource</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNsourceType</td>
<td>XTSourceType</td>
<td>OLSourceType</td>
<td>OL_STRING_SOURCE</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNtabTable</td>
<td>XTCtabTable</td>
<td>Pointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNtopMargin</td>
<td>XTCMargin</td>
<td>Dimension</td>
<td>4</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNtraversalOn</td>
<td>XTCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XTNuserData</td>
<td>XTCUserData</td>
<td>Pointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
</tbody>
</table>
**TextEdit Resource Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNWrapMode</td>
<td>XtCWrapMode</td>
<td>OWrapMode</td>
<td>OL_WRAP_WHITE_SPACE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

### XtNblinkRate
This resource is used to specify the rate that the active input caret blinks. The value of this resource is interpreted as the number of milliseconds between blinks. Setting this value to zero (0) turns the blink effect off.

### XtNbottomMargin
**Range of Values:**

\[
1 \leq \text{XtNbottomMargin} \leq \text{height} - \text{XtNtopMargin} - \text{font_height}
\]

This resource specifies the number of pixels used for the bottom margin. Note: the range is relative to the top margin value.

### XtNcharsVisible
This resource is used to specify the initial width of the text in terms of characters. It overrides the XtNwidth resource setting. The XtNwidth is recalculated to be the value of XtNcharsVisible multiplied by the width of the 'n' (en) character in the font plus the values for the left and right margins. The value of this resource changes to reflect the effects of geometry changes imposed by the widget tree and the user. SetValues for this resource is ignored.

### XtNcursorPosition
**Range of Values:**

\[
0 \leq \text{cursorPosition} < \text{size of buffer}
\]

This resource is used to specify the relative character position in the text source of the insert cursor. Changing the value of this resource may affect the XtNdisplayPosition resource if the cursorPosition is not visible in the pane.

### XtNdisplayPosition
**Range of Values:**

\[
0 \leq \text{displayPosition} < \text{size of buffer}
\]

This resource contains the position in the text source that will be displayed at the top of the pane. A value of 0 indicates the beginning of the text source. When the position is near the end of the buffer, this position is recalculated to ensure that the last line in the buffer appears as the last line in the pane.

### XtNeditType
**Range of Values:**

- OL_TEXT_READ/"textread"
- OL_TEXT_EDIT/"textedit"

This resource controls the edit state of the source:

- OL_TEXT_READ
  The source is read-only; the end-user cannot edit it.
The source is fully editable.

**XtNinputFocusColor**

Range of Values:

\[
\text{Valid Pixel value for the display} / (\text{Valid color name})
\]

This resource specifies the color of the input caret. Normally, this color is derived from the value of the **XtNinputFocusColor** resource and is dynamically maintained. This dynamic behavior is abandoned if the application explicitly sets this resource either at initialization or through a SetValues.

**XtNinsertTab**

If this resource is set to FALSE, a TAB character is not insertable. Setting this resource to FALSE makes traversal of the controls easier if the TAB key is bound as OL_NEXT_FIELD; if set to TRUE, a TAB character is insertable.

**XtNmargin**

This is the NULL terminated callback list of **XtCallbackList** used when the pane is redisplayed. The **call_data** parameter is a pointer to an **OlTextMarginCallData** structure:

```c
typedef enum {
    OL_MARGIN_EXPOSED,
    OL_MARGIN_CALCULATED,
} OlTextMarginHint;

typedef struct {
    OlTextMarginHint hint;
    XRectangle * rect;
} OlTextMarginCallData, *OlTextMarginCallDataPointer;
```

The **OlTextMarginHint** indicates whether the area to be redrawn was explicitly known because of an exposure event (OL_MARGIN_EXPOSED) or if the rectangle was calculated relative to the textual display (OL_MARGIN_CALCULATED). The margin callback should respond to the OL_MARGIN_EXPOSED hint by repainting the area defined by the XRectangle rect. The margin callback may wish to calculate its own rectangle in the OL_MARGIN_CALCULATED case. It can freely use the rectangle structure passed in with the call data for this purpose.

This callback can be used to repair the margins for the text.

For example, this callback may be used to display line numbers for the text in the left margin.

**XtNmodifyVerification**

This is the NULL terminated callback list of type **XtCallbackList** used when a modification of the buffer is attempted. The **call_data** parameter is a pointer to an **OlTextModifyCallData** structure:
typedef struct {
    Boolean   ok;
    TextPosition current_cursor;
    TextPosition select_start;
    TextPosition select_end;
    TextPosition new_cursor;
    TextPosition new_select_start;
    TextPosition new_select_end;
    char * text;
    int     text_length;
} OlTextModifyCallData, *OlTextModifyCallDataPointer;

All of the fields in this structure, with the exception of the ok flag, are treated as read-only information. The application can return without changing the value of the ok flag, in which case the update will occur. The application can also set the ok flag to FALSE, perform any other operations it desires, and return, in which case the update will not be performed.

**XtNmotionVerification**

This NULL terminated callback list of type XtCallbackList is used whenever the cursor position moves within the widget. The call_data parameter is a pointer to an OlTextMotionCallData structure:

typedef struct {
    Boolean   ok;
    TextPosition current_cursor;
    TextPosition new_cursor;
    TextPosition select_start;
    TextPosition select_end;
} OlTextMotionCallData, *OlTextMotionCallDataPointer;

This callback list is used whenever the cursor position changes due to cursor movement operations or by modification of the text.

The application can distinguish between a simple cursor movement and a modify operation by comparing the current_cursor and new_cursor values.

When these values are equal the callback is the result of a modify operation. In this case the ok field is ignored and the application should not attempt to perform updates to the text or its display during this callback.

If the values of current_cursor and new_cursor are different, the application is guaranteed that the operation is the result of a cursor movement. In this mode all of the fields in this structure, with the exception of the ok flag, are treated as read-only information. The application can return without changing the value of the ok flag, in which case the movement will occur. The application can also set the ok flag to FALSE, perform any other operations it desires, and return, in which case the movement will not be performed.
XtNpostModifyNotification
This is the NULL terminated callback list of type XtCallbackList used after a buffer update has completed. The call_data parameter is a pointer to an OlTextFocusCallData structure:

```c
typedef struct {
    Boolean requestor;
    TextPosition new_cursor;
    TextPosition new_select_start;
    TextPosition new_select_end;
    char * inserted;
    char * deleted;
    TextLocation delete_start;
    TextLocation delete_end;
    TextLocation insert_start;
    TextLocation insert_end;
    TextLocation cursor_position;
} OlTextFocusCallData, *OlTextFocusCallDataPointer;
```

This callback is used to synchronize the application with the text once a modify operation is completed. For example, the application may record successful edit operations in an undo buffer to provide multi-level undo functionality. The data provided in this callback is considered read-only and volatile (for example, the application should copy what it needs from this structure before returning).

XtNrightMargin
Range of Values:

\[ 1 \leq \text{XtNrightMargin} \leq \text{width} - \text{XtNleftMargin} - \text{font_width} \]

This resource specifies the number of pixels used for the right margin. Note: the range is relative to the left margin value.

XtNselectEnd
This resource is used to specify the last character position selected in the text. It is used along with the XtNselectStart and XtNcursorPosition resources to specify a selection. To be effective, the XtNselectStart value must be less than or equal to XtNselectEnd and the XtNcursorPosition must be equal to either XtNselectStart or XtNselectEnd. If either of these tests fails then XtNselectStart and XtNselectEnd are set equal to the value of the XtNcursorPosition.

XtNselectStart
This resource is used to specify the first character position selected in the text. It is used along with the XtNselectEnd and XtNcursorPosition resources to specify a selection. To be effective, the XtNselectStart value must be less than or equal to XtNselectEnd and the XtNcursorPosition must be equal to either XtNselectStart or XtNselectEnd. If either of these tests fails then XtNselectStart and XtNselectEnd are set equal to the value of the XtNcursorPosition.
XtNsource
Range of Values:
- <string> for OL_STRING_SOURCE
- name of file> for OL_DISK_SOURCE
- <pointer to an existing TextBuffer> for OL_TEXT_BUFFER_SOURCE

This resource is used in tandem with the XtNsourceType resource to specify the source. See the XtNsourceType resource for a full description of these resources.

XtNsourceType
Range of Values:
- OL_STRING_SOURCE/"stringsource"
- OL_DISK_SOURCE/"disksource"
- OL_TEXT_BUFFER_SOURCE/NA

This resource controls the interpretation of the XtNsource resource value. When set to OL_STRING_SOURCE the XtNsource value is interpreted as the string to be used as the source, when set to OL_DISK_SOURCE the XtNsource value is interpreted as the name of the file containing the source, when set to OL_TEXT_BUFFER_SOURCE the XtNsource value is interpreted as a pointer to a previously initialized TextBuffer (see the TextBuffer Utilities manual page for a description of TextBuffers).

XtNtabTable
This resource is used to specify a pointer to an array of tab Positions. These tab positions are specified in terms of pixels and must be terminated by a zero (0) entry. The widget calculates tabs by finding the next tab table entry that exceeds the current x offset for the line. If no such entry exists in the table or if the pointer to the tab table is NULL, the tab is set to the next greater multiple of eight (8) times the size of the 'n' (en) character in the font.

XtNtopMargin
Range of Values:
- 1 <= XtNtopMargin <= height - XtNbottomMargin - font_height

This resource specifies the number of pixels used for the top margin. Note: the range is relative to the bottom margin value.

XtNwrapMode
Range of Values:
- OL_WRAP_WHITE_SPACE/"wrapwhitespace"
- OL_WRAP_ANY/"wrapany"
- OL_WRAP_OFF/"wrapoff"

This resource is used to control how the source is wrapped in the pane. When OL_WRAP_ANY, lines are wrapped at the last character before the right margin; when set to OL_WRAP_WHITE_SPACE, lines are wrapped at the last white space before the right margin; as in OL_WRAP_ANY if the line does not contain any white space; when OL_WRAP_OFF, lines are not wrapped and the pane may horizontally scroll.
NAME

`TextField` – provides a one-line editable text field

SYNOPSIS

```c
#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <TextField.h>

widget = XtCreateWidget(name, textFieldWidgetClass, ...);
```

DESCRIPTION

**TextField Components**

The `TextField` widget is a one-line input field for text data that contains the following elements:

- Input Caret
- Input Field
- Left Arrow (conditional)
- Right Arrow (conditional)

![TextField Widget](image)

**Figure 1. One-Line Text Field**

**Keyboard Input**

Once the input focus has been moved to the Input Field, keyboard entry is allowed. The `TextField` widget does not validate the input, leaving that up to the application.
**Keyboard Traversal**

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

The TextField widget responds to the following keyboard navigation keys:

- NEXT FIELD and DOWN move to the next traversable widget in the window
- PREV FIELD and UP move to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application
- CHARBAK moves the caret backward one character
- CHARFWD moves the caret forward one character
- WORDFWD moves the caret forward one word
- WORDBAK moves the caret back one word
- LINESTART moves the caret to the beginning of the display
- LINEEND moves the caret to the end of the display

The MENUKEY posts the menu attached to the TextField.

The TextField widget responds to the following selection keys:

- SELCHARFWD adjusts the selection one character forward
- SELWORDFWD adjusts the selection to the end of the current (or next) word
- SELLINEFWD adjusts the selection to the end of the current (or next) line
- SELCHARBAK adjusts the selection one character backward
- SELWORDBAK adjusts the selection to the beginning of the current (or previous) word
- SELLINEBAK adjusts the selection to the beginning of the current (or previous) line
- SELLINE adjusts the selection to include the entire current line
- SELFLIPENDS reverses the "anchor" and cursor position of the selection

The TextField widget responds to the following scrolling keys:

- SCROLLLEFT scroll the view one screen to the left
- SCROLLRIGHT scroll the view one screen to the right
SCROLLLEFTEDGE
scroll to the left edge of the textfield

SCROLLRIGHTEDGE
scroll to the right edge of the textfield

The TextField widget responds to the following edit keys:

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

Display of Keyboard Mnemonic

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

Display of Keyboard Accelerators

The TextField does not respond to a keyboard accelerator because clicking the SELECT button on a TextField activates depending on the pointer position. So, the TextField does not display a keyboard accelerator.

Scrolling Long Text Input

If an input value exceeds the length of the Input Field, the Left Arrow and/or Right Arrow appear and the input value is visually truncated on the left and/or the right to show only as many characters as can fit in the Input Field. The truncation is at a character boundary. Since the Arrows take up space that would otherwise be used for the input, the truncation is more severe than would be necessary if they were not visible. An Arrow is present only if characters are hidden in the direction expressed by the arrow.

The user can scroll to show the hidden parts of the input by clicking or pressing SELECT on the Left or Right Arrow. Clicking SELECT on the Left Arrow scrolls the input one character to the right to show the next character that was hidden to the left. Clicking SELECT on the Right Arrow scrolls the input one character to the left to show the next character that was hidden to the right. Pressing SELECT scrolls continuously, with a user-adjustable wait between changes.

The text does not scroll beyond its limits, so that the left-most character never moves beyond the right edge of the TextField widget and the right-most character never moves beyond the left edge. If the user attempts to scroll beyond the limits by clicking SELECT, the system beeps. If the user is pressing SELECT when the limit is reached, the text stops scrolling but the system does not beep. If the user releases SELECT and then presses SELECT again to exceed the scrolling limit, the system beeps once regardless of how long SELECT is pressed.
**Input Validation**
A validation callback list can be used to perform limited per-field validation. This callback is used when the end-user hits the RETURN, PREV_FIELD, or NEXT_FIELD. It is not called if the user mouses the focus to another input area.

**Position of the Input Caret**
As characters are entered from the keyboard, the Input Caret moves to the right until it reaches the right end of the Input Field. As additional characters are typed the text scrolls to the left (the Left Arrow appears as discussed above) and the Input Caret moves relative to the text but remains stationary on the screen.

**Selecting and Operating on the Input Field**
The TextField widget allows text to be copied or moved to and from the Input Field. See TEXT SELECTION(3W) earlier in this manual for the description of these operations.

**Coloration**
When this widget receives the input focus, it changes the text caret in the text field to the active caret.

Figure 2 illustrates the resources that affect the coloration of the TextField widget.

![Diagram of TextField coloration](image)

**Figure 2. Text Field Coloration**
**SUBSTRUCTURE**
TextEdit component
Name: Textedit  
Class: TextEdit

**TextField Initialize Resources Passed from TextField Widget to TextEdit Widget**
At creation time, the following resources are passed from the textfield to the textedit widget:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNbackground</td>
<td>(textfield XtNbackground)</td>
</tr>
<tr>
<td>XtNbottomMargin</td>
<td>3</td>
</tr>
<tr>
<td>XtNcharsVisible</td>
<td>(depends on size of textfield)</td>
</tr>
<tr>
<td>XtNfont</td>
<td>(textfield font)</td>
</tr>
<tr>
<td>XtNinsertTab</td>
<td>(textfield XtNinsertTab)</td>
</tr>
<tr>
<td>XtNleftMargin</td>
<td>1</td>
</tr>
<tr>
<td>XtNlinesVisible</td>
<td>1</td>
</tr>
<tr>
<td>XtNRegisterFocusFunc</td>
<td>(textfield private routine EditsRegisterFocus())</td>
</tr>
<tr>
<td>XtNrightMargin</td>
<td>1</td>
</tr>
<tr>
<td>XtNtopMargin</td>
<td>1</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>(width of textfield)</td>
</tr>
<tr>
<td>XtNwrapMode</td>
<td>OL_WRAP_OFF</td>
</tr>
</tbody>
</table>

**TextField SetValues Resources Passed from TextField Widget to TextEdit Widget**
At *SetValues* time, *XtNstring* is passed to the TextEdit widget as *XtNsource* and the following resources are set:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNbackground</td>
<td>(TextField XtNbackground)</td>
</tr>
<tr>
<td>XtNcursorPosition</td>
<td>0</td>
</tr>
<tr>
<td>XtNinsertTab</td>
<td>(TextField XtNinsertTab)</td>
</tr>
<tr>
<td>XtNselectStart</td>
<td>0</td>
</tr>
<tr>
<td>XtNselectEnd</td>
<td>0</td>
</tr>
<tr>
<td>XtNsourceType</td>
<td>OL_STRING_SOURCE</td>
</tr>
</tbody>
</table>
**RESOURCES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>XtDefaultBackground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNcharsVisible</td>
<td>XtCCharsVisible</td>
<td>int</td>
<td>0</td>
<td>GI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SG</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent’s)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct *</td>
<td>(OPEN LOOK font)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>XtDefaultForeground</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Red</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinitialDelay</td>
<td>XtCInitialDelay</td>
<td>int</td>
<td>800</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinsertTab</td>
<td>XtCInsertTab</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaximumSize</td>
<td>XtCMaximumSize</td>
<td>int</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNreferenceName</td>
<td>XtCReferenceName</td>
<td>String</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNreferenceWidget</td>
<td>XtCReferenceWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>GI</td>
</tr>
<tr>
<td>XtNrepeatRate</td>
<td>XtCRepeatRate</td>
<td>int</td>
<td>100</td>
<td>SGI</td>
</tr>
<tr>
<td>XTnsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>GI</td>
</tr>
<tr>
<td>XtNstring</td>
<td>XtCString</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNextEditWidget</td>
<td>XtCTextEditWidget</td>
<td>Widget</td>
<td>NULL</td>
<td>G</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XlPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNverification</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

**XtNcharsVisible**

This resource is used to specify the initial width of the list in terms of characters. It overrides the **XtNwidth** resource setting. The **XtNwidth** is recalculated to be the value of **XtNcharsVisible** multiplied by the width of the 'n' (en) character in the font plus the values for the internal left and right margins. The value of this resource changes to reflect the effects of geometry changes imposed by the widget tree and the user. SetValues for this resource is ignored.

**XtNinitialDelay**

Range of Values:

\[ 0 < \text{XtNinitialDelay} \]
This resource gives the time, in milliseconds, before the first action occurs when SELECT is pressed on an Arrow. Note that millisecond timing precision may not be possible for all implementations, so the value may be rounded up to the nearest available unit by the toolkit.

**XtNinitialDelay**
This resource is used to specify the time in milliseconds of the initial repeat delay to be used when the scrolling arrows are pressed.

**XtNinsertTab**
If this resource is set to FALSE, a TAB character is not insertable; Setting this resource to FALSE makes traversal of controls easier if the TAB key is bound as OL_NEXT_FIELD. If set to TRUE, a TAB character is insertable.

**XtNmaximumSize**
Range of Values:
\[0 \leq \text{XtNmaximumSize}\]
This resource is the maximum number of characters that can be entered into the internal buffer. If this value is not set or is zero, the internal buffer will increase its size as needed limited only by the space limitations of the process.

**XtNreferenceWidget**
This resource specifies a position for inserting the current widget in its managing ancestor’s traversal list. If the reference widget is non-null and exists in the managing ancestor’s traversal list, the current widget will be inserted in front of it. Otherwise, the current widget will be inserted at the end of the list.

**XtNrepeatRate**
This resource is used to specify the time in milliseconds of the repeat delay to be used when the scrolling arrows are pressed.

**XtNstring**
This is the content of the Input Field. On being set a copy of the value is made in an internal buffer. Using **XtGetValues()** on this resource gets a new copy that the application is responsible for freeing when no longer needed.

**XtNTextEditWidget**
This resource is used to retrieve the TextEdit widget associated with the TextField. This value can be used to directly access the underlying TextEdit widget (and its TextBuffer) used to manage the textual display.

**XtNtraversalOn**
Range of Values:
\[
\text{TRUE} \quad \text{FALSE}
\]
This resource specifies whether this widget is selectable during traversal.

**XtNverification**
This is the callback list used when the end-user hits the RETURN, PREV_FIELD, or NEXT_FIELD. The call_data parameter is a pointer to an OTextFieldVerify structure.
typedef enum {
    OlTextFieldReturn,
    OlTextFieldPrevious,
    OlTextFieldNext
} OlTextVerifyReason;

typedef struct {
    OlTextVerifyReason reason;
    String string;
    Boolean ok;
    OlTextVerifyReason reason;
} OlTextFieldVerify, *OlTextFieldVerifyPointer;

This callback list can be used to perform per-field validation. Note: the callback is called only when a key is used to traverse from the field; it is not called when the user mouses the focus to another input area, therefore applications will still need to perform per-form validation.
# Manual Pages: Obsolete Routines

## Introduction to the Obsolete Routines

List of Obsolete Routines and Widgets  

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-1</td>
</tr>
</tbody>
</table>

## Obsolete Routines

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-2</td>
</tr>
</tbody>
</table>

Table of Contents

i
Introduction to the Obsolete Routines

The routines included in this section have been made obsolete for various reasons. Usually they have been replaced with more updated versions. The documentation for these routines has been made available to you for three reasons:

- We wish to remain consistent with Intronics documentation from other sources including the X Consortium.
- You are given an opportunity to remove these obsolete routines from your application program conveniently.
- You have the opportunity to compare the description of the old routine with its replacement.

List of Obsolete Routines and Widgets

The obsolete routines and widgets for this release are as follows:

- Abbrevstack Widget
- Buttonstack Widget
- LookupOlInputEvent()
- OlDetermineMouseAction()
- OlGetApplicationResources()
- OlReplayBtnEvent()
- Text Widget
- Virtual Key/Button
NAME

AbbrevStack — no longer used

The AbbreviatedButtonStack Widget is no longer an X11 toolkit widget, but it will be supported for compatibility reasons. Since it has the exact functionality and resources as the AbbreviatedMenuButton widget, refer to the AbbreviatedMenuButton(3W) manual page for a discussion of its use.
NAME

ButtonStack – no longer used

The ButtonStack widget is no longer an Xol toolkit widget, but it will be supported for compatibility reasons. Since it has the exact functionality and resources as the MenuButton widget, refer to the MenuButton(3W) manual page for a discussion of its use.
NAME
LookupOlInputEvent

SYNOPSIS
#include <Dynamic.h>

extern OlInputEvent LookupOlInputEvent(w, event, keysym, buffer, length)

Widget w;
XEvent * event;
KeySym * keysym;
char ** buffer;

DESCRIPTION
The LookupOlInputEvent function is used to decode the event for widget w to an OlInputEvent. The event passed should be a ButtonPress, ButtonRelease, or KeyPress event. The function attempts to decode this event based on the settings of the OPEN LOOK defined dynamic mouse and keyboard settings.

If the event is a KeyPress, the function may return the keysym, buffer, and/or length of the buffer returned from a call to XLookupString(3X). It returns these values if non-NULL values are provided by the caller.

SEE ALSO
LookupOlColors(3W)
OlReplayBtnEvent(3W)
OlDetermineMouseAction(3W)
**OIDetermineMouseAction (3W)**

**Dynamic Settings**

**NAME**

OIDetermineMouseAction

**SYNOPSIS**

```c
#include <OpenLook.h>
#include <Dynamic.h>
...
extern ButtonAction OIDetermineMouseAction(w, event)
Widget w;
XEvent * event;
```

**DESCRIPTION**

The `OIDetermineMouseAction` function is used to determine the kind of mouse gesture that is being attempted: MOUSE_CLICK, MOUSE_MULTI_CLICK, or MOUSE_MOVE. This function is normally called immediately upon receipt of a mouse button press event. It uses the current settings for mouseDampingFactor and multiClickTimeout to determine the kind of gesture being made.

**SEE ALSO**

OIDragAndDrop(3W)
OIGrabDragPointer(3W)
OIUngrabDragPointer(3W)

**NOTES**

This function performs an active pointer grab. This grab is released for the CLICK type actions but not for MOUSE_MOVE. It is the responsibility of the caller to ungrab the pointer if the action is MOUSE_MOVE.

**EXAMPLE**

```c
static void ButtonConsumeCB (w, client_data, call_data)
widget w;
XtPointer client_data;
XtPointer call_data;
{
  Window drop_window;
  Position x;
  Position y;
  OIVirtualEvent ve;

  ve = (OIVirtualEvent) call_data

  switch (ve -> virtual_name)
  {
    case OL_SELECT:
      switch(OIDetermineMouseAction(widget, event))
```
{ 
    case MOUSE_MOVE:­
        OlGrabDragPointer(widget, OlGetMoveCursor(XtScreen(widget),
                    None);
        OlDragAndDrop(widget, &drop_window, &x, &y);
        DropOn(widget, drop_window, x, y, ....));
        OlUngrabDragPointer(widget);
        break;
    case MOUSE_CLICK:­
        ClickSelect(widget, ....);
        break;
    case MOUSE_MULTI_CLICK:­
        MultiClickSelect(widget, ....);
        break;
    default:­
        Panic(widget, ....);
        break;
}
break;
default:­
    OlReplayBtnEvent(widget, NULL, event);
    break;
}
NAME
OlReplayBtnEvent

SYNOPSIS

extern void OlReplayBtnEvent(w, client_data, event)
Widget w;
caddr_t client_data;
XEvent * event;

DESCRIPTION
The OlReplayBtnEvent procedure is used to replay a button press event to the
next window (towards the root) that is interested in button events. This provides a
means of propagating events up a window tree.

SEE ALSO
LookupOlInputEvent(3W)
Virtual Key/Button: OlConvertVirtualTranslation  - takes a virtual translation string as input and returns a Standard X Toolkit translation string

**Note:** this routine is obsolete since translation tables cannot support dynamic rebinding of keys and buttons. OlLookupInputEvent should be used instead.

**SYNOPSIS**

```c
#include <Intrinsic.h>
#include <OpenLook.h>

char *OlConvertVirtualTranslation(virtual_translation)
char *virtual_translation;
```

**DESCRIPTION**

**Converts Virtual Translations into Standard Translations**

OlConvertVirtualTranslation takes a virtual translation string as input and returns a standard X Toolkit Intrinsic's translation string. The function parses the input string replacing any virtual key or button expressions with their real representation(s). A virtual translation string has the same format as a standard X Toolkit Intrinsics translation string, except that virtual expressions can appear as modifiers or event types. (However, a virtual expression cannot appear as an event type detail.)


**Virtual Expressions**

Virtual expressions are key or mouse button names that are independent of any physical mapping of the keyboard or mouse buttons. For instance, MENU (which in this document refers to the mouse button used to pop up menus) has the virtual expression `menuBtn` in a virtual translation. If the end user has assigned mouse button three to be MENU, then the OlConvertVirtualTranslation routine would convert a virtual translation that contains the expression `menuBtn` into a standard translation containing the expression `Button3`.

The following is the list of virtual expressions for the various keys and mouse buttons defined in the OPEN LOOK user interface. The left column gives the name used in this document.

<table>
<thead>
<tr>
<th>Name</th>
<th>Virtual Expression</th>
<th>Virtual Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUST</td>
<td>adjustBtn</td>
<td>OL_ADJUST</td>
</tr>
<tr>
<td>ADJUSTKEY</td>
<td>adjustKey</td>
<td>OL_ADJUSTKEY</td>
</tr>
<tr>
<td>CANCEL</td>
<td>cancelKey</td>
<td>OL_CANCEL</td>
</tr>
<tr>
<td>CHARBACK</td>
<td>charBakKey</td>
<td>OL_CHARBAK</td>
</tr>
<tr>
<td>CHARFWD</td>
<td>charFwdKey</td>
<td>OL_CHARFWD</td>
</tr>
<tr>
<td>CONSTRAIN</td>
<td>constrainBtn</td>
<td>OL_CONSTRAIN</td>
</tr>
<tr>
<td>COPY</td>
<td>copyKey</td>
<td>OL_COPY</td>
</tr>
<tr>
<td>Name</td>
<td>Virtual Expression</td>
<td>Virtual Event</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>CUT</td>
<td>cutKey</td>
<td>OL CUT</td>
</tr>
<tr>
<td>DEFAULTACTION</td>
<td>defaultActionKey</td>
<td>OL DEFAULTACTION</td>
</tr>
<tr>
<td>DELCHARBACK</td>
<td>delCharBakKey</td>
<td>OL DELCHARBAK</td>
</tr>
<tr>
<td>DELCHARFWD</td>
<td>delCharFwdKey</td>
<td>OL DELCHARFWD</td>
</tr>
<tr>
<td>DELLINE</td>
<td>delLineKey</td>
<td>OL DELLINE</td>
</tr>
<tr>
<td>DELLINEBACK</td>
<td>delLineBakKey</td>
<td>OL DELLINEBAK</td>
</tr>
<tr>
<td>DELLINEFWD</td>
<td>delLineFwdKey</td>
<td>OL DELLINEFWD</td>
</tr>
<tr>
<td>DELWORDBACK</td>
<td>delWordBakKey</td>
<td>OL DELWORDBAK</td>
</tr>
<tr>
<td>DELWORDFWD</td>
<td>delWordFwdKey</td>
<td>OL DELWORDFWD</td>
</tr>
<tr>
<td>DOCEND</td>
<td>docEndKey</td>
<td>OL DOCEND</td>
</tr>
<tr>
<td>DOCSTART</td>
<td>docStartKey</td>
<td>OL DOCSTART</td>
</tr>
<tr>
<td>DOWN</td>
<td>downKey</td>
<td>OL DOWN</td>
</tr>
<tr>
<td>DRAG</td>
<td>dragKey</td>
<td>OL DRAG</td>
</tr>
<tr>
<td>DROP</td>
<td>dropKey</td>
<td>OL DROP</td>
</tr>
<tr>
<td>DUPLICATE</td>
<td>duplicateBtn</td>
<td>OL DUPLICATE</td>
</tr>
<tr>
<td>DUPLICATEKEY</td>
<td>duplicateKey</td>
<td>OL DUPLICATEKEY</td>
</tr>
<tr>
<td>HELP</td>
<td>helpKey</td>
<td>OL HELP</td>
</tr>
<tr>
<td>HORIZSBMENU</td>
<td>horizSDMenuKey</td>
<td>OL HSBMENU</td>
</tr>
<tr>
<td>LEFT</td>
<td>leftKey</td>
<td>OL LEFT</td>
</tr>
<tr>
<td>LINEEND</td>
<td>lineEndKey</td>
<td>OL LINEEND</td>
</tr>
<tr>
<td>LINSTART</td>
<td>lineStartKey</td>
<td>OL LINSTART</td>
</tr>
<tr>
<td>MENU</td>
<td>menuBtn</td>
<td>OL MENU</td>
</tr>
<tr>
<td>MENUEKEY</td>
<td>menuKey</td>
<td>OL MENUEKEY</td>
</tr>
<tr>
<td>MENUDEFAULT</td>
<td>menuDefaultBtn</td>
<td>OL MENUDEFAULT</td>
</tr>
<tr>
<td>MENUDEFAULTKEY</td>
<td>menuDefaultKey</td>
<td>OL MENUDEFAULTKEY</td>
</tr>
<tr>
<td>NEXTAPP</td>
<td>nextAppKey</td>
<td>OL NEXTAPP</td>
</tr>
<tr>
<td>NEXTFIELD</td>
<td>nextFieldKey</td>
<td>OL NEXT</td>
</tr>
<tr>
<td>NEXTWINDOW</td>
<td>nextWinKey</td>
<td>OL NEXTWINDOW</td>
</tr>
<tr>
<td>PAGEDOWN</td>
<td>pageDownKey</td>
<td>OL PAGEDOWN</td>
</tr>
<tr>
<td>PAGELEFT</td>
<td>pageLeftKey</td>
<td>OL PAGELEFT</td>
</tr>
<tr>
<td>PAGERIGHT</td>
<td>pageRightKey</td>
<td>OL PAGERIGHT</td>
</tr>
<tr>
<td>PAGEUP</td>
<td>pageUpKey</td>
<td>OL PAGEUP</td>
</tr>
<tr>
<td>PAN</td>
<td>panBtn</td>
<td>OL PAN</td>
</tr>
<tr>
<td>PANEEND</td>
<td>paneEndKey</td>
<td>OL PANEEND</td>
</tr>
<tr>
<td>PANSTART</td>
<td>paneStartKey</td>
<td>OL PANSTART</td>
</tr>
<tr>
<td>PASTE</td>
<td>pasteKey</td>
<td>OL PASTE</td>
</tr>
<tr>
<td>PREVAPP</td>
<td>prevAppKey</td>
<td>OL PREVAPP</td>
</tr>
<tr>
<td>PREVFIELD</td>
<td>prevFieldKey</td>
<td>OL PREV</td>
</tr>
<tr>
<td>PREVWINDOW</td>
<td>prevWinKey</td>
<td>OL PREVWINDOW</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>propertiesKey</td>
<td>OL PROPERTY</td>
</tr>
<tr>
<td>RIGHT</td>
<td>rightKey</td>
<td>OL RIGHT</td>
</tr>
<tr>
<td>ROWDOWN</td>
<td>rowDownKey</td>
<td>OL ROWDOWN</td>
</tr>
</tbody>
</table>
Virtual Key/Button (3W)  

<table>
<thead>
<tr>
<th>Name</th>
<th>Virtual Expression</th>
<th>Virtual Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWUP</td>
<td>rowUpKey</td>
<td>OL ROWUP</td>
</tr>
<tr>
<td>SCROLLBOTTOM</td>
<td>scrollBottomKey</td>
<td>OL SCROLLBOTTOM</td>
</tr>
<tr>
<td>SCROLLDOWN</td>
<td>scrollDownKey</td>
<td>OL SCROLLDOWN</td>
</tr>
<tr>
<td>SCROLLLEFT</td>
<td>scrollLeftKey</td>
<td>OL SCROLLLEFT</td>
</tr>
<tr>
<td>SCROLLLEFTEDGE</td>
<td>scrollLeftEdgeKey</td>
<td>OL SCROLLLEFTEDGE</td>
</tr>
<tr>
<td>SCROLLRIGHT</td>
<td>scrollRightKey</td>
<td>OL SCROLLRIGHT</td>
</tr>
<tr>
<td>SCROLLRIGHTEDGE</td>
<td>scrollRightEdgeKey</td>
<td>OL SCROLLRIGHTEDGE</td>
</tr>
<tr>
<td>SCROLLTOP</td>
<td>scrollTopKey</td>
<td>OL SCROLLTOP</td>
</tr>
<tr>
<td>SCROLLUP</td>
<td>scrollUpKey</td>
<td>OL SCROLLUP</td>
</tr>
<tr>
<td>SELECT</td>
<td>selectBtn</td>
<td>OL SELECT</td>
</tr>
<tr>
<td>SELECTKEY</td>
<td>selectKey</td>
<td>OL SELECTKEY</td>
</tr>
<tr>
<td>SELECTCHARBACK</td>
<td>selCharBakKey</td>
<td>OL SELCHARBAK</td>
</tr>
<tr>
<td>SELECTFLIPENDS</td>
<td>selFlipEndsKey</td>
<td>OL_SELFLIPENDS</td>
</tr>
<tr>
<td>SELECTLINE</td>
<td>selLineKey</td>
<td>OL SELLINE</td>
</tr>
<tr>
<td>SELECTLINEBACK</td>
<td>selLineBakKey</td>
<td>OL_SELLINEBAK</td>
</tr>
<tr>
<td>SELECTWORDBACK</td>
<td>selWordBakKey</td>
<td>OL_SELWORKBAK</td>
</tr>
<tr>
<td>SELECTCHARFWD</td>
<td>selCharFwdKey</td>
<td>OL_SELCHARFWD</td>
</tr>
<tr>
<td>SELECTLINEFWD</td>
<td>selLineFwdKey</td>
<td>OL_SELLINEFWD</td>
</tr>
<tr>
<td>SELECTWORDFWD</td>
<td>selWordFwdKey</td>
<td>OL_SELWORDFWD</td>
</tr>
<tr>
<td>STOP</td>
<td>stopKey</td>
<td>OL STOP</td>
</tr>
<tr>
<td>TOGGLEPUSHPIN</td>
<td>togglePushpinKey</td>
<td>OL_TOGGLEPUSHPIN</td>
</tr>
<tr>
<td>UNDO</td>
<td>undoKey</td>
<td>OL UNDO</td>
</tr>
<tr>
<td>UP</td>
<td>upKey</td>
<td>OL UP</td>
</tr>
<tr>
<td>VERTSBMENU</td>
<td>vertSBMenuKey</td>
<td>OL VSBMENU</td>
</tr>
<tr>
<td>WINDOWMENU</td>
<td>windowMenuKey</td>
<td>OL WINDOWMENU</td>
</tr>
<tr>
<td>WORDBACK</td>
<td>wordBakKey</td>
<td>OL WORDBAK</td>
</tr>
<tr>
<td>WORDFWD</td>
<td>wordFwdKey</td>
<td>OL WORDFWD</td>
</tr>
<tr>
<td>WORKSPACEMENU</td>
<td>workspaceMenuKey</td>
<td>OL WORKSPACEMENU</td>
</tr>
</tbody>
</table>

The following example illustrates a virtual translation that contains three productions. (The Message production does not involve any virtual translations but is included for illustration.)

```
" ! selectBtn <selectBtnUp>:  notify() \n <Message>:  checkClientMessage() \n ! <copyKeyDown>:  clipToClipboard()"
```

**Space Allocated Only If Necessary**

If the input string does not contain any virtual expressions, the original string is returned. If the input string contains at least one valid virtual production, the function allocates memory for the returned string. The application is responsible for checking this difference and freeing the allocated memory, if appropriate.

**Invalid Productions Dropped**

If the input string contains a valid virtual production, any invalid virtual productions are not included in the returned string. If the input string contains no virtual productions, error checking is not done.
SEE ALSO
OlLookupInputEvent(3W)
NAME

Text – provides an interface for the end-user to enter and edit text

Note: The Text widget is obsolete -- use the TextEdit widget instead.

SYNOPSIS

#include <Intrinsic.h>
#include <StringDefs.h>
#include <OpenLook.h>
#include <Text.h>

widget = XtCreateWidget(name, textWidgetClass, ...);

DESCRIPTION

The Text widget provides a single and multi-line text editor that has both a customizable user interface and a programmatic interface. It can be used for single-line string entry, forms entry with verification procedures, multiple-page document viewing, and full-window editing. It provides an application with a consistent editing paradigm for entry of textual data.

The display of the textual data on the screen can be adjusted to scroll, wrap, or grow automatically as the user reaches the edge of the view of the text.

The Text widget provides separate callback lists to verify insertion cursor movement, modification of the text, and leaving the Text widget. Each of these callbacks provides the verification function with the widget instance, the event that caused the callback, and a data structure specific to the verification type. From this information, the function can verify if the application considers this to be a legitimate state change and signal the widget whether to continue with the action. The verification function can also manipulate the widget through the class methods defined by the Text widget class. The verification callback lists are explained in detail below.

Editing Capabilities

The Text widget provides the editing capabilities listed in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Editing Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARFWD</td>
<td>Move the caret forward one character</td>
</tr>
<tr>
<td>CHARBAK</td>
<td>Move the caret back one character</td>
</tr>
<tr>
<td>ROWDOWN</td>
<td>Move the caret down one line in the current column</td>
</tr>
<tr>
<td>ROWUP</td>
<td>Move the caret up one line in the current column</td>
</tr>
<tr>
<td>WORDFWD</td>
<td>Move the caret forward one word</td>
</tr>
<tr>
<td>WORDBAK</td>
<td>Move the caret back one word</td>
</tr>
<tr>
<td>LINESTART</td>
<td>Move the caret to the beginning of the current display line</td>
</tr>
<tr>
<td>LINEEND</td>
<td>Move the caret to the end of the current display line</td>
</tr>
<tr>
<td>DOCSTART</td>
<td>Move the caret to the beginning of the source</td>
</tr>
<tr>
<td>DOCEND</td>
<td>Move the caret to the end of the source</td>
</tr>
<tr>
<td>DELCHARFWD</td>
<td>Delete the character to the right of the caret</td>
</tr>
<tr>
<td>DELCHARBAK</td>
<td>Delete the character to the left of the caret</td>
</tr>
</tbody>
</table>
This second table displays the virtual expressions and keyboard equivalents for the editing functions. See VIRTUAL KEY/BUTTON(3W) earlier in this guide for more information on virtual expressions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Virtual Expression</th>
<th>Keyboard Equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARFWD</td>
<td>charFwdKey</td>
<td>CTRL-F, →</td>
</tr>
<tr>
<td>CHARBAK</td>
<td>charBakKey</td>
<td>CTRL-B, ←</td>
</tr>
<tr>
<td>ROWDOWN</td>
<td>rowDownKey</td>
<td>CTRL-N, ↓</td>
</tr>
<tr>
<td>ROWUP</td>
<td>rowUpKey</td>
<td>CTRL-P, ↑</td>
</tr>
<tr>
<td>WORDFWD</td>
<td>wordFwdKey</td>
<td>ALT-F, ALT-→</td>
</tr>
<tr>
<td>WORDBAK</td>
<td>wordBakKey</td>
<td>ALT-B, ALT-←</td>
</tr>
<tr>
<td>LINESTART</td>
<td>lineStartKey</td>
<td>CTRL-A, CTRL-←</td>
</tr>
<tr>
<td>LINEEND</td>
<td>lineEndKey</td>
<td>CTRL-E, CTRL-→</td>
</tr>
<tr>
<td>DOCSTART</td>
<td>docStartKey</td>
<td>ALT-↑, ALT-&lt;</td>
</tr>
<tr>
<td>DOCEND</td>
<td>docEndKey</td>
<td>ALT-↓, ALT-&gt;</td>
</tr>
<tr>
<td>DELCHARFWD</td>
<td>delCharFwdKey</td>
<td>CTRL-D, DELETE</td>
</tr>
<tr>
<td>DELCHARBAK</td>
<td>delCharBakKey</td>
<td>CTRL-H, BACKSPACE</td>
</tr>
<tr>
<td>DELWRDFWD</td>
<td>delWordFwdKey</td>
<td>ALT-D</td>
</tr>
<tr>
<td>DELWORDBAK</td>
<td>delWordBakKey</td>
<td>ALT-H</td>
</tr>
<tr>
<td>DELLINEFWD</td>
<td>delLineFwdKey</td>
<td>CTRL-K</td>
</tr>
<tr>
<td>DELLINEBAK</td>
<td>delLineBakKey</td>
<td>ALT-K</td>
</tr>
</tbody>
</table>

Hierarchical Text

Text is considered to be hierarchically composed of white space, words, lines and paragraphs. White space is defined as any non-empty sequence of the ASCII characters space, tab, linefeed or carriage return (decimal values of 32, 9, 10, 13, respectively); a word is any non-empty sequence of characters bounded on both sides by whitespace. A source line is any (possibly empty) sequence of characters bounded by newline characters; a display line is any (possibly empty) sequence of characters appearing on a single screen display line. A source paragraph is any sequence of characters bounded by sets of two or more adjacent newline characters. A display paragraph is any (possibly empty) sequence of characters bounded by newline characters (Note: This is identical to the definition of a source line.)

In all cases, the beginning or end of the edit text is an acceptable bounding element in the previous definitions.
Sizing the Display
When making display decisions, the Text widget first determines whether all the text will fit in the current display. If it does not, and growing is enabled, the widget will request a resize from its parent. If the request is denied or only partially satisfied, no future growth requests will be made unless there is an intervening resize operation externally imposed.

If any source line is still too long to fit in the display after growing is attempted, wrapping is checked. If wrapping is disabled, one display line is drawn for each source line. If a source line is too long for the display, it is truncated at the right margin after the last full character that fits. If wrapping is enabled, a new display line will be started with the first word that does not fit on the current line. If the wrap break option is OL_WRAP_ANY, as many characters from that word as will fit before the right margin are written to the current display line, then the next character starts at the left margin of the next display line. If the wrap break option is OL_WRAP_WHITE_SPACE, the line break is instead made after the first whitespace character that follows the last full word that does fit on the current display line. If, however, under white space break, the first full word that does not fit is also the first word on the line, the wrap break is made as if OL_WRAP_ANY were selected.

Scrolled Window
The application can decide if the Text widget can have a scrollbar at the side. With a scrollbar, the end user can move through the text easily. Without a scrollbar, the Text widget either grows its window, if possible, to show the complete content, or wraps a long line onto another line.

The proportion indicators on the scrollbar show how much of the text the user can see at once, compared with the entire text buffer or file.

As the user enters text, the view will automatically scroll left to keep the insert point in view, unless the Text widget is operating in a wrap or grow mode.

Application Callbacks
Three types of verification callbacks are supported by the Text widget:
- one for motion operations, to verify a new insert position;
- one for modifying operations, to verify insertion, deletion or replacement of text; and
- one for widget exit, to verify state consistency on loss of focus by the widget. The call_data value is a pointer to an OlTextVerifyCD structure. The C data types used here are:
typedef enum {
    motionVerify,
    modVerify,
    leaveVerify
} OlVerifyOpType;

typedef struct {
    int firstPos;
    int length;
    unsigned char *ptr;
} OlTextBlock, *OlTextBlockPtr;

typedef struct {
    XEvent *xevent;
    OlVerifyOpType operation;
    Boolean doit;
    OlTextPosition currInsert, newInsert;
    OlTextPosition startPos, endPos;
    OlTextBlock *text;
} OlTextVerifyCD, *OlTextVerifyPtr;

The elements of an OlTextBlock structure are as follows:
- **firstPos** the offset of the starting character in the text block.
- **length** the size of the text block.
- **ptr** a pointer to the text block.

Before the verification callbacks are issued for any given operation, a structure of type OlTextVerifyCD is initialized. The initial values are:
- **xevent** for a leave operation, the current event pointer.
- **operation** element of OlVerifyOpType signifying the type of verification operation.
- **doit** TRUE.
- **currInsert** current position of the insert point.
- **newInsert** for a motion operation, the position the user is attempting to move the insert point to; otherwise, the same value as **currInsert**.
- **startPos** for a modify operation, the beginning position in the current source of the text about to be deleted or replaced, or where new text will be inserted. If not a modify operation, it will have the same value as **currInsert**.
- **endPos** for a modify operation, the ending position in the current source of the text about to be deleted or replaced. If no text is being removed, it will have the same value as **startPos**. If not a modify operation, the same value as **currInsert**.
text for a modify operation with new text to be inserted, a pointer to a structure of type OlTextBlock, that references the text to be inserted. Otherwise, NULL.

It is possible for the client to register more than one callback procedure for any of these callback types. Since there can be more than one callback, each verification procedure should first check the doit field.

On return from the last callback, the Text widget will look at the doit member of the OlTextVerifyCD structure. If it is false, a callback has already rejected the operation, so there is no need for further evaluation. If it is still true, the Text widget will proceed with the operation; otherwise, it will not. Any user feedback for the rejected operation is the responsibility of the verification procedure.

Verification callbacks are permitted to modify some of the data in the OlTextVerifyCD structure. The Text widget will only look at certain fields on return, though, according to the operation type:

- For a motion operation, only the newInsert position will be looked at.
- For a modify operation, only startPos, endPos, and text will be examined for changes.
- For a leave operation, no fields will be examined.

There is no mechanism for preventing a verification callback from making other changes to the editing state through the documented interface, but the results of such behind-the-back actions are undefined.

**Application Access to Text**

The Text has several resources that identify entry points that the application can use to access the internal buffer that the Text widget manages. For example, if the widget is being used to enter a string, the program can get a copy of the string (that is, the internal buffer) with the function under the resources `XtNtextCopyBuffer` or `XtNtextReadSubString`.

**Selecting and Copying the Text**

Text can be moved or copied to and from the Text widget. See TEXT SELECTION(3W) earlier in this manual for the description of these operations.

**Coloration**

When this widget receives the input focus, it changes the text caret in the text field to the active caret.

Figure 1 illustrates the resources that affect the coloration of the Text widget.
Figure 1. Text Coloration

**SUBSTRUCTURE**

**Vertical Scrollbar component**

Names: verticalscrollbar
Class: Scrollbar

See the regular resource list for alternate names used for some key **Scrollbar** resources.

**RESOURCES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>G*</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>Pixel</td>
<td>White</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>Pixmap</td>
<td>(none)</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNbottomMargin</td>
<td>XtCMargin</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNconsumeEvent</td>
<td>XtCConsumeEvent</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcursorPosition</td>
<td>XtCCursorPosition</td>
<td>OITextPosition</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNcurrentPage</td>
<td>XtCCurrentPage</td>
<td>int</td>
<td>1</td>
<td>SGI</td>
</tr>
<tr>
<td>Name</td>
<td>Class</td>
<td>Type</td>
<td>Default</td>
<td>Access</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>int</td>
<td>(parent's)</td>
<td>GI</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNdisplayPosition</td>
<td>XtCPosition</td>
<td>OITextPosition</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNeditType</td>
<td>XtCEditType</td>
<td>OlDefine</td>
<td>OL_TEXT_EDIT</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfile</td>
<td>XtCFile</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNfont</td>
<td>XtCFont</td>
<td>XFontStruct *</td>
<td>(OPEN LOOK font)</td>
<td>SI</td>
</tr>
<tr>
<td>XtNfontColor</td>
<td>XtCFontColor</td>
<td>Pixel</td>
<td>Black*</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>XtCForeground</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI†</td>
</tr>
<tr>
<td>XtNgrow</td>
<td>XtCGrow</td>
<td>OlDefine</td>
<td>OL_GROW_OFF</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNhorizontalSB</td>
<td>XtCHorizontalSB</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNinputFocusColor</td>
<td>XtCInputFocusColor</td>
<td>Pixel</td>
<td>Black</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNleaveVerification</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNleftMargin</td>
<td>XtCMargin</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmaximumSize</td>
<td>XtCLength</td>
<td>int</td>
<td>(none)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNmodifyVerification</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNmotionVerification</td>
<td>XtCCallback</td>
<td>XtCallbackList</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>XtNrecomputeSize</td>
<td>XtCRecomputeSize</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNrightMargin</td>
<td>XtCMargin</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNshowPage</td>
<td>XtCShowPage</td>
<td>OlDefine</td>
<td>OL_NONE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNsourcetype</td>
<td>XtCSourceType</td>
<td>OlDefine</td>
<td>OL_STRING_SRC</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNstring</td>
<td>XtCString</td>
<td>String</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtextClearBuffer</td>
<td>XtCTextClearBuffer</td>
<td>void(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextCopyBuffer</td>
<td>XtCTextCopyBuffer</td>
<td>unsigned char(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextGetInsertPoint</td>
<td>XtCTextGetInsertPoint</td>
<td>OITextPosition(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextGetLastPos</td>
<td>XtCTextGetLastPos</td>
<td>OITextPosition(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextInsert</td>
<td>XtCTextInsert</td>
<td>void(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtextReadSubStr</td>
<td>XtCTextReadSubStr</td>
<td>int(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextRedraw</td>
<td>XtCTextRedraw</td>
<td>void(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextReplace</td>
<td>XtCTextReplace</td>
<td>int</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextSetInsertPoint</td>
<td>XtCTextSetInsertPoint</td>
<td>void(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtextUpdate</td>
<td>XtCTextUpdate</td>
<td>void(*)</td>
<td>(n/a)</td>
<td>G</td>
</tr>
<tr>
<td>XtNtopMargin</td>
<td>XtCMargin</td>
<td>Dimension</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNtraversalOn</td>
<td>XtCTraversalOn</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNuserData</td>
<td>XtCUserData</td>
<td>XtPointer</td>
<td>NULL</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNverticalSB</td>
<td>XtCVerticalSB</td>
<td>Boolean</td>
<td>FALSE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNviewHeight</td>
<td>XtCViewHeight</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>Dimension</td>
<td>(calculated)</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNwrap</td>
<td>XtCWrap</td>
<td>Boolean</td>
<td>TRUE</td>
<td>SGI</td>
</tr>
</tbody>
</table>
### Text Resource Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Type</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNwrapBreak</td>
<td>XtCWrapBreak</td>
<td>OIDefine</td>
<td>OL_WRAP_WHITE_SPACE</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>Position</td>
<td>0</td>
<td>SGI</td>
</tr>
</tbody>
</table>

### XtNbottomMargin

Range of Values:

\[
0 \leq \text{XtNbottomMargin}
\]

This resource is the number of pixels used for the bottom margin.

### XtNcursorPosition

Range of Values:

\[
0 \leq \text{XtNcursorPosition} < (\text{current size of the text})
\]

This resource is the position in the text source of the insert cursor.

### XtNdisplayPosition

Range of Values:

\[
0 \leq \text{XtNdisplayPosition}
\]

This resource contains the position in the text source that will be displayed at the top of the screen. A value of 0 indicates the start of the text source.

**Note:**
The specified position must correspond to the first character position of a source line (that is, it must be 0, or it must be a position immediately following a newline character). Otherwise, correct behavior is not guaranteed.

### XtNeditType

Range of Values:

- OL_TEXT_READ/"read"
- OL_TEXT_EDIT/"edit"

This resource controls the edit state of the source:

- **OL_TEXT_READ** The source is read-only; the end user cannot edit it.
- **OL_TEXT_EDIT** The source is fully editable.

**Note:**
This option is available for text buffers only; text files cannot be edited.

### XtNfile

This resource is used only if the *XtNsourceType* resource has the value OL_DISK_SOURCE. It is the absolute pathname of a disk file to be viewed.

### XtNfont

Range of Values:

(any valid return from XLoadQueryFont())

**Default:**

(chosen to match the scale and screen resolution)
This resource identifies the font to be used to display the text.

The default value points to a cached font structure; an application should not expect to get this value with a call to XtGetValue() and use it reliably thereafter.

**XtNfontColor**

Range of Values:

(any Pixel value valid for the current display)/(any name from the rgb.txt file)

This resource specifies the color for the font. If not set, the color from the XtNforeground resource, if available, is used for the font.

See the note about the interaction of this resource with other color resources under the description of the XtNbackground resource in CORE(3W).

**XtNforeground**

This resource defines the foreground color for the widget.

See the note about the interaction of this resource with other color resources under the description of the XtNbackground resource in CORE(3W).

**XtNgrow**

Range of Values:

OL_GROW_OFF/"off"
OL_GROW_HORIZONTAL/"horizontal"
OL_GROW_VERTICAL/"vertical"
OL_GROW_BOTH/"both"

This resource controls if the widget will try to resize its window when it needs more height or width to display the text:

OL_GROW_OFF It will not resize itself.
OL_GROW_HORIZONTAL It will attempt to change its width when lines are too long for the current screen width.
OL_GROW_VERTICAL it will attempt to resize its height when the number of text lines is greater than can be displayed with the current screen height.
OL_GROW_BOTH It will attempt resizes in both dimensions.

**XtNverticalSB**

Range of Values:

TRUE
FALSE

These resources determine if the Text widget will have a scrollbar along the side.

**XtNleaveVerification**

This is the callback list used when the input focus leaves the Text widget. The call_data parameter is a pointer to an OlTextVerifyCD structure described earlier in TEXT.
**XtNleftMargin**  
Range of Values:  
\[ 0 \leq \text{XtNleftMargin} \]  
This resource is the number of pixels used for the left margin.

**XtNmaximumSize**  
Range of Values:  
\[ 0 \leq \text{XtNmaximumSize} \]  
This resource is used only if the \texttt{XtNsourceType} resource has the value \texttt{OL\_STRING\_SOURCE}. It is the maximum number of characters that can be entered into the internal buffer. If this value is not set, then the internal buffer will increase its size as needed, limited only by the space limitations of the process.

**XtNmodifyVerification**  
This callback list is called before text is deleted from or inserted into the text source. The \texttt{call\_data} parameter is a pointer to an \texttt{OlTextVerifyCD} structure described earlier.

**XtNmotionVerification**  
This callback list is called before the insertion cursor is moved to a new position. The \texttt{call\_data} parameter is a pointer to an \texttt{OlTextVerifyCD} structure described earlier in \texttt{TEXT}.

**XtNrecomputeSize**  
Range of Values:  
\[ \text{TRUE}, \text{FALSE} \]  
This resource indicates whether the \texttt{Text} widget should calculate its size and automatically set the \texttt{XtNheight} and \texttt{XtNwidth} resources. If set to \text{TRUE}, the \texttt{Text} widget will do normal size calculations that may cause its geometry to change. If set to \text{FALSE}, the \texttt{Text} widget will leave its size alone.  
This resource is ignored for each dimension that has an associated scrollbar.

**XtNrightMargin**  
Range of Values:  
\[ 0 \leq \text{XtNrightMargin} \]  
This resource is the number of pixels used for the right margin.

**XtNshowPage**  
This resource is directed to the vertical scrollbar in the \texttt{Text} widget. See \texttt{SCROLLBAR} for more detail.

**XtNsourceType**  
Range of Values:  
\[ \text{OL\_STRING\_SOURCE/"stringsrc"}, \text{OL\_DISK\_SOURCE/"disksrc"} \]  
This resource defines the type of the text source.
**XtNstring**

This resource is used only if the **XtNsourceType** resource has the value **OL_STRING_SOURCE**. This is the string to be viewed and/or edited. A copy is made into an internal buffer allocated by the Text widget. A call to **XtGetValues()** on this resource will return a copy of the internal buffer. The application program is responsible for freeing the space allocated by this copy.

**XtNtextClearBuffer**

Synopsis:

```c
void (*textClearBuffer)();
static Arg query[] = {
    { XtNtextClearBuffer, (XtArgVal)&textClearBuffer }
};
 XtGetValues(widget, query, XtNumber(query));
(*textClearBuffer)(w)
Widget w;
```

This function clears the internal buffer. After this call, all characters in the buffer have been removed.

**XtNtextCopyBuffer**

Synopsis:

```c
unsigned char *(*textCopyBuffer)(), *buf;
static Arg query[] = {
    { XtNtextCopyBuffer, (XtArgVal)&textCopyBuffer }
};
 XtGetValues(widget, query, XtNumber(query));
buf = (*textCopyBuffer)(w)
Widget w;
```

This function uses **XtMalloc()** to create space for copying the internal buffer and returns the pointer to that copy. The application is responsible for freeing the space.

**XtNtextGetInsertPoint**

Synopsis:

```c
OLTextPosition (*textGetInsertPoint)(), pos;
static Arg query[] = {
    { XtNtextGetInsertPoint, (XtArgVal)&textGetInsertPoint }
};
 XtGetValues(widget, query, XtNumber(query));
pos = (*textGetInsertPoint)(w)
Widget w;
```

This function returns the insertion position.
**XtNtextGetLastPos**

Synopsis:

```c
OlTextPosition (*textGetLastPos)(), pos;
static Arg query[] = {
    { XtNtextGetLastPos, (XtArgVal)&textGetLastPos },
};
XtGetValues(widget, query, XtNumber(query));
pos = (*textGetLastPos)(w, lastPos)
Widget w;
OltextPosition lastPos;
```

This function returns the last character position in the buffer.

**XtNtextInsert**

Synopsis:

```c
void (*textlnsert)();
static Arg query[] = {
    { XtNtextlnsert, (XtArgVal)&textlnsert },
};
XtGetValues(widget, query, XtNumber(query));
(*textlnsert)(w, string)
Widget w;
unsigned char *string;
```

This function inserts the string at the current insertion position and advances the insertion position to the end of the string.

**XtNtextReadSubString**

Synopsis:

```c
int (*textReadSubString)();
static Arg query[] = {
    { XtNtextReadSubString, (XtArgVal)&textReadSubString },
};
XtGetValues(widget, query, XtNumber(query));
(*textReadSubString)(w,startpos,endpos,target,tsize,tused)
Widget w;
OltextPosition startpos, endpos;
unsigned char *target;
int tsize, *tused;
```

This function will move characters from the buffer into the caller’s space. The caller must provide the space to copy into and its size in bytes. The routine will return the number of positions moved. The value of tused returns the number of bytes used in the target string by the move.
XtNtextRedraw
Synopsis:

```c
void (*textRedraw)();
static Arg query[] = {
    { XtNtextRedraw, (XtArgVal)&textRedraw },
};
XtGetValues(widget, query, XtNumber(query));
(*textRedraw)(w);
Widget w;
```

This function refreshes the widget’s window.

XtNtextReplace
Synopsis:

```c
OEditResult (*textReplace)(), result;
static Arg query[] = {
    { XtNtextReplace, (XtArgVal)&textReplace },
};
XtGetValues(widget, query, XtNumber(query));
result = (*textReplace)(w, startPos, endPos, text);
Widget w;
OlttextPosition startPos, endPos;
unsigned char *text;
```

This function removes text in the source from `startPos` to `endPos` and inserts the string `text` starting at `startPos`. If `startPos` and `endPos` are the same, the action is an insertion. If `text` is the empty string, the action is a deletion.

XtNtextSetInsertPoint
Synopsis:

```c
void (*textSetInsertPoint)();
static Arg query[] = {
    { XtNtextSetInsertPoint, (XtArgVal)&textSetInsertPoint },
};
XtGetValues(widget, query, XtNumber(query));
(*textSetInsertPoint)(w, position);
Widget w;
OlttextPosition position;
```

This function sets the insertion point.
Synopsis:

```c
void (*textUpdate)();
static Arg query[] = {
    { XtNtextUpdate, (XtArgVal)&textUpdate }
};
XtGetValues(widget, query, XtNumber(query));
(*textUpdate)(w, status)
Widget w;
Boolean status;
```

This function turns the widget's screen updating function on and off. If the application needs to make a sequence of source change calls, a call to OlTextUpdate(FALSE) will prevent screen flash. After the sequence of calls the application calls OlTextUpdate(TRUE) to update the window and resume normal updating. Note that it is not necessary to turn off updating for functions that only get values from the widget, nor is it necessary to turn it off if the application only makes one call that changes the widget.

XtNtopMargin

Range of Values:

0 ≤ XtNtopMargin

This resource is the number of pixels used for the top margin.

XtNvviewHeight

Range of Values:

0 ≤ XtNviewHeight

This resource gives the preferred height, in lines, of the text pane. If a nonzero value is given, the corresponding XtNheight resource is computed by converting this number to pixels and adding the thickness of any scrollbar and border that appears. In this case, any value in the XtNheight resource is overwritten.

If a zero value is given in the XtNviewHeight resource, the XtNheight resource is used as an estimate. The text pane is sized to show an integral number of lines, such that the overall height of the Text widget is less than or equal to XtNheight, if possible. However, the text pane is always large enough to show at least one line and is no shorter than the minimum scroll bar size.

If neither the XtNviewHeight resource nor the XtNheight resource is set, or both are set to zero, the text pane is made as small as possible, limited as described above.

XtNtraversalOn

Range of Values:

TRUE
FALSE

This resource specifies whether this widget is selectable during traversal.
**XtNwrap**

Range of Values:

- **TRUE**
- **FALSE**

This resource specifies how the widget displays lines longer than the screen width. When set to **FALSE**, the lines may extend off screen to the right. When set to **TRUE**, the lines will be wrapped at the right margin, with the position determined by the resource *XtNwrapBreak*.

**XtNwrapBreak**

Range of Values:

- **OL_WRAP_ANY**/"wrapany"
- **OL_WRAP_WHITE_SPACE**/"wrapwhitespace"

This resource specifies how the wrap position is determined. When set to **OL_WRAP_ANY**, the wrap will happen at the character position closest to the right margin. When set to **OL_WRAP_WHITE_SPACE**, the wrap will happen at the last white space before the right margin. If the line does not have white space, it will be wrapped as **OL_WRAP_ANY**.

**SEE ALSO**

TextEdit(3W)
The italicized words refer to entries within this glossary.

**ADJUST**

The mouse button or keyboard equivalent used to adjust a selection (cf. SELECT); usually the middle button on a right hand mouse.

**anchor**

Either end of a Scrollbar widget or a Slider widget. The part of the widget that remains fixed while the elevator or drag box moves along.

**button**

Generic term for any of several widgets, specifically RectButton widgets and OblongButton widgets. The RectButtons are implicitly defined in flattened widgets, as well. A button, when pressed usually initiates certain actions, like popping up a menu or executing an application routine.

**cable**

In a Scrollbar widget, the cable is the "line" on which the elevator moves. One end of the cable is connected to the anchor and the other is connected to the elevator.

**callback**

A callback routine is a routine written by an application programmer and associated with a specific widget resource. The callback routine is invoked as a result of a specific activity associated with that widget (that is, the widget calls back the program via that routine). For example, the XtNselect resource contains the name of the callback routine that is entered when a button is pushed or when a CheckBox is selected; the XtNVerification resource contains the name of the callback routine to invoke when a TextField widget is exited. The act of associating the name of a callback routine with a widget resource is called registration.

**click**

The act of pressing and releasing a mouse button without moving the mouse pointer more than a few pixels.

**click-move-click**

A method of user interaction with a set of objects where the user clicks MENU to display the objects, moves the pointer over the one of interest, then clicks MENU or SELECT to select or activate the object.
Glossary

composite widget  See widget. A widget that is a parent of other widgets, that physically contains other widgets.

container  A widget that defines a region that holds zero or more sub-objects of a given type.

control area  The area located directly under the header of a window. It is used to display “command buttons,” if the application in the window provides them.

dimmed  A visual effect on an object. A control, such as a button, is dimmed if its visible manifestation represents the state of just one of several objects that are in inconsistent states. When such a control is manipulated (for example, by clicking SELECT over the button), it is no longer dimmed because the manipulation sets the state for all the objects.

double click  To press and release a mouse button twice in succession.

drag area  In a Scrollbar widget, the drag area is the center portion of the elevator that is moved by the mouse.

drag box  In a Slider widget, the drag box is the portion of the slider that is moved by the mouse.

dragging  The act of moving the pointer while a mouse button or keyboard equivalent is pressed.

elevator  The center portion of a Scrollbar widget; that part which moves along the cable.

flat widget  See widget. A single widget that maintains a collection of similar user-interface components that together give the appearance and behavior of many widgets.

flattened widget  Same as flat widget.

focus  To specify a particular area of the screen. (See input focus and keyboard focus).

gadget  A windowless object; an object that could be defined as a widget but, instead, is defined as having its parent’s window resources.
grab
To position the mouse pointer on a resize corner and take hold of it for the purpose of resizing the window.

HELP
The mouse button or keyboard equivalent used to bring up an OPEN LOOK Help window.

highlighted
A visual indication that an object is in a special state. For two-color ("monochrome") objects, the colors are exchanged. Multi-color objects cannot be highlighted.

icon
The state where an application base window and all its pop ups are removed from the screen and replaced with a single, small figure that represents the application. Icon also refers to this small figure.

input focus
To have the cursor on a particular field, designating that field as "next."

instance
A specific realization of a widget; one particular widget as opposed to a class of widgets.

keyboard focus
The area of the screen that will accept the next input from the keyboard.

MENU
The mouse button or keyboard equivalent used to display (pop up) a menu.

menu
When unqualified, any of the three states of an OPEN LOOK menu: popup menu, stay-up menu, or pinned menu.

pane
The rectangular area within a window where an application displays text or graphics.

pinned menu
An OPEN LOOK menu that has a pushpin that is "in." This menu behaves much like a control area in a pinned command window.

pixel
An addressable point on the screen.

pixmap
A bitmap of an area of the screen stored within the program. A "pixmap" is also a defined data type in the Xt Intrinsics.
Glossary

pointer
The screen representation of the location of the mouse or equivalent.

pop up
As a noun, pop up is a generic term referring to an OPEN LOOK window other than the base window. As a verb, this phrase is the act of making a menu or OPEN LOOK popup window visible. As an adjective, it is used to refer to a window that can be popped up and is spelled with or without a dash, as in “popup menu” or “pop-up menu.”

popup menu
An OPEN LOOK menu that was brought up by pressing MENU. While MENU remains pressed, the menu remains a popup menu and operates in a press-drag-release mode.

press
The act of pressing a mouse button or keyboard key. This is distinct from the act of releasing the button or key, so that both can be discussed separately. Thus “press SELECT” means to press, but not release, the SELECT mouse button or keyboard equivalent key.

press-drag-release
A method of user interaction with a set of objects where the user presses MENU to display the objects, drags the pointer over the objects until it is over the one of interest, then releases MENU to select or activate the object.

primitive widget
See widget. A widget that does not have any child widgets; one that either performs a specific action, allows input or allows output.

push a button
The act of moving the pointer to a button widget and then selecting the button.

pushpin
A screen object that is part of a popup menu. It can be pointed to and selected. When it is first selected, it is “pushed in” and causes the menu to stay up after the user moves out of it. When it is again selected, it is pulled out and the menu pops down.
realized In the context of the X Toolkit Intrinsics, the point at which all the data structures of a widget have been allocated. Windows and other information are not created when the widget is created with the XtCreateWidget routine, but are created in a later call to XtRealizeWidget on the widget itself or on an ancestor widget.

register, registration To make a routine name known to the API. When the application programmer develops a callback routine, that routine needs to be registered when the widget is created so that it can be properly invoked.

release The act of releasing a pressed button or keyboard key, as in "release MENU."

resize corners Hollow, L-shaped symbols located on all four corners of a window which, when grabbed, are used to change the size of the window.

resource An attribute of a widget or a widget class. A resource is a named data value in the defining structure of a widget.

screen The surface on your computer monitor where information is displayed.

select To move the pointer to an object and press the SELECT mouse button. The result is to initiate either an application action or a change in the window content or structure.

SELECT The mouse button or keyboard equivalent used to select and move an object, manipulate an OPEN LOOK control, or set the input focus.

stay-up menu An OPEN LOOK menu that was brought up and made to stay on the screen for one round of use. The controls in this menu behave like controls in an unpinned command window, except that the menu is removed from the screen even if nothing is selected from the menu.
sub-object: A sub-object is the equivalent of a primitive widget contained in a flattened widget. In a Flat Exclusives or NonExclusives widget, the sub-objects are the equivalents of RectButtons. In a Flat CheckBox, the sub-objects are the equivalents of CheckBox widgets.

toggle: This is an action performed on an object with two states; it is the switching from one state to the other.

whitespace: Typically any characters that have no visible form. For this toolkit, these characters are space, tab, newline, and carriage return.

widget: A specific example or realization of a widget class.

widget class: A collection of code and data structures that provides a generic implementation of a part of a look-and-feel.

window: A work area on the screen that you use to run and display an application.
Index

A
API 2: 1
application defaults 2: 7
Application Programmer Interface 2: 1
application resource values 2: 8
assigning widget values 2: 8
atom names 4: 3

B
base windows, multiple 2: 49

C
callback
definition 2: 2
function 3: 8, 10, 14
parameters 3: 10
registering 2: 2
client
definition 4: 3
message 4: 2
color representation 2: 15
command line options 2: 7
command window 2: 42
communication conventions
file manager 4: 21
workspace manager 4: 21
compilation command, prototype 3: 4
compile command, on-line programs 3: 6
composite child widget 3: 16
composite widget 2: 2
container 2: 3
convenience routines 2: 49
core widget, definition 2: 9
customized window decorations 4: 8

d
default font 2: 17
default window decorations 4: 7
directory
/usr/X/include/X11 3: 4
/usr/X/include/Xol 3: 4
/usr/X/lib 3: 4

e
efficiency 4: 4
error handling 2: 46
errors, warnings 2: 46
events 4: 2
exclusives widget, sample 3: 18

F
file manager 4: 4
communication conventions 4: 21
flat widget, definition 2: 3
flattened widgets 2: 3
focus model
globally active 4: 19
locally active 4: 19
no input 4: 20
passive 4: 19
font, default 2: 17
full size window 4: 27
Index

G

gadget
  ButtonStack 2: 22
  definition 2: 4
  OblongButton 2: 19

gadget class 2: 4

globally active focus model 4: 19

graphical user interface, definition 4: 1

H

header files 3: 8, 10, 14
help, text 2: 46

I

icon mask 4: 15
icon pixmap 4: 15
icon window 4: 15
include files 3: 4-5
input focus 2: 47, 4: 19-20
  transfer 4: 13
internal defaults 2: 7
intrinsics 2: 1

L

library
  /usr/X/lib/libXt.a 3: 4
  /usr/X/libX11.a 3: 4

locally active focus model 4: 19
long jump 3: 36

M

main event loop 3: 8
manager
  definition 4: 3
  file 4: 4
  session 4: 4
  window 4: 4
  workspace 4: 4

mapped, definition 2: 11
multiple base windows 2: 49

N

naming conventions 2: 5
no input focus model 4: 20

O

object, definition 2: 1
object libraries 3: 4
Object Oriented Programming 3: 1
object-specific defaults 2: 7
obtaining widget values 2: 8

_OL_DECOR_ADD property 4: 8
_OL_DECOR_DEL property 4: 8
_OL_FM_QUEUE property 4: 21
_OL_FM_REPLY property 4: 21
OlInitialize routine 2: 45

_OL_PIN_STATE property 4: 10-11
_OL_WIN_ATTR property 4: 6
_OL_WIN_BUSY property 4: 12
_OL_WIN_COLORS property 4: 11
olwsm program 2: 7

_OL_WSM_QUEUE fields 4: 21-26
_OL_WSM_QUEUE property 4: 21
_OL_WSM_REPLY property 4: 21
on-line programs
  compile commands 3: 6
  pathnames 3: 6
OPEN LOOK
  conventions 4: 1
  mini-protocol 4: 3
  Workspace Manager program 2: 7
overlapping resources 2: 6
override-redirect window 4: 27
overriding values 2: 8

P
  pane 2: 26
  passive focus model 4: 19
  pathnames, on-line programs 3: 6
  pinned menus 4: 27
  pixel densities 2: 16
  PPPosition flag 4: 14
  primitive widget 2: 2
  procedure names 2: 5
  properties 4: 2
  property windows 2: 42
  protocol, extensibility 4: 4

R
  resource
    definition 2: 2
    server 2: 7
    values 2: 7
    widget 2: 6
  resource class 2: 6
  resource name 2: 5
  routine, OlInitialize 2: 45
  routines
    convenience 2: 49
  error 2: 46

S
  screen formats 2: 16
  server resource 2: 7
  session manager 4: 4
  standard font 2: 17
  structure member names 2: 5
  sub-object 2: 3
  switch statement 3: 10

T
  type names 2: 5
  type-defined 2: 5

U
  uninterpreted data 4: 2
  unrelated resources 2: 6
  USPosition flag 4: 14
  /usr/X/include/X11 directory 3: 4
  /usr/X/include/Xol directory 3: 4
  /usr/X/lib directory 3: 4
  /usr/X/lib/libXt.a library 3: 4
  /usr/X/lib/libXol.a library 3: 4

W
  widget
    AbbrevStack 2: 23
    BulletinBoard 2: 30
    ButtonStack 2: 22
    Caption 2: 34
    CheckBox 2: 21
Index

class 2: 6
composite 2: 2
ControlArea 2: 31
core 2: 9
definition 2: 1
Exclusives 2: 35
FooterPanel 2: 34
Form 2: 33
Menu 2: 43
Nonexclusives 2: 36
Notice 2: 41
OblongButton 2: 19
PopupWindow 2: 42
primitive 2: 2
RectButton 2: 20
resources 2: 6
Scrollbar 2: 26
ScrolledWindow 2: 37
ScrollingList 2: 38
Slider 2: 24
StaticText 2: 28
stub 2: 27
Text 2: 28
TextField 2: 29
widget class 2: 4
widgets
    composite 2: 30
    packed 2: 46
window colors 4: 11
window decorations
    customized 4: 8
    default 4: 7
    standard 4: 6
window footer 4: 9
window groups 4: 16-18
window manager 4: 4
WM_DELETE_WINDOW protocol 4: 15
WM_HINTS property 4: 15

WM_NORMAL_HINTS property 4: 14
WM_SAVE_YOURSELF protocol 4: 15
WM_TAKE_FOCUS message 4: 13
WM_TAKE_FOCUS protocol 4: 16
workspace manager 4: 4
    communication conventions 4: 21

X

X Window System, description 4: 1
X11 atoms 4: 3
X11 protocol 4: 1
XENVIRONMENT variable 2: 7
Xlib 2: 1
Xt Intrinsics 2: 1
XtGetValues function 2: 8
XtSetValues function 2: 8
The OPEN LOOK® Graphical User Interface (GUI) is a user-friendly front-end to the UNIX® operating system and is available on many major workstation platforms. The latest release features 3D visuals and mouseless operation. The OPEN LOOK GUI Toolkit (often referred to as OLIT or Xt+) is based on Release X11R4 of the MIT Intrinsics.

Written for experienced C language programmers, this manual is a comprehensive step-by-step guide to using the OLIT toolkit and covers:

- Xt Intrinsic functions
- widget resources
- how to implement OPEN LOOK GUI functions
- programming procedures
- conventions for using the toolkit

This manual also features advice for screen layout, offers hints for creating and managing widgets and callbacks, and includes a complete reference manual to OPEN LOOK GUI widgets, gadgets, convenience routines, and the high-performance flattened widgets. A sample program using all the widgets is included.

UNIX and OPEN LOOK are registered trademarks of UNIX System Laboratories, Inc.